Automatische Konfiguration der Bewegungssteuerung von Industrierobotern

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

D i s s e r t a t i o n

von

Dipl.–Inform. Michael Wenz

aus Karlsruhe

Tag der mündlichen Prüfung:	17. Juni 2008
Erster Gutachter:	Prof. Dr.–Ing. Heinz Wörn
Zweiter Gutachter:	Prof. Dr.rer.nat. Uwe Brinkschulte

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Prozessrechentechnik, Automation und Robotik der Universität Karlsruhe (TH). Ein großer Teil der Arbeit wurde innerhalb des von der Deutschen Forschungsgemeinschaft (DFG) geförderten Teilprojekts „Organic Robot Control“ im Rahmen des Schwerpunktprogramms 1183 „Organic Computing“ durchgeführt. Die Entwicklung von selbstkonfigurierenden Systemen ist ein aktuelles Forschungsthema. Ein relevantes Anwendungsgebiet ist die automatische, schnelle und zuverlässige Konfiguration der Bewegungssteuerung von Industrierobotern.

Mein besonderer Dank gilt Herrn Prof. Dr.–Ing. Heinz Wörn für die Betreuung der vorliegenden Dissertation. Er hat mich stets unterstützt und mir die nötigen Freiräume für Forschung und selbstständiges wissenschaftliches Arbeiten gegeben. Mein Interesse für Bewegungssteuerungen wurde von ihm geweckt und viele seiner Kenntnisse auf diesem hochinteressanten Gebiet gab er an mich weiter. Für die Übernahme des Korreferates und für die Durchsicht der Arbeit möchte ich Herrn Prof. Dr.rer.nat. Uwe Brinkschulte danken. Ebenfalls danke ich Frau Prof. Dr.–Ing. Tanja Schultz und Herrn Prof. Dr.–Ing. Jürgen Beyerer, die beide als mündliche Prüfer in meiner Disputation fungierten. Den Vorsitz der Prüfungskommission hatte Herr Prof. Dr.–Ing. Roland Vollmar übernommen.

An dieser Stelle danke ich auch den Kolleginnen und Kollegen am Institut für die fachlichen Gespräche und Diskussionen, die wertvolle Denkanstöße lieferten und mir sehr geholfen haben. Des Weiteren gilt mein Dank allen von mir betreuten Studenten, die im Rahmen von Praktika und Studienarbeiten sehr zum Gelingen dieser Arbeit beigetragen haben. Für die vielseitige Unterstützung und das gute Arbeitsklima an diesem Institut möchte ich mich bedanken.

Michael Wenz — Karlsruhe, September 2008

Inhaltsverzeichnis

Symbolverzeichnis

Selten benutzte Symbole und abweichende Bedeutungen werden im Text erläutert. Physikalische Größen werden entsprechend des Internationalen Einheitensystems (SI) angegeben ($N = kg \cdot m/s^2$, $J = kg \cdot m^2/s^2$). Die Einheit von manchen Variablen hängt von dem Gelenktyp ab. In der folgenden Tabelle wird zu jedem Symbol das Kapitel beziehungsweise der Abschnitt angegeben, in dem das Symbol zum ersten Mal verwendet wird.

Physikalische Größen

Symbol	*Bedeutung*	*SI–Einheit*	*Abschnitt*
n	Anzahl der Gelenkachsen	–	4
q_i	allgemeine Gelenkvariable	$[rad]$ $[m]$	4
α	Eulersche Orientierungswinkel	$[rad]$	4
β	Eulersche Orientierungswinkel	$[rad]$	4
γ	Eulersche Orientierungswinkel	$[rad]$	4
θ_i	Gelenkwinkel	$[rad]$	4.1
d_i	Hebelverschiebung	$[m]$	4.1
a_i	Hebellänge	$[m]$	4.1
α_i	Hebelverschiebung	$[rad]$	4.1
t	Zeit	$[s]$	5
$\dot{q}_i$	Geschwindigkeit des i–ten Gelenks	$[\frac{rad}{s}]$ $[\frac{m}{s}]$	5.1
$\ddot{q}_i$	Beschleunigung des i–ten Gelenks	$[\frac{rad}{s^2}]$ $[\frac{m}{s^2}]$	5.1
$\mathcal{L}$	Lagrange Funktion	$[J]$	5.1
$\mathcal{K}$	Kinetische Energie	$[J]$	5.1
$\mathcal{P}$	Potentielle Energie	$[J]$	5.1
I_{xx}	Massenträgheitsmoment bezüglich x–Achse	$[kg \cdot m^2]$	5.1
I_{yy}	Massenträgheitsmoment bezüglich y–Achse	$[kg \cdot m^2]$	5.1
I_{zz}	Massenträgheitsmoment bezüglich z–Achse	$[kg \cdot m^2]$	5.1
I_{xy}	Deviationsmoment bezüglich xy–Ebene	$[kg \cdot m^2]$	5.1
I_{xz}	Deviationsmoment bezüglich xz–Ebene	$[kg \cdot m^2]$	5.1
I_{yz}	Deviationsmoment bezüglich yz–Ebene	$[kg \cdot m^2]$	5.1
g	Erdbeschleunigung	$[\frac{m}{s^2}]$	5.1
m_i	konzentrierte Masse des i–ten Manipulatorglieds	$[kg]$	5.1
τ_i	Moment / Kraft, die im i–ten Gelenk wirkt	$[J]$ $[N]$	5.1
σ_i	gelenkspezifischer Kennungsparameter	–	5.2
q_{min}	untere Gelenkkoordinatenbegrenzung	$[rad]$ $[m]$	6.3
q_{max}	obere Gelenkkoordinatenbegrenzung	$[rad]$ $[m]$	6.3
v_{max}	maximale Gelenkgeschwindigkeit	$[\frac{rad}{s}]$ $[\frac{m}{s}]$	6.3
a_{max}	maximale Gelenkbeschleunigung	$[\frac{rad}{s^2}]$ $[\frac{m}{s^2}]$	6.3
j_{max}	maximaler Gelenkruck	$[\frac{rad}{s^3}]$ $[\frac{m}{s^3}]$	6.3

Vektoren und Matrizen

Symbol	*Bedeutung*	*Dimension*	*Abschnitt*
$\vec{q}$	Gelenkvariablenvektor	$n \times 1$	4
$T_{i,j}$	homogene Transformationsmatrix	4×4	4.1
$R_{i,j}$	Rotationsmatrix	3×3	4.1
ξ_i	Twist–Koordinate	6×1	4.5
$\hat{\xi}_i$	Twist	4×4	4.5
$so(3)$	Menge der antisymmetrischen 3×3–Matrizen	3×3	4.5
$SO(3)$	spezielle orthogonale Gruppe	3×3	4.5
$se(3)$	Menge der Twists	4×4	4.5
$SE(3)$	spezielle euklidische Gruppe	4×4	4.5
$\mathcal{J}$	Jacobi–Matrix	$6 \times n$	4.6
$\mathcal{J}^+$	Pseudoinverse der Jacobi–Matrix	$n \times 6$	4.6
$I_{n,n}$	Einheitsmatrix	$n \times n$	5.1
${}^A\mathcal{I}$	Trägheitstensormatrix	3×3	5.1
M_i	verallgemeinerte Massenmatrix für das i–te Glied	6×6	5.1
$\mathcal{J}_i$	partielle Jacobi–Matrix bis zum i–ten Glied	$6 \times n$	5.1
$\vec{r}_i$	relative Schwerpunktkoordinaten des i–ten Glieds	3×1	5.1
$\vec{\tau}$	Kraft– / Momentenvektor	$n \times 1$	5.1
$\mathcal{M}$	Massenmatrix	$n \times n$	5.1
$\mathcal{C}$	Matrix der Zentrifugal– und Corioliskräfte	$n \times n$	5.1
$\mathcal{G}$	Gravitationsvektor	$n \times 1$	5.1
$\mathcal{CF}$	Zentrifugalmatrix	$n \times n$	5.1
$\mathcal{CO}$	Coriolismatrix	$n \times \frac{n \cdot (n-1)}{2}$	5.1
$\mathcal{F}_i$	Kraftvektor im Schwerpunkt des i–ten Glieds	3×1	5.2
$\mathcal{N}_i$	Momentvektor im Schwerpunkt des i–ten Glieds	3×1	5.2
$\vec{\omega}_i$	Winkelgeschwindigkeitsvektor	3×1	5.2
$\dot{\vec{\omega}}_i$	Winkelbeschleunigungsvektor	3×1	5.2
$\dot{\vec{v}}_i$	Linearbeschleunigungsvektor	3×1	5.2
$\vec{p}_i$	Positionsvektor	3×1	5.2
$\dot{\vec{v}}_{ci}$	Beschleunigungsvektor des i–ten Schwerpunkts	3×1	5.2
$\mathcal{I}_{ci}$	Trägheitstensor im Basiskoordinatensystem	3×3	5.2
$\vec{f}_i$	Kraftvektor, den Glied i auf Glied $i-1$ ausübt	3×1	5.2
$\vec{n}_i$	Momentvektor, den Glied i auf Glied $i-1$ ausübt	3×1	5.2
$\hat{q}$	Quaternion	4×1	6.3

Kapitel 1

Einführung

1.1 Motivation und Problembeschreibung

Die Steuerung eines Industrieroboters kapselt verschiedene Softwaresysteme für einzelne Komponentengruppen (Antriebe, Sensoren, Technologiesteuerung, Peripheriekomponenten) auf die über Bussysteme zugegriffen wird. Solche Softwaresysteme sind beispielsweise die Bewegungssteuerung für einen oder mehrere Roboter, Sensordatenverarbeitungssysteme, applikations– und technologieabhängige Bedienoberflächen sowie variantenreiche Technologiesteuerungen für Schweißen, Kleben, Handhaben, Montieren oder Bearbeiten. Die Bewegungssteuerung eines Roboters setzt sich aus den Komponenten für den Interpreter, Interpolationsvorbereitung, Interpolation, inverse Kinematik, inverse Dynamik, Kaskadenregelung und anderes mehr zusammen. Die Hauptaufgabe der Bewegungssteuerung ist die Entgegennahme und Abarbeitung von Roboterprogrammen und Bewegungsbefehlen. Anhand der vorgegebenen Zielstellungen des Endeffektors erzeugen die Interpolationskomponenten entsprechende zeitliche Zwischenstellungen. Die Aufgabe der inversen Kinematik ist es, Endeffektorlagen im kartesischen Raum in den Gelenkwinkelraum zu transformieren und dadurch die Sollwerte für die Antriebsregelkreise zu bestimmen. Um das Bewegungsverhalten zu verbessern, erfolgt eine Vorsteuerung des Sollstroms mit der inversen Dynamik, die parallel zu den Lage– und Geschwindigkeitsreglern geschalten wird.

Eine wichtige Herausforderung bei der Entwicklung von Steuerungssoftware für Roboter ist die Sicherstellung einer hohen Qualität der Software. Dazu gehört auch die garantierte Einhaltung vorgegebener Zeitbedingungen. Durch fehlerhafte Steuerungssoftware können erhebliche Schäden entstehen oder sogar Menschen gefährdet werden. Aufgrund der stetig ansteigenden Anzahl von Robotertypen ist die Beherrschung der zunehmenden Variantenvielfalt eine weitere wesentliche Herausforderung. Der Entwickler und Projektierer steht vor einer nahezu unüberschaubaren Vielfalt an Komponenten sowie an Kombinations– und Konfigurationsmöglichkeiten. Mit dem Umfang an Steuerungsfunktionen und der Anzahl möglicher Konfigurationen steigen auch die Anforderungen an die einzusetzenden Softwaretechnologien. Die Software zur Bewegungssteuerung von Industrierobotern weist bei verschiedenen Robotertypen erhebliche Unterschiede auf. Bisher war die Entwicklung von Steuerungssoftware an einen bestimmten Robotertyp gekoppelt. Jeder Roboter hat damit seine eigene spezielle Software, die genau die Fähigkeiten dieser Baureihe abdeckt. Die Erweiterbarkeit, Portierbarkeit und Wartbarkeit von Steuerungssoftware ist damit eine langwierige und schwierige Aufgabe. Insbesondere die Wiederverwendbarkeit von Robotermodellen ist kaum möglich. Deswegen soll dieser Softwareentwicklungsvorgang im Rahmen dieser Arbeit in wesentlichen Teilen automatisiert und sehr vereinfacht werden.

1.2 Zielsetzung und Beitrag der Arbeit

Ziel dieser Arbeit ist es, den Vorgang der Entwicklung der Bewegungssteuerung von Robotern zu automatisieren, welcher bisher nur von menschlichen Experten und von Hand erledigt werden konnte. Hierfür wurde ein neues Werkzeug mit graphischer Benutzeroberfläche zur automatisierten Konfiguration komponentenbasierter Software auf Basis einer Beschreibung der mechanischen Struktur und Eigenschaften von Industrierobotern entwickelt. Damit stehen dem Benutzer vielfältige Auswahl– und Kombinationsmöglichkeiten zur Verfügung, zum Beispiel Anzahl und Typen der Gelenkachsen, Bewegungsmöglichkeiten der Haupt– und Nebenachsen, Anordnung der Achsen, geometrische Abmessungen, zu unterstützende Interpolationsarten, Beschleunigungsprofilart, Interpolationstakt, maximale Geschwindigkeit bei der Interpolation und vieles mehr. Das System unterstützt den Benutzer bei der Auswahl zulässiger Konfigurationen. Diese Arbeit zielt insbesondere darauf ab, die Bewegungssteuerungen einer Vielzahl unterschiedlicher Robotertypen und kinematischer Aufbauarten durch eine gemeinsame Referenzarchitektur zu integrieren. Die automatische Konfiguration der Bewegungssteuerung stellt ein Softwaregenerierungs– und Kompositionsproblem dar, das anhand von Zielvorgaben und Randbedingungen, zum Beispiel Echtzeitanforderungen, gelöst werden muss. Aufgrund von starken Nichtlinearitäten sowie von räumlich–geometrischen Mehrdeutigkeiten und Singularitäten handelt es sich bei der Bestimmung von Robotermodellen für die Bewegungssteuerung um eine sehr komplexe und wissenschaftlich anspruchsvolle Problemstellung.

Die in dieser Arbeit betrachtete Bewegungssteuerung wird durch das flexible Konfigurieren von Softwarekomponenten für die jeweilige Robotergeometrie erzeugt. Für einen modularen Aufbau wurden zwischen den einzelnen Komponenten einheitliche und roboterunabhängige Schnittstellen definiert. Der erste Teil dieser Arbeit konzentriert sich auf die vollautomatische Bestimmung der für die Bewegungssteuerung notwendigen kinematischen und dynamischen Robotermodelle. Hierzu gehört auch die analytische Herleitung der Jacobi–Matrix von Manipulatoren. Im zweiten Teil der Arbeit werden wichtige Algorithmen zur Trajektoriengenerierung und ein herstellerunabhängiger Interpreter für Roboterprogramme entwickelt sowie die einzelnen Softwarekomponenten in einer Referenzarchitektur integriert. Es wurde ein spezielles System entwickelt, mit dem sehr effiziente und schnell berechenbare Robotermodelle generiert werden können. Die Modelle können entweder als LaTeX–Dokumentation oder als optimierter C/C++ Code mit hoher Recheneffizienz ausgegeben werden. Beispielsweise liegt der Zeitbedarf für die numerische Ausführung des kinematischen Modells eines sechsachsigen Gelenkarmroboters im Mikrosekundenbereich. Außer für den Realzeiteinsatz sind die Modelle auch für die Simulation und den Neuentwurf von Manipulatoren verwendbar. Darüber hinaus können die Modelle sehr gut in Hardware parallelisiert werden, zum Beispiel auf einem FPGA.

Um Manipulatoren mit einer beliebigen Anzahl von Gelenkachsen und Kombinationen steuern zu können, wird zunächst der Manipulator anhand der Benutzereingabe nach der Denavit–Hartenberg Konvention dargestellt. Anschließend wird das direkte und inverse kinematische Problem gelöst. Die direkte Kinematik berechnet die Position und Orientierung des Endeffektors in Abhängigkeit der Gelenkvariablenwerte. Für die Bewegungssteuerung ist die inverse Kinematik von größerer Bedeutung. Für die Ableitung exakter algebraischer Kinematikmodelle wurde ein regelbasiertes System entwickelt, mit dem sehr große, trigonometrische Gleichungssysteme, bestehend aus mehreren hundert Gleichungen, effektiv gelöst werden können. Die kinematischen Gleichungen werden als ge-

schachtelte Symbolstrukturen beziehungsweise als Fakten im Arbeitsspeicher dargestellt. Die Lösung der Gleichungen erfolgt mit einer neu entwickelten Wissensbasis, die aus einer Sammlung von komplexen Produktionsregeln besteht. Die linke Seite jeder Regel repräsentiert eine oder mehrere trigonometrische Gleichungen und die rechte Seite die dazugehörende analytische Lösung. Die Auswertung von Regeln und Fakten erfolgt mit dem RETE–Algorithmus. Für die meisten seriellen Kinematiken dauert das Aufstellen und Lösen der Gleichungen nur wenige Sekunden. Ferner wurde untersucht, wie besonders komplizierte Robotergeometrien, zum Beispiel Lackierroboter, bei welchen sich die Handachsen nicht in einem Punkt schneiden, mit Hilfe algebraischer Eliminationsmethoden analytisch gelöst werden können. Hierfür werden die trigonometrischen Gleichungen durch eine rationale Parametrisierung in polynomiale Gleichungen umgewandelt und anschließend die Gröbnerbasis bestimmt.

Um die generierten Robotermodelle zu vergleichen und zu bewerten, werden Manipulatoren neben den homogenen 4×4 Transformationsmatrizen auch mit Twist–Koordinaten modelliert. Für das direkte kinematische Modell und für die Jacobi–Matrix einer seriellen Kinematik werden bei beiden Verfahren exakt dieselben analytischen Ausdrücke automatisch generiert. Durch dieses diversitäre Vorgehen wird eine sehr hohe Softwarequalität erzielt und damit eine wesentliche Anforderung bei der Entwicklung von Echtzeitsystemen erfüllt. Zur Bestimmung der inversen Kinematik mit Twist–Koordinaten werden die Ausdrücke der direkten Kinematik auf eine Menge von kanonischen Teilproblemen zurückgeführt. Die analytischen Lösungen dieser Teilprobleme wurden erstmals in dieser Arbeit durch algebraische Umformungen bestimmt. Mit diesen Teilproblemen wird auch erkannt, ob es sich um einen kinematisch redundanten Manipulator handelt, für den es keine geschlossene Lösung gibt. In diesem Fall wird eine iterative Näherungslösung angewendet.

Darüber hinaus wurde ein System entwickelt, um Modelle der Roboterdynamik automatisch zu erstellen und dabei die inverse und direkte Dynamik zu lösen. Für die Bewegungssteuerung relevant ist die inverse Dynamik, die die Kräfte und Momente berechnet, die notwendig sind, um eine bestimmte Bewegung durchzuführen. Das Dynamikmodell in Matrixdarstellung beinhaltet die Massenmatrix, Coriolismatrix und Zentrifugalmatrix sowie den Gravitationsvektor und gegebenenfalls Reibungskräfte. Ausgehend von der Beschreibung des Manipulators werden die Bewegungsgleichungen automatisch erzeugt. Hierbei wurden mehrere Möglichkeiten verfolgt, und zwar Lagrange Verfahren, Newton'sche Axiome und Kane Formalismus. Beim Lagrange Verfahren wird die potentielle und die kinetische Energie für jedes Gelenk bestimmt, aufsummiert und gemäß den Lagrange Gleichungen differenziert und umgeformt. Beim Newton–Euler Verfahren wird in einer Vorwärtsiteration, die an der Roboterbasis beginnt und am Endeffektor endet, die Kräfte und Momente in den Schwerpunkten der Manipulatorglieder berechnet. Anschließend wird in einer Rückwärtsiteration, die am Endeffektor beginnt und an der Roboterbasis endet, die Kräfte und Momente, die ein Glied auf sein Vorgängerglied ausübt, berechnet. Die Komplexität der symbolisch generierten Differentialgleichungen nimmt mit steigender Gelenkanzahl des Roboters sehr zu. Deshalb wurde das Newton–Euler Verfahren in dieser Arbeit dahingehend modifiziert, dass die Gleichungen in insgesamt vier Iterationen bestimmt werden, um sie leichter umformen und vereinfachen zu können. In der ersten Iteration werden die Terme bezüglich des Gravitationsvektors, in der zweiten bezüglich der Massenmatrix, in der dritten bezüglich der Zentrifugalmatrix und in der vierten bezüglich der Coriolismatrix bestimmt. Durch dieses Vorgehen werden übersichtlichere Gleichungen generiert, als wenn alle Gleichungen in einer einzigen Iteration berechnet würden.

Damit beliebige Roboterprogramme von der Bewegungssteuerung ausgeführt werden können, wurde ein Roboterinterpreter nach der INDUSTRIAL ROBOT LANGUAGE (IRL), die in DIN 66312 spezifiziert ist, entwickelt. Insbesondere wurde keine herstellerspezifische Roboterprogrammiersprache verwendet, die nicht für Roboter anderer Hersteller verwendet werden kann. Bei IRL handelt es sich um eine PASCAL–ähnliche Programmiersprache mit Konstrukten für Bewegungsanweisungen und robotikspezifischen Datentypen. Der IRL–Interpreter arbeitet Roboter– und Bewegungsbefehle kontinuierlich ab und kommuniziert mit den Komponenten für die Trajektoriengenerierung. Ergänzt wurde das System mit einer Vielzahl von Algorithmen sowohl für Punkt–zu–Punkt Bewegungen als auch für kartesische Bahnsteuerung. Um zeitabhängige Zwischenstellungen zu berechnen werden zum Beispiel unterstützt: asynchrone, synchrone und vollsynchrone Bewegungen ohne und mit Überschleifen, trapez– und sinusförmige Beschleunigungsprofile, lineare und zirkulare Bahnen ohne und mit Nachführung der Orientierung sowie raumbezogene und bahnbezogene Orientierungsinterpolation. Die Interpolation von Orientierungen im kartesischen Raum erfolgt mit Quaternionen, da der Rechenaufwand niedriger ist als bei Rotationsmatrizen und im Gegensatz zu Euler Winkeln keine Mehrdeutigkeiten auftreten können.

Schließlich erfolgte eine Integration der einzelnen Softwarekomponenten in ein Open–Source–System für die Bewegungssteuerung von Robotern. Insgesamt betrachtet wurde der Entwicklungszyklus von Steuerungssoftware für Roboter durch diese Arbeit stark vereinfacht und beschleunigt. Kinematische und dynamische Robotermodelle müssen nun nicht mehr von Hand hergeleitet werden, sondern können automatisch und auf unterschiedlichen Wegen generiert werden. Die Ergebnisse dieser Forschungsarbeit wurden auf mehreren Internationalen Konferenzen und Workshops sowie auf der 36. Jahrestagung der Gesellschaft für Informatik präsentiert [183, 184, 185, 186, 187, 188]. Die eingereichten Beiträge wurden allesamt von Fachleuten auf diesem Gebiet positiv begutachtet.

1.3 Aufbau und Kapitelübersicht

Die Ausarbeitung ist entsprechend der Aufgabenstellung strukturiert und gliedert sich in insgesamt sechs Kapitel und einen Anhang. Im größten Teil der vorliegenden Arbeit werden serielle Kinematiken betrachtet. Der Schwerpunkt liegt hierbei auf der automatischen Erzeugung und Konfiguration der Steuerungssoftware und insbesondere auf der symbolischen Herleitung der Robotermodelle. Während im einleitenden ersten Kapitel das Promotionsthema und die verfolgten Ziele dargestellt werden, wird in den Kapiteln zwei und drei auf den Stand der Technik näher eingegangen. In den Kapiteln vier bis sechs wird die eigentliche Arbeit behandelt und es werden die eigenen Leistungen vorgestellt.

Das **zweite Kapitel** gibt eine ausführliche Übersicht über die wichtigsten kinematischen Aufbaumöglichkeiten von Industrierobotern. Es erfolgt eine Klassifizierung von Robotertypen nach der Anzahl, Art und Anordnung ihrer Haupt– und Nebenachsen sowie eventueller externer Achsen. Verschiedene Grundbauarten von Robotern, wie zum Beispiel kartesische, zylindrische und sphärische Roboter sowie horizontale und vertikale Knickarmkinematiken, werden genauer betrachtet und es werden ihre Merkmale und Einsatzgebiete vorgestellt. Diese Betrachtungen bilden die Grundlage für die Entwicklung eines allgemeinen Konfigurationswerkzeuges für die Bewegungssteuerung von Robotern.

Im **dritten Kapitel** folgt ein detaillierter Überblick über den gegenwärtigen Stand der Forschung und es werden offene Probleme aufgezeigt. Projekte, die verwandte Zielset-

zungen wie diese Arbeit verfolgen, werden genauer betrachtet und abgegrenzt. Manche dieser Projekte rechnen ausschließlich numerisch, während andere sowohl symbolisch als auch numerisch rechnen. Beispielsweise löst ROBOOP das kinematische Problem durch numerische Approximation und das dynamische Problem durch numerische Integration. Es werden die Vor– und Nachteile von numerischen und symbolischen Rechnen diskutiert. Insbesondere wird aufgezeigt, weshalb es günstiger ist, die Modelle für die Bewegungssteuerung analytisch und nicht näherungsweise zu bestimmen. Mehrkörpersimulationssysteme, Physik–Engines und Robotersimulationssysteme grenzen ebenfalls an diese Arbeit an. Mehrkörpersimulationssysteme dienen der Modellierung und der Aufstellung der Bewegungsgleichungen dynamischer Systeme. Physik–Engines werden zur Simulation dynamischer Vorgänge und zur Berechnung von Bewegungsabläufen in einer virtuellen Umgebung eingesetzt. Robotersimulationssysteme werden zur automatischen Generierung kollisionsfreier Bahnen und zur Bewegungssimulation von Kinematiken verwendet.

In **Kapitel vier** wird die Modellierung von kinematischen Ketten sowohl mit Denavit–Hartenberg Parametern als auch mit Twist–Koordinaten sowie die Bestimmung der direkten und inversen Kinematik beschrieben. Während die direkte Kinematik durch symbolische Matrizenmultiplikation vergleichsweise leicht ermittelt wird, müssen zur Ermittlung der inversen Kinematik nichtlineare Gleichungen in mehreren Unbekannten gelöst werden. Zuerst wird gezeigt, wie das inverse kinematische Problem für viele Robotertypen mit regelbasierten Methoden automatisch gelöst wird. Falls keine analytische Lösung mit den in einer Wissensbasis abgelegten mathematischen Lösungsregeln gefunden werden kann, so werden zusätzlich algebraische Eliminationsverfahren eingesetzt. Anhand einiger Beispielroboter wird gezeigt, wie sehr komplexe Gleichungssysteme durch die Bestimmung von Gröbnerbasen geschlossen gelöst werden können. Als Alternative zur Denavit–Hartenberg Konvention wird die direkte und inverse Kinematik auch auf Basis von Twist–Koordinaten bestimmt. Während die erste Methode eine weite Verbreitung in der Robotik gefunden hat, wird die zweite Methode noch selten verwendet. Das Kapitel endet mit der Bestimmung der Jacobi–Matrix für allgemeine Roboter nach diesen beiden Methoden.

In **Kapitel fünf** wird die automatische Bestimmung des dynamischen Robotermodells ausführlich beschrieben. Dynamik ist ebenso wie Kinematik ein Teilgebiet der technischen Mechanik. Der Unterschied ist, dass in der Dynamik im Gegensatz zur Kinematik bei der Betrachtung der Bewegungen von Körpern auch die verursachenden Kräfte und Momente berücksichtigt werden. Es wird gezeigt, wie das direkte und inverse dynamische Problem nach Lagrange und nach Newton–Euler gelöst wird. Beim ersten Verfahren werden Differentialgleichungen 2. Ordnung mittels Energiebetrachtungen und unter Verwendung der Jacobi–Matrix hergeleitet. Beim zweiten Verfahren wird der Impuls– und Drehimpulssatz auf die einzelnen Gelenkachsen angewendet, um die in den Schwerpunkten der Manipulatorglieder wirkenden Kräfte und Momente zu berechnen.

Kapitel sechs beschreibt in einem Referenzmodell, wie die Bewegungssteuerung aus den verschiedenen Einzelkomponenten zusammengesetzt wird. Dabei gelangen die in beiden vorherigen Kapiteln hergeleiteten Manipulatormodelle zur Anwendung und es werden experimentelle Implementierungsergebnisse präsentiert. Außerdem wird das Konfigurationswerkzeug beschrieben, mit dem der Benutzer die kinematische Struktur eines Roboters besonders einfach festlegen kann. Das Kapitel beinhaltet ebenfalls den im Rahmen dieser Arbeit entwickelten Interpreter für Roboterprogramme und die Trajektoriengenerierungskomponenten, um die verschiedenen Betriebsarten eines Roboters zu realisieren.

Die Ausarbeitung schließt mit einer Zusammenfassung der Ergebnisse sowie einer qualitativen und quantitativen Bewertung dieser Arbeit. Ein Ausblick auf mögliche Erweiterungen des Systems und auf zukünftige Forschungsaufgaben auf dem Gebiet der Robotersteuerungen wird gegeben. In **Anhang A** werden die wichtigsten mathematischen Sachverhalte, die für diese Arbeit relevant sind, aufgeführt. Es werden die verschiedenen Möglichkeiten, die Orientierung des Endeffektors zu beschreiben, erläutert sowie auf Lie Gruppen, Lie Algebren und die Formel von Rodrigues eingegangen. In **Anhang B** wird die Bewegungssteuerung eines vierachsigen Roboters beispielhaft konfiguriert. Das kinematische Modell und die Jacobi–Matrix werden rein analytisch mit Denavit–Hartenberg Parametern und Twist–Koordinaten hergeleitet. Die Bewegungsgleichungen des Roboters werden sowohl nach dem Lagrange als auch nach dem Newton–Euler Verfahren aufgestellt. In **Anhang C** wird gezeigt, wie die mechanische Struktur von Robotern in XML beschrieben wird. Dieses XML–Dokument enthält alle notwendigen Informationen um die Bewegungssteuerung automatisch zu konfigurieren. Es wird das XML Schema vorgestellt, das das Format dieser XML–Dokumente wiederum in XML festlegt.

Kapitel 2

Bauarten von Industrierobotern

Die Entwicklung eines Konfigurationswerkzeugs für die Bewegungssteuerung von Robotern ist auch mit der Fragestellung verbunden, welche Anordungen von Gelenkachsen industrietauglich und sinnvoll sind. Im ersten Schritt werden deswegen die zulässigen und nicht zulässigen mechanischen Konfigurationsmöglichkeiten und Randbedingungen bestimmt. Dabei liegt das Hauptaugenmerk auf ortsfest verankerten Industrierobotern.

Derzeit gibt es keine einheitliche Definition des Begriffs „Industrieroboter“. In der VDI–Richtlinie 2860 [173] sind folgende Kriterien festgelegt: *„Industrieroboter sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegung hinsichtlich Bewegungsfolgen und Wegen beziehungsweise Winkel frei programmierbar und gegebenenfalls sensorgeführt sind.“* Eine weitere Definition ist in der europäischen Norm EN 775 [39] zu finden: *„Ein Roboter ist ein automatisch gesteuertes, wiederprogrammierbares, vielfach einsetzbares Handhabungsgerät mit mehreren Freiheitsgraden, das entweder ortsfest oder beweglich in automatisierten Fertigungssystemen eingesetzt wird.“* Das Robotics Institute of America (RIA) verwendet wiederum eine etwas andere Begriffsdefinition.

Aufgrund unterschiedlicher Anforderungen und Aufgaben in der Fabrikautomation gibt es viele verschiedene Bauformen von Industrierobotern. Die einzelnen Robotertypen unterscheiden sich unter anderem hinsichtlich des Arbeitsraums, der Genauigkeit, der Tragfähigkeit, der Geschwindigkeiten und Beschleunigungen sowie der Zahl der Freiheitsgrade. Deswegen ist jeder Robotertyp nur für bestimmte Arbeiten geeignet. Ein SCARA kann zum Beispiel nicht sinnvoll für Lackier– oder Schweißaufgaben umprogrammmiert werden.

2.1 Klassische Kinematiken und Grundbauarten

Bei Roboterkinematiken unterscheidet man, je nachdem ob die Antriebe seriell oder parallel auf den Endeffektor einwirken, zwischen Seriell– und Parallelkinematiken. Eine Parallelkinematik besteht aus einer raumfesten und einer beweglichen Plattform, die miteinander über mehrere parallel angeordnete kinematische Ketten verbunden sind. Bei den meisten Industrierobotern handelt es sich um serielle Kinematiken, die aus einer einzelnen kinematischen Kette bestehen. Die Gelenkachsen einer seriellen Kinematik werden in Haupt– und Nebenachsen unterteilt. Die *Hauptachsen* (Grundachsen) dienen dazu, den Endeffektor zu positionieren. Typischerweise werden die Hauptachsen durch *Nebenachsen* (Handachsen) ergänzt, die überwiegend die Orientierung des Endeffektors bestimmen.

Gelenkachsen können translatorisch oder rotatorisch ausgelegt sein. *Translationsgelenke* (abgekürzt mit T) werden auch als Linear– oder Schubgelenke beziehungsweise prismati-

sche Gelenke bezeichnet und ermöglichen geradlinige Bewegungen. *Rotationsgelenke* (abgekürzt mit R) gestatten schnelle Dreh– und Schwenkbewegungen. Robotertypen werden häufig entsprechend den Typen ihrer Hauptachsen klassifiziert, zum Beispiel hat ein Gelenkarmroboter eine RRR–Kinematik, das heißt seine Hauptachsen sind Rotationsachsen. Hierbei handelt es sich um eine sehr grobe Klassifikation, da es für die Kombination von Translations– und Rotationsgelenken sowie deren Anordnung viele Möglichkeiten gibt. Entsprechend der Kombinationen der drei Hauptachsen eines Roboters unterscheidet man kartesische, zylindrische, sphärische sowie horizontale und vertikale Knickarmroboter.

2.1.1 Kartesischer Roboter / Portalroboter

Ein kartesischer Roboter hat drei rechtwinklig zueinander angeordnete Translationsgelenke. Deswegen wird diese Kinematik auch als TTT–Kinematik bezeichnet. Diese Achsanordnung ergibt eine sehr stabile und steife mechanische Konstruktion und der Arbeitsraum hat grundsätzlich eine quaderförmige Form. Die Orientierung des Endeffektors wird durch die Hauptachsen nicht verändert. Bei kartesischen Robotern kann deshalb die Koordinatentransformation sehr einfach bestimmt werden und es gibt keine Probleme mit Singularitäten. Andererseits ergibt sich bei dieser Bauart der Nachteil, dass die Bewegungsmöglichkeiten eingeschränkt sind und deswegen dieser Robotertyp nicht universell einsetzbar ist. Mit der Portalbauweise können sehr große Arbeitsräume realisiert und Objekte mit großem Gewicht transportiert werden. Die Wiederholgenauigkeit dieses Robotertyps liegt je nach Bauart und Traglast typischerweise im Bereich von $\pm 0,1$ mm bis $\pm 0,5$ mm. Der Verfahrweg in horizontaler Richtung kann 20 m und mehr betragen.

Kartesische Roboter sind zum Be– und Entladen von Werkzeugmaschinen sehr gut geeignet. Häufig werden Portalroboter zum Palettierern und Kommissionieren, zum Bohren, Fräsen und Schrauben, zum Transport von Gütern über lange Reichweiten sowie zum Plasma– und Wasserstrahlschneiden und für einfache Montageaufgaben eingesetzt.

2.1.2 SCARA–Roboter / horizontaler Knickarmroboter

Der SCARA beziehungsweise Schwenkarmroboter wurde bereits in den siebziger Jahren entwickelt, als Professor Makino in Japan feststellte, dass die meisten Füge– und Montageaufgaben durch Bewegungsabläufe senkrecht von oben nach unten durchgeführt werden können. Der Begriff SCARA steht für „Selective Compliance Assembly Robot Arm“ (engl.: Montageroboter mit gezielter Nachgiebigkeit). Ein Industrieroboter dieses Typs besteht aus zwei rotatorischen und einer translatorischen Hauptachse und hat demzufolge eine RRT, RTR oder TRR–Kinematik. Die drei Hauptachsen sind immer parallel angeordnet. Somit verfügt der Roboter über einen waagrechten Arm und eine vertikale Translationsachse zur Höhenverstellung des Endeffektors. Üblicherweise hat der SCARA auch ein viertes Gelenk, das als senkrechtes Rotationsgelenk ausgelegt ist. Die Grundform des Arbeitsraums ist annäherend zylinderförmig. Dieser Robotertyp zeichnet sich durch eine hohe Steifigkeit in der vertikalen Richtung und durch eine hohe Horizontalgeschwindigkeit und Beweglichkeit aus. Insbesondere Punkt–zu–Punkt Bewegungen werden sehr schnell ausgeführt. Der SCARA arbeitet mit einer hohen Wiederholgenauigkeit, die je nach Typ im Bereich von $\pm 0,005$ mm bis $\pm 0,05$ mm liegt. Dadurch ist der SCARA für sehr präzise Arbeiten mit hohen Genauigkeitsanforderungen geeignet. Die Größe des Arbeitsraums ist hingegen eher gering und die Orientierungsmöglichkeiten sind eingeschränkt. Die Traglasten sind auch eher gering und liegen typischerweise im Bereich von 1 kg bis zu 20 kg.

Der horizontale Knickarmroboter wird aufgrund seiner kleinen Standfläche in engen Arbeitsräumen an Produktionsmaschinen eingesetzt. Er ist insbesondere für senkrechte Montageaufgaben mit hohen Anforderungen an die Positionier– und Wiederholgenauigkeit geeignet. Dieser Robotertyp eignet sich außerdem zum Fügen, Löten, Kleben und Schrauben sowie für Handhabungs–, Palettier–, Bestückungs– und Kommissionieraufgaben.

2.1.3 Zylindrischer Roboter / Zylinderkoordinatenroboter

Bei zylindrischen Robotern bestehen die Hauptachsen aus zwei translatorischen und einer rotatorischen Gelenkachse. Die drei Hauptachsen sind so angeordnet, dass durch sie ein zylindrisches Koordinatensystem aufgespannt wird. Dieser Robotertyp zeichnet sich durch eine steife und robuste Bauweise bei großer Reichweite und großem Platzbedarf aus. Er kann sehr schnelle kreisförmige Bewegungen in der Horizontalen durchführen. Das Umgreifen von Hindernissen ist nicht möglich. Der Arbeitsraum ist verhältnismäßig groß und hat die Form eines Hohlzylinders beziehungsweise des Segments eines Hohlzylinders.

Der zylindrische Robotertyp kann beispielsweise zum Kommissionieren, zum Palettieren, zur Materialhandhabung und zur Maschinenbeschickung verwendet werden.

2.1.4 Sphärischer Roboter / Kugelkoordinatenroboter

Die Hauptachsen eines sphärischen Roboters bestehen aus zwei rotatorischen und einer translatorischen Gelenkachse. Sie sind so angeordnet, dass durch sie ein sphärisches beziehungsweise polares Koordinatensystem aufgespannt wird. Der große Arbeitsraum hat annähernd die Form einer Hohlkugel. Bereits 1969 entwickelte V. Scheinman im Stanford Artificial Intelligence Lab einen sphärischen Roboter, den berühmten *Stanford–Manipulator*. Dieser Manipulator hat eine RRT–Kinematik und besitzt insgesamt fünf rotatorische und ein translatorisches Gelenk. Der Stanford–Arm ist einer der ältesten Roboter. Er gilt als Vorbild für viele spätere Robotertypen hauptsächlich in der Automobilindustrie.

Der sphärische Robotertyp kann für Handhabungsaufgaben an Druckgußmaschinen, zum Lackieren von Bauteilen und zum Punktschweißen an Karosserien verwendet werden.

2.1.5 Gelenkarmroboter / vertikaler Knickarmroboter

Zylindrische und sphärische Roboter werden heutzutage nur noch selten eingesetzt. Die meisten seriellen Kinematiken in der Fabrikautomation sind heutzutage Gelenkarmroboter, da sie von allen Roboterbauarten am universellsten einsetzbar sind. Gelenkarmroboter werden daher auch als Universalroboter bezeichnet. Sie haben grundsätzlich eine RRR–Kinematik und der Arbeitsraum ist hohlkugelförmig. Die Rotationsgelenke der Hauptachsen eines Gelenkarmroboters knicken in der vertikalen Ebene. Die Wiederholgenauigkeit liegt je nach Bauart im Bereich von $\pm 0,05$ mm bis $\pm 0,2$ mm. Dieser Robotertyp zeichnet sich durch einen großen Arbeitsraum bei geringem Platzbedarf, durch hohe Beweglichkeit, schnelle Rotationsbewegungen und geringes Störvolumen aus. Der Gelenkarmroboter kann auch für Arbeiten an schwer zugänglichen Stellen eingesetzt werden. Er kann Hindernisse umgehen, in Hohlkörper greifen und bei manchen Bauarten ist auch Überkopfarbeit möglich. Die Traglast von Gelenkarmrobotern reicht heutzutage von $2,5$ bis zu 1000 kg.

Für den Gelenkarmroboter gibt es wegen seiner Universalität zahlreiche Einsatzmöglichkeiten. Er kann für Bahn– und Punktschweißprozesse sowie für Montage–, Palettier– und

Handhabungsaufgaben genutzt werden. Darüber hinaus kann er als Schneidroboter zum Fräsen, Sägen und Wasserstrahlschneiden, als Beschichtungsroboter zum Kleben und Lackieren, oder als Bearbeitungsroboter zum Schleifen und Entgraten eingesetzt werden.

2.1.6 Hexapod / Stewart–Gough–Plattform

Beim Hexapod–Roboter (griech.: Sechsfüßler) handelt es sich um eine geschlossene kinematische Kette, bei der sechs Translationsachsen eine gemeinsame Plattform bewegen. Das Konzept des Hexapods wurde 1965 erstmals von D. Stewart [163] vorgestellt. Die erste Maschine zum Testen von Fahrzeugreifen auf Basis des Hexapod–Prinzips wurde von V. Gough [64] entworfen. Deswegen wird der Hexapod oftmals auch als Stewart–Plattform beziehungsweise als Stewart–Gough–Plattform bezeichnet. Der Hexapod zeichnet sich durch seine extreme Steifigkeit und außerordentlich hohe Wiederholgenauigkeit aus. Außerdem sind große Geschwindigkeiten und Beschleunigungen erreichbar. Beim Hexapod ist die Vorwärtstransformation mehrdeutig und aufwendig zu bestimmen, während die Rückwärtstransformation eindeutig und sehr viel einfacher zu ermitteln ist.

In der Fabrikautomation wird der Hexapod zum Schweißen, zur Laserbearbeitung, zur Kleinteilmontage sowie für Schleif–, Fräs–, Bohr– und Drehaufgaben eingesetzt. Weitere Einsatzgebiete sind Präzisionsanwendungen in der Medizin– und der Feinmeßtechnik. Der Hexapod wird oft auch als Bewegungssystem für Flug– und Fahrsimulatoren verwendet.

2.1.7 Gegenüberstellung der verschiedenen Kinematiken

Der Gelenkarmroboter bietet von allen hier vorgestellten Robotertypen die höchste Flexibilität und Beweglichkeit. Der Portalroboter kann im Gegensatz zu anderen kinematischen Ketten auch extrem hohe Traglasten von mehreren Tonnen bewegen und Objekte über sehr große Distanzen hinweg transportieren. Der SCARA erreicht von allen seriellen Kinematiken die höchste Genauigkeit. Die Genauigkeit und Steifigkeit von Parallelkinematiken ist höher als die von seriellen Kinematiken, da sich Positionierfehler und Elastizitäten der einzelnen Gelenkachsen nicht aufsummieren. Das Last–Masse–Verhältnis und damit die Dynamik bei Parallelkinematiken ist ebenfalls besser als bei seriellen Kinematiken, da die Antriebe nicht die Last der nachfolgenden Gelenkachsen tragen müssen, sondern direkt auf den Endeffektor wirken. Als Nachteil von Parallelkinematiken ergeben sich der eingeschränkte Arbeitsraum, insbesondere hinsichtlich der Orientierung, und der höhere Berechnungsaufwand für das kinematische und dynamische Robotermodell. In Tabelle 2.1 wird die Dualität von seriellen Kinematiken und von Parallelkinematiken erkennbar.

Tabelle 2.1 Vergleich der Eigenschaften von Seriell– und Parallelkinematiken

Merkmal	seriell	parallel
Genauigkeit	+−	++
Steuerungsaufwand	+−	−
Last/Masse–Verhältnis	−	+
Größe des Arbeitsraums	++	−
Beschleunigungen am Endeffektor	+−	++
Anpassbarkeit an Aufgabenstellung	++	+−

2.2 Handkinematiken und Sonderbauformen

Industrieroboter können unterschiedlich viele Gelenkachsen haben. Roboter mit vier oder fünf Gelenkachsen verfügen über drei Hauptachsen und einer beziehungsweise zwei Nebenachsen. Die Orientierbarkeit des Endeffektors ist hierdurch eingeschränkt. Allerdings ist diese Achsanzahl akzeptabel bei Arbeiten mit rotationssymmetrischen Werkzeugen oder bei Arbeiten bei denen nur eine eingeschränkte Orientierung erforderlich ist. Einfache Montageaufgaben können mit vierachsigen und einfache Handhabungsaufgaben mit fünfachsigen Robotern bewerkstelligt werden. Portalroboter haben meistens eine bis drei rotatorische Nebenachsen zur Drehung des Endeffektors. Die meisten Gelenkarmroboter haben sechs Achsen und verfügen damit über sechs Freiheitsgrade im kartesischen Raum, um anspruchsvolle Bewegungsabläufe wie zum Beispiel beim Schweißen auszuführen.

Für besondere Aufgabenstellungen gibt es Sonderbauformen von Robotern [181, 189]. Diese Sonderbauformen bieten für spezielle Anwendungsbereiche eine bessere Zugänglichkeit, Bewegungsfreiheit oder Steifigkeit. So können Roboter mit mehr als sechs Achsen in einem Raum mit Hindernissen von Vorteil sein beziehungsweise den Arbeitsraum erweitern. Außerdem können singuläre Konfigurationen leichter vermieden werden. Diese zusätzlichen Achsen werden auch als *redundante Gelenkachsen* bezeichnet. Kinematisch redundante Manipulatoren sind zudem robuster gegenüber dem Ausfall einzelner Gelenke.

2.2.1 Anordnungen der Nebenachsen

Die Nebenachsen bei Robotern dienen hauptsächlich dazu, den Effektor zu orientieren. Es handelt sich bei ihnen deswegen immer um Rotationsachsen. Gewöhnlicherweise besitzt ein Roboter eine bis drei Nebenachsen, die unterschiedlich angeordnet werden können.

Die *Zentralhand* ist die am weitesten verbreitete Konstruktionsform der Nebenachsen eines Roboters. Bei dieser Anordnung schneiden sich die drei Rotationsachsen in einem Punkt, dem sogenannten *Handwurzelpunkt* beziehungsweise *Handgelenkspunkt*. Dadurch hat die Zentralhand den Vorteil, dass die Handachsen nur die Orientierung, nicht aber die Position des Effektors verändern. Durch diese Entkopplung der Achsen ergeben sich Erleichterungen bei der Berechnung der inversen Kinematik [135]. Außerdem besitzt die Zentralhand eine lediglich kleine Störkontur. Der Nachteil der Zentralhand ist, dass eine Singularität entsteht, wenn die erste und dritte Rotationsachse parallel zueinander liegen.

Lackierroboter werden in der Regel mit speziell für Spritzapplikation entwickelten Handgelenken ausgestattet, die präzise Bewegungen und das Auslackieren von komplizierten Geometrien ermöglichen. Die *Hohlkegelhand* (engl.: hollow wrist) verfügt über einen sehr großen Arbeitsbereich. Sie ist so konstruiert, dass die Lackier– und Luftzufuhrschläuche, die in der Hand integriert sind, nicht beschädigt werden oder durch Abnutzung brechen.

2.2.2 Roboter mit externen Gelenkachsen

Zur Vergrößerung des Arbeitsraums sowie zur Fortbewegung und Vorpositionierung des Roboters werden zusätzliche Achsen eingesetzt. Roboter erhalten durch Schienen am Boden, an den Wänden oder durch Laufkräne unter der Decke ein bis zwei weitere Freiheitsgrade. Dadurch ist zum Beispiel die gleichmäßige Lackierung einer ganzen Fahrzeugseite, das Schweißen von mehreren Meter langen Nähten, der Transport von Werkstücken sowie die Verkettung von Werkzeugmaschinen möglich. Bei der Portalbauweise ist der Boden-

bereich frei und kann so für weitere Fertigungseinrichtungen genutzt werden. Die Verfahrschienen, auf denen Roboter befestigt sind, werden als *externe Gelenkachsen* bezeichnet.

Außer dem Roboter kann die Robotersteuerung auch die Peripherie und externe Positioniersysteme steuern. Bei einem sechsachsigen Roboter mit einem Drehkipptisch werden insgesamt acht Achsen synchron gesteuert. Das Werkzeug am Roboterflansch wird gleichzeitig mit dem Werkstück auf dem Drehkipptisch bewegt. Dadurch kann das Werkstück von mehreren Seiten bearbeitet und schwer zugängliche Stellen besser erreicht werden. Drehkipptische werden heutzutage beim Schweißen, Entgraten und Montieren eingesetzt.

2.2.3 Parallelogramm–Roboter

Der Parallelogramm–Roboter ist eine Sonderbauform, bei der Rotationsachsen über ein Parallelogramm fest miteinander verbunden sind. Der Ursprung dieser Konstruktionsform ist das *Wattsche Parallelogramm*, das erstmals in der von *James Watt* (1736–1819) entwickelten Dampfmaschine eingesetzt wurde, um eine rotatorische in eine geradlinige Bewegung umzuwandeln. Parallelogramm–Roboter sind in der Regel mit vier Achsen ausgestattet und der Endeffektor kann automatisch parallel zum Boden geführt werden. Der Arbeitsraum hat die Form einer Halbkugel. Zu den Einsatzgebieten dieses Robotertyps gehören die Handhabung schwerer Werkstücke, das Palettieren und Punktschweißen.

2.2.4 Modulare rekonfigurierbare Roboterstrukturen

Klassische Industrieroboter bestehen aus einer zusammenhängenden mechanischen Struktur. Modulare Roboter werden hingegen aus vorgefertigten und standardisierten Antriebsmodulen für die jeweilige Applikation zusammengesetzt. Diese Standardmodule selbst sind relativ einfach aufgebaut und besitzen nur wenige Freiheitsgrade. Es gibt Module mit unterschiedlicher Größe und Geometrie, beispielsweise können sie würfel– oder zylinderförmig sein. Mit diesen Modulen als Gelenkachsen und verschiedenen mechanischen Verbindungselementen können sehr flexibel individuelle Leichtbauroboter baukastenartig zusammengesetzt werden. Bei einer Umstellung der Produktion können diese Kinematiken sehr leicht adaptiert und anders aufgebaut werden. Für den ortsunabhängigen Einsatz werden modulare Roboter auf mobilen Plattformen fest montiert. Außer in der Fabrikautomation werden modulare Roboter heutzutage auch in der Servicerobotik eingesetzt.

2.2.5 Kooperierende Mehrrobotersysteme

Zur Verringerung der Taktzeit beziehungsweise Erhöhung der Fertigungsgeschwindigkeit sowie zur Verminderung teurer Spann– und Fixiereinrichtungen kooperieren mehrere Roboter miteinander. Die Steuerung erfolgt über das *Master–Slave–Verfahren*, das heißt es wird ein Hauptroboter bestimmt und die anderen beteiligten Roboter passen ihre Bewegungen an die des Hauptroboters an. Für die Kooperation von Robotern gibt es verschiedene Anwendungsszenarien. Beispielsweise hält und dreht ein Positionierroboter das Bauteil stets so, dass andere Bearbeitungsroboter in optimaler Position am Bauteil arbeiten können (Entgraten, Laserbeschnitte, Schutzgasschweißen). Durch die Bearbeitung eines Bauteils während dessen Transports können unproduktive Transportzeiten gesenkt werden. Neben der koordinierten Handhabung von Bauteilen, erhöht das Aufteilen der Traglast auf mehrere Roboter außerdem die maximale Traglast. Ein Beispiel sind zwei Industrieroboter, die gemeinsam eine Windschutzscheibe in eine Autokarosserie einsetzen.

Kapitel 3

Aktueller Stand der Forschung

Die Automatisierungstechnik zeichnet sich bislang vor allem durch proprietäre Hard– und Softwaresysteme aus. Jeder Roboterhersteller verwendet seine eigene Steuerung und seine eigene Roboterprogrammiersprache. Nur im Bereich der Feldbusse haben sich diverse Standards etabliert, zum Beispiel PROFIBUS. Darüber hinaus gibt es den Trend für Steuerungen zunehmend PC–Technik einzusetzen. Aufgrund der wachsenden Leistung von PC–Hardware können in Steuerungen immer mehr Funktionalitäten integriert werden.

Grundlegende Anforderungen an Steuerungssoftware beinhalten neben Echtzeitfähigkeit auch Anpassbarkeit, Zuverlässigkeit und Wartbarkeit. Bezüglich Softwarequalität werden sehr hohe Anforderungen gestellt, da Fehler enorme Schäden anrichten können. Heutzutage erfordern Steuerungen sehr hohe Entwicklungskosten und lange Entwicklungszeiten. Steuerungssoftware wird überwiegend von Hand entwickelt und implementiert. Es fehlen vor allem robotikspezifische Komponentenbibliotheken sowie Entwicklungsumgebungen und Konfigurationswerkzeuge. Steuerungen können deswegen immer schwerer weiterentwickelt und nur schlecht an neue Anforderungen angepasst werden. Da die einzelnen Funktionen sehr eng miteinander verzahnt sind und es sich überwiegend um proprietäre Software handelt, ist die Integration neuer Komponenten auf Basis von *„Plug and Work“* nicht möglich. Die Verwendung einer proprietären Steuerung führt zu Abhängigkeiten gegenüber einem Hersteller. Bei Open–Source–Software ist es im Gegensatz zu proprietärer Software möglich und lizenzrechtlich zulässig, die Software zu erweitern und zu ändern.

Da Steuerungssoftware für Roboter stark an die mechanische Struktur eines Roboters gebunden ist, ist es erforderlich, diese Software für neue Robotertypen anzupassen oder ganz neu zu entwickeln. Oftmals wird Steuerungssoftware speziell auf einen einzelnen Roboter zugeschnitten und kann nicht wiederverwendet werden. In der Fabrikautomation gibt es eine hohe Anzahl an unterschiedlichen Robotertypen. Aufgrund der zunehmenden Variantenvielfalt erfordert die Beherrschung der damit verbundenenen Softwarevarianten einen hohen Aufwand. Die Entwicklung einer allgemeinen und modular erweiterbaren Softwarearchitektur für Steuerungen, die weitestgehend auf alle Klassen von Robotern angewandt werden kann, ist daher von großem Nutzen. Bislang gibt es nur wenige Initiativen, die das Ziel der Vereinheitlichung sowie der flexiblen und schnellen Erstellung von Steuerungssoftware für Roboter verfolgen. Die folgenden Beschreibungen geben einen Überblick über diese Projekte. Es werden auf Projekte im Bereich des numerischen Rechnens (Abschnitt 3.1), im Bereich des symbolischen Rechnens (Abschnitt 3.2), auf Mehrkörpersimulationssysteme (Abschnitt 3.3) und auf Physik–Engines (Abschnitt 3.4) sowie auf 3D–Robotersimulationssysteme (Abschnitt 3.5) eingegangen. Danach werden die Projekte bezüglich ihrer Funktionalitäten miteinander verglichen (Abschnitt 3.6).

3.1 Projekte im Bereich des numerischen Rechnens

Eine Reihe von Projekten auf dem Gebiet universeller Steuerungssoftware für Roboter rechnen rein numerisch und nicht symbolisch. Die Parameter eines Roboters müssen vom Benutzer mit konkreten Werten belegt werden. Komplexe Problemstellungen wie beispielsweise das kinematische Robotermodell, das kartesische Koordinaten in Gelenkkoordinaten umrechnet, werden durch näherungsweise Iterationsverfahren mit hohem Rechenbedarf gelöst [192]. Numerische Lösungen haben oft einen sehr viel geringeren Entwicklungsaufwand als symbolische Lösungen. Manche Probleme können auch nur numerisch und nicht symbolisch gelöst werden, zum Beispiel die Bestimmung der Nullstellen von Polynomfunktionen fünften Grades. Nur in manchen Fällen ist hier auch eine symbolische Lösung möglich. Allerdings haben numerische Verfahren auch Nachteile gegenüber symbolischen Verfahren. Es besteht die Gefahr, dass Startwerte ungünstig gewählt werden und Iterationsverfahren dann zu langsam oder überhaupt nicht konvergieren. Numerische Verfahren liefern in der Regel keine exakten, sondern nur näherungsweise Lösungen und ungenaue Zwischenergebnisse können zu Folgefehlern führen. Bereits die Gleitkommadarstellung von reellen Zahlen in Rechnern ist nur eine Näherung. Die endliche Zahlendarstellung führt dazu, dass Rundungsfehler auftreten können. Numerische Verfahren können vor allem bei rekursiver Ausführung nur schwierig parallelisiert und auf mehrere Prozessorknoten verteilt werden. Bei sehr großer Rekursionstiefe können rekursive Programme einen Speicherüberlauf verursachen. Die relevanten Projekte, die ausschließlich numerisch rechnen und allgemeine Steuerungssoftware entwickeln, die nicht nur hinsichtlich einer bestimmten Manipulatorgeometrie verwendet werden kann, werden nachfolgend beschrieben. Zu diesen Projekten gehören Orocos, Oscar, Roboop, Matlab Robotics Toolbox, RTSS, PMT, ManDy, Microb, Synthetica, Robotect und RobotDraw.

3.1.1 Orocos: Open Robot Control Software

Auf europäischer Ebene ist das Projekt Orocos [13, 14, 15] zu nennen, welches von der Universität Leuven in Belgien koordiniert wird. Zu den weiteren Projektpartner gehören das französische Laboratorium für Analyse und Systemarchitektur (LAAS) aus Toulouse und die schwedische Hochschule KTH aus Stockholm. Orocos besteht aus vier eng gekoppelten Teilsystemen, um alle Aspekte einer Robotersteuerung abzudecken. Das erste Teilsystem namens *Real–Time Toolkit (RTT)* ist eine echtzeitfähige Middleware, deren Kommunikationssystem auf einer Corba Implementierung basiert. Komponenten können auch in heterogenen und verteilten Umgebungen ablaufen und sind zwischen verschiedenen Plattformen austauschbar. Das zweite Teilsystem, das sich in der Entwurfsphase befindet, ist die *Kinematics and Dynamics Library (KDL)*. Mit dieser Bibliothek soll zukünftig die Kinematik und Dynamik von Mehrkörpersystemen berechnet werden können. Das dritte Teilsystem ist eine Softwarebibliothek zum Probabilistischen Schließen in Bayes Netzwerken, die sogenannte *Bayesian Filtering Library (BFL)*. Das vierte Teilsystem ist die *Orocos Component Library (OCL)*, eine roboterspezifische Klassenbibliothek, die unter anderem Algorithmen für Steuer– und Regelungsaufgaben enthält. Der C/C++ Code läuft unter Linux ohne beziehungsweise unter RTAI (Real–Time Application Interface) mit harten Echtzeiteigenschaften ab. Der Quellcode ist frei erhältlich und wurde unter die GNU LGPL (Lesser General Public License) gestellt. Dadurch kann die Software auch in kommerziellen Projekten eingesetzt werden, ohne den Quellcode offenlegen zu müssen.

3.1.2 Oscar: Operational Software Components for Advanced Robotics

OSCAR [82, 85, 103, 145, 165] ist ein Projekt der Universität von Texas in Austin, das auch von der amerikanischen Raumfahrtbehörde NASA und dem Energieministerium der Vereinigten Staaten (Department of Energy) verwendet wird. Das Projekt ist primär auf die Steuerung von kinematisch redundanten Robotern mit vielen Gelenkachsen und vielen Freiheitsgraden ausgerichtet. Zur Beherrschung der Variantenvielfalt bezüglich Roboter unterstützt OSCAR den Anwender bei der Entwicklung einer Bewegungssteuerung durch Produktlinienmethoden und durch eine allgemeine Klassenbibliothek, die in der Programmiersprache C++ implementiert ist. Die Bibliothek enthält beispielsweise Klassen zur Vektor– und Matrizenrechnung. Zur kinematischen Beschreibung eines Roboters werden Denavit–Hartenberg Transformationsmatrizen verwendet. Gelenke können nur aktiv, das heißt rotatorisch oder translatorisch, aber nicht passiv sein. Die direkte und inverse Kinematik wird für allgemeine Robotertypen numerisch gelöst. In der Bibliothek ist auch eine analytische Lösung des inversen kinematischen Problems von einem Unimate Puma 760 (Programmable Universal Manipulator for Assembly) und von einem Mitsubishi PA 10 (Portable Arm) enthalten. Der Benutzer kann für diese beiden Roboter durch entsprechendes Setzen von booleschen Variablen eine bestimmte Gelenkkonfiguration bei gegebener Endeffektorlage auswählen. Das dynamische Modell wird sowohl mit dem *rekursiven Newton–Euler Verfahren* [92] als auch mit dem *Lagrange Verfahren* [72] numerisch bestimmt. Die Bibliothek enthält auch diverse Algorithmen zur Trajektoriengenerierung. Beispielsweise erfolgt die Berechnung einer Trajektorie im Gelenkwinkelraum mit einer Polynomfunktion 5. Grades, bei der allerdings der Benutzer die Bewegungsdauer angegeben muß. OSCAR ist keine Open–Source–Software und es erfolgt keine Weitergabe an Dritte. Nur die Vorgänger–Software RRG KINEMATIX 4.0, die allerdings keine Dynamikalgorithmen enthält, ist als DLL–Datei (Dynamic Link Library) für WINDOWS verfügbar.

3.1.3 Roboop: A Robotics Object Oriented Package in C++

ROBOOP [65] wurde von R. Gourdeau an der École Polytechnique Montréal in Kanada entwickelt. ROBOOP stellt verschiedene Klassen zur Verfügung, um die direkte und inverse Kinematik beziehungsweise Dynamik zu berechnen. Außerdem gibt es Klassen, um die Jacobi–Matrix und die Ableitung der Jacobi–Matrix nach der Zeit zu berechnen. Hierfür werden die Koordinatensysteme nach der Denavit–Hartenberg Konvention festgelegt. Die inverse Kinematik wird durch Reihenentwicklung und Anwendung der Taylorschen Formel approximiert [171]. Die Roboterdynamik wird mit dem rekursiven Newton–Euler Verfahren bestimmt. Für die Matrizen– und Vektorrechnungen verwendet ROBOOP die frei erhältliche Mathematikbibliothek NEWMAT von R. Davies [32]. Zur Simulation des Bewegungsverhaltens von Robotern werden nichtlineare Differentialgleichungen integriert. Die Integration der Differentialgleichungen erfolgt numerisch nach dem nach zwei deutschen Mathematikern benannten *Runge–Kutta–Verfahren* (4. Ordnung) [143]. Die Ordnung dieses Verfahrens gibt die Anzahl der Funktionsauswertungen pro Integrationsschritt an. Die Software von ROBOOP ist frei erhältlich und steht unter GNU LGPL. Sie läuft unter WINDOWS und LINUX Betriebssystemen und es wird keine kommerzielle Zusatzsoftware benötigt. Darüber hinaus wurde ROBOOP von C. Lia an der Universität von Neapel nach JAVA portiert. Diese ebenfalls frei erhältliche Bibliothek (GNU GPL), die für die Matrizenrechnungen das Paket JAMA (JAVA MATRIX PACKAGE) verwendet, heißt JROBOOP.

3.1.4 Matlab Robotics Toolbox

Die MATLAB ROBOTICS TOOLBOX [26, 27] wurde am CSIRO ICT Centre in Australien von P. Corke entwickelt und ist eine Sammlung von elementaren Funktionen in der Robotik. Sie beinhaltet Funktionen für geometrische Transformationen, Berechnungen der direkten und inversen Kinematik und Dynamik sowie der Generierung von Bewegungsbahnen. Manipulatoren werden mit Hilfe der Denavit–Hartenberg Konvention modelliert. Hinsichtlich der Gelenkarten werden Rotations– und Translationsgelenke unterschieden. Die Lösung der Rückwärtskinematik wird durch ein iteratives, numerisches Verfahren bestimmt. Zur Bestimmung des dynamischen Robotermodells implementiert die Toolbox den rekursiven Newton–Euler Algorithmus. Die Toolbox enthält außerdem geschlossene Lösungen für die kinematischen und dynamischen Modelle von ausgewählten Robotertypen, wie zum Beispiel für den Unimate Puma 560 und den Stanford Manipulator. Außerdem enthält die Toolbox Funktionen zur rudimentären Visualisierung von Roboterpositionen. Zur Trajektoriengenerierung im Gelenkwinkelraum wird eine Polynomfunktion 7. Grades verwendet. Hierbei ist vom Benutzer das zeitliche Intervall vom Beginn bis zum Ende der Bewegung zu spezifizieren. Die Toolbox selbst ist frei erhältlich. Es wird allerdings das kommerzielle Programmpaket MATLAB (MATRIX LABORATORY) benötigt. Die Toolbox von P. Corke kann als etwas veraltet betrachtet werden, da sie MATLAB Release 7 verwendet, während Release 19 (R2008a) die aktuelle MATLAB Version ist. Heutzutage wird sie teilweise für Lehr– und Ausbildungszwecke eingesetzt. Später wurde die MATLAB ROBOTICS TOOLBOX am Centro de Investigación en Matemáticas (CIMAT) in Mexiko von D. N. Vila–Rosado und J. A. Dominguez–Lopez [177, 178] etwas verbessert. Der Benutzer kann nun zwischen mehr Optionen bei der Trajektoriengenerierung auswählen. Diese Version wurde allerdings bislang nicht verbreitet oder Dritten zugänglich gemacht.

3.1.5 RTSS: Robotics Toolbox for Scilab/Scicos

Inspiriert von der MATLAB ROBOTICS TOOLBOX (Abschnitt 3.1.4) hat M. Morelli an der Universität von Pisa in Italien die ROBOTICS TOOLBOX FOR SCILAB/SCICOS entwickelt. Hierbei handelt es sich um eine Erweiterung von SCILAB/SCICOS zur Modellierung und Simulation von Robotern. SCILAB [22] ist ein ähnliches Software–Paket wie MATLAB, das seit 1990 am Institut National de Recherche en Informatique et en Automatique (INRIA) in Frankreich entwickelt wird. In RTSS werden Manipulatoren mit der Denavit–Hartenberg Konvention beschrieben. Die Funktionen zur Berechung des kinematischen und dynamischen Robotermodells sowie zur Trajektoriengenerierung wurden im Wesentlichen von der MATLAB ROBOTICS TOOLBOX übernommen und nach SCILAB portiert. Die Software ist für LINUX und WINDOWS Betriebssysteme bei SOURCEFORGE erhältlich. Sie ist quelloffen und lizenzkostenfrei entsprechend GNU GPL.

3.1.6 PMT: Planar Manipulators Toolbox

PLANAR MANIPULATORS TOOLBOX [194] ist eine Umgebung zur Simulation von planaren n–Gelenk–Manipulatoren in dem Numeriksystem MATLAB und wurde von L. Zlajpah am slowenischen Jozef Stefan Institut entwickelt. Als Gelenkarten werden ausschließlich Rotationsgelenke unterstützt. Die direkte Kinematik wird durch eine rekursive Funktion, die jedoch nur bei Gelenkanordnungen in einer Ebene anwendbar ist, berechnet. Deswegen werden keine Denavit–Hartenberg Parameter verwendet. Die Jacobi–Matrix und die zeitliche Ableitung der Jacobi–Matrix werden anhand der direkten Kinematik berechnet. Diese

beiden Matrizen haben bei planaren Manipulatoren die Dimension $2 \times n$. Das dynamische Modell wird nach dem Lagrange Verfahren bestimmt. Der Quellcode ist frei verfügbar und benötigt MATLAB Release 6.5. Das System wurde für Lehrzwecke entwickelt.

3.1.7 ManDy: Manipulator Dynamics

MANDY [182] ist ein in dem Computermathematiksystem MATLAB Release 13 und der Programmiersprache C geschriebenes Softwarepaket zur Programmierung, Simulation und Visualisierung von Roboterarmen und anderen offenen kinematischen Ketten. Der Benutzer kann serielle Roboter modellieren, Erreichbarkeitsstudien durchführen, programmierte Roboterbewegungen visualisieren sowie die Regelungsgüte testen. MANDY wurde von W. Weber im Zentrum für Robotik an der Fachhochschule Darmstadt entwickelt. Der Benutzer kann die Denavit–Hartenberg Parameter, die eine kinematische Kette mit bis zu zehn Rotations– oder Translationsgelenken beschreiben, in eine graphische Benutzeroberfläche eingeben. Die Rückwärtstransformation wird näherungsweise mit der inversen Jacobi–Matrix bestimmt. Das dynamische Modell wird numerisch mit dem rekursiven Newton–Euler Algorithmus berechnet. Differentialgleichungen können mit dem Runge–Kutta Verfahren der Ordnung zwei oder vier integriert werden. Es gibt auch eine Offline–Programmierumgebung in der Programme in einer selbstdefinierten Roboterprogrammiersprache erstellt und von einem Interpreter ausgeführt werden können. Da die Animationen von Robotern in der 3D–Beschreibungssprache VRML (Virtual Reality Modeling Language) geschrieben sind, benötigt MANDY einen gewöhnlichen Browser mit VRML Plug–In für die Visualisierung. Diese Software ist frei erhältlich und als Quellcode verfügbar. Bislang wurde sie als Ergänzung in Lehrveranstaltungen eingesetzt.

3.1.8 Microb: Modules Intégrés pour le Contrôle de Robots

MICROB [76, 142] ist eine Sammlung von C++ Klassen zur vereinfachten Entwicklung von Steuerungen für elektromechanische Systeme. Es wurde am Forschungsinstitut Hydro–Québec (IREQ) in Kanada entwickelt und ist unter die GNU LGPL gestellt. Außer für serielle Kinematiken ist es auch für mobile Roboter geeignet. MICROB ist in mehrere Module gegliedert, die unabhängig voneinander verwendet werden können. Eine der größten Vorteile von MICROB ist, dass extrem viele verschiedene Betriebssysteme unterstützt werden, beispielsweise QNX, VXWORKS, WINDOWS NT/2000, SOLARIS, LINUX, IRIX und SUNOS. In dem Modul *Operating System Dependant Library (OSDL)* wird der plattformspezifische Code gekapselt auf den über eine einheitliche Schnittstelle zugegriffen wird. Bei der Portierung von MICROB auf andere Plattformen braucht nur dieses Modul angepasst werden. Alle anderen Module sind plattformunabhängig. Eine Anwendung kann auf einem Arbeitsplatzrechner entwickelt und dann auf einem Echtzeitbetriebssystem rekompiliert und ausgeführt werden. Das Modul *Robot* enthält Klassen, um kinematische Ketten durch Denavit–Hartenberg Parameter zu beschreiben und das direkte kinematische Problem numerisch zu lösen. Außerdem enthält das Modul die Lösung der inversen Kinematik für bestimmte sechsachsige Robotertypen. Für Robotertypen, deren Lösung nicht bekannt ist, muß der Benutzer eine neue Roboterklasse von der Klasse *Serial_robot*, *Mobile_robot* beziehungsweise *Wheeled_robot* ableiten und die entsprechenden Methoden überladen. Die gegenwärtige Version von MICROB enthält keine Algorithmen, um das dynamische Robotermodell zu berechnen. Um die Dynamik zu berücksichtigen, muss der Benutzer die entsprechenden Methoden, wie zum Beispiel *compute_static_forces()*

und *compute_inertia()*, selbst implementieren. Das Modul *Vectmath* enthält Klassen für Vektoren– und Matrizenrechnung, beispielsweise Euler–Winkel, Quaternionen und Transformationsmatrizen. Das Modul *Signal_processing* stellt verschiedene Algorithmen zur Sensordatenverarbeitung, wie Mittelwert–, Median–, Butterworth–, Ableitungs– und Integrationsfilter, zur Verfügung. Das Modul *Control* enthält Klassen zur Trajektoriengenerierung, wobei allerdings ein trapezförmiges Geschwindigkeitsprofil verwendet wird.

3.1.9 Synthetica: Kinematic Synthesis of Robots

SYNTHETICA [25, 137, 160] ist ein System zur Darstellung, Analyse und Simulation von offenen und geschlossenen Kinematiken und wurde am Department of Mechanical and Aerospace Engineering an der Universität von Kalifornien entwickelt. Es ist in insgesamt vier Pakete gegliedert. Das Paket *mechanism* beinhaltet Klassen und spezifiziert Schnittstellen, um mechanische Strukturen zu definieren. Es werden Kugel–, Schrauben–, Scharnier–, Translations– und Rotationsgelenke unterstützt. Die Schnittstelle *ForwardKinematics* ist für allgemeine serielle Kinematiken bereits implementiert, während die Schnittstelle *InverseKinematics* für selbstdefinierte Kinematiken vom Benutzer implementiert werden kann beziehungsweise für bestimmte Kinematiken bereits implementiert ist. Im allgemeinen Fall wird zur Lösung der inversen Kinematik der *Levenberg–Marquardt Algorithmus* [90, 105] verwendet, der eine Modifikation des Newton–Raphson Algorithmus ist. Das Paket *GUI-Modules* enthält diejenigen Klassen, die zum Aufbau der graphischen Benutzerschnitte verwendet werden. Hierzu gehört unter anderem ein Eingabefeld für Denavit–Hartenberg Parameter. Intern werden jedoch keine homogenen Transformationsmatrizen verwendet, um die kinematischen Gleichungen zu bestimmen und zu lösen, sondern Dualquaternionen mit denen sowohl eine Rotation um als auch eine Translation entlang eines Richtungsvektors dargestellt werden kann. Das Paket *glprimitives* ermöglicht die dreidimensionale Darstellung von kinematischen Strukturen sowie deren Animation anhand von Bezier–Kurven. Das Paket *kinemath* wird von den anderen Paketen zum Rechnen mit komplexen Zahlen, Polynomen, Vektoren, Matrizen und Quaternionen verwendet. SYNTHETICA ist in JAVA implementiert und weitestgehend plattformunabhängig. Die aktuelle Version 2.1 verwendet die OPENGL Anbindung GL4JAVA für die 3D–Darstellung und wurde unter WINDOWS und MACOS getestet. Sie ist quelloffen und als JAVA Archivdatei frei erhältlich. Das dynamische Robotermodell kann mit SYNTHETICA nicht berechnet werden.

3.1.10 Robotect

Das Softwarepaket ROBOTECT [120, 159] wurde von H. D. Nayar in Altadena, Kalifornien entwickelt. Es enthält Funktionen, um serielle Kinematiken zu modellieren, zu visualisieren, zu animieren und zu analysieren. Dieses proprietäre Softwaresystem läuft unter WINDOWS Betriebssystemen. ROBOTECT verwendet Denavit–Hartenberg Parameter und homogene Transformationsmatrizen, um einen Manipulator zu modellieren. Es werden neben Rotations– und Translationsgelenken auch passive Gelenkachsen unterstützt. Die Bestimmung der inversen Kinematik wird als numerisches Optimierungsproblem betrachtet und über eine Annäherungsformel, die die inverse beziehungsweise Pseudoinverse Jacobi–Matrix verwendet, gelöst. Außerdem werden Algorithmen zur Trajektoriengenerierung basierend auf Polynomfunktionen 5. Grades angeboten. Darüber hinaus kann der Benutzer auch eigene Algorithmen zur Rückwärtstransformation beziehungsweise Trajektoriengenerierung einbinden. Diese Algorithmen können entweder als externe MATLAB

Funktionen oder als externe EXCEL Tabellenblätter eingebunden werden. Als Datenquelle zur Trajektoriengenerierung kann auch eine ASCII–Textdatei verwendet werden. Das inverse dynamische Modell wird mit Hilfe des Newton–Euler Algorithmus bestimmt. Hingegen kann das direkte dynamische Modell *nicht* bestimmt werden. Deswegen können keine dynamischen Simulationen durchgeführt werden. ROBOTECT war ein kommerzielles Softwareprodukt, das inzwischen allerdings nicht länger vertrieben wird.

3.1.11 RobotDraw

ROBOTDRAW wurde von R. Manseur [98, 99, 149, 154] am Department of Electrical and Computer Engineering an der Universität von West Florida zur Visualisierung von kinematischen Ketten entwickelt. Der Benutzer kann damit ein dreidimensionales geometrisches Modell eines Manipulators in einer VRML–Repräsentation erzeugen. Dabei wird der Roboter durch Denavit–Hartenberg Parameter beschrieben, die der Benutzer über einen Webbrowser eingibt. Allerdings kann hierbei kein Gelenkwinkeloffset eingegeben werden. Die Denavit–Hartenberg Parameter werden an ein PERL–Skript übergeben, das auf einem Webserver läuft. Das Skript generiert daraus eine VRML–Datei, die dann innerhalb des Browsers angezeigt wird. In der Visualisierung werden Rotationsgelenke in Form von roten Zylindern und Translationsgelenke in Form von roten Boxen dargestellt. Eine dünne blaue Umrandung am Ende des Gelenks zeigt die positive Dreh– beziehungsweise Translationsrichtung an. Die Hebelverschiebung wird als purpurner Zylinder und die Hebellänge als grüner Zylinder dargestellt. Die aktuelle Version unterstützt Manipulatoren mit drei bis sechs Freiheitsgraden. In Zukunft sollen auch Bewegungsbahnen von Robotern visualisiert werden können. Dieses System ist hauptsächlich für Lehr– und Ausbildungszwecke gedacht und kann ausschließlich zur Visualisierung verwendet werden.

3.2 Projekte im Bereich des symbolischen Rechnens

Eine Reihe von Projekten rechnen zuerst symbolisch und greifen erst bei der Ausführung der hergeleiteten Manipulatormodelle auf numerische Methoden zurück. Auf diese Weise können Formeln vor deren numerischen Auswertung algebraisch vereinfacht und dadurch wertvolle Rechenzeit eingespart werden. Redundante Berechnungen in Algebraausdrücken können ferner durch das Einführen von Hilfsvariablen vermieden werden. Daneben kann auch die benötigte Anzahl an Rechenoperationen von symbolischen Lösungen angegeben werden. Die abgeleiteten, mathematischen Lösungen sind zudem exakt und haben einen höheren Informationsgehalt als rein numerische Ausdrücke. Insbesondere kann auch *ohne* Zahlen gerechnet werden, um dann in der gefundenen Lösung die symbolischen Variablen durch Zahlenwerte, beispielsweise die Abmessungen eines konkreten Roboters, zu ersetzen. Geschlossene Lösungen lassen sich darüber hinaus deutlich besser parallelisieren als iterative Näherungslösungen. Außerdem können viele Probleme, wie beispielsweise Instabilitäts– und Konvergenzprobleme, vermieden werden, die bei einer rein numerischen Berechnung auftreten können. Die Herleitung symbolischer Lösungen lässt sich oft mit entsprechenden Transformationsregeln automatisieren und braucht nicht von Hand durchgeführt zu werden. Der Nachteil des symbolischen Rechnens ist, dass bei Robotern mit vielen Gelenkachsen teilweise lange und unübersichtliche Gleichungen generiert werden. Bei manchen Robotertypen kann auch keine geschlossene Lösung berechnet werden. Es folgt eine Beschreibung der Projekte in diesem Bereich. Diese Projekte sind IKAN, SRAST, ROBOTRAN, ROBOTICA, JFKENGINE, LINKAGE DESIGNER und SPACELIB.

3.2.1 IKAN: Inverse Kinematics using Analytical Solutions

Die IKAN Bibliothek [166, 167] verwendet eine Kombination von analytischen und numerischen Lösungsverfahren, um die Rückwärtstransformation von *anthropomorphen Gliedern* wie zum Beispiel dem menschlichen Arm oder Bein zu bestimmen. Durch diese Kombination werden effizientere Lösungen berechnet, als bei einer ausschließlich numerischen Rechnung. IKAN wurde an der Universität von Pennsylvania entwickelt, um kinematische Ketten mit sieben Rotationsgelenken, die in Schulter, Ellbogen und Arm unterteilt sind, zu lösen. Das Schulter– und das Armgelenk haben jeweils drei Freiheitsgrade und werden über das Ellbogengelenk, das einen Freiheitsgrad besitzt, verbunden. Für beliebige Kinematiken kann IKAN nicht eingesetzt werden. Die Roboterdynamik kann ebenfalls nicht simuliert werden. Die Software ist für nichtkommerzielle Zwecke frei erhältlich. Sie ist in C++ geschrieben und wurde unter IRIX, SUNOS und WINDOWS getestet.

3.2.2 SRAST: Symbolic Robot Arm Solution Tool

SRAST [70] ist ein System zum analytischen Lösen von kinematischen Gleichungen und wurde am Department of Electrical Engineering an der Universität von Pittsburgh entwickelt. Es ist in der funktionalen Programmiersprache COMMON LISP [80] geschrieben. Das System erwartet als Eingaben die Denavit–Hartenberg Parameter von seriellen Robotern und berechnet das direkte und gegebenfalls inverse kinematische Modell symbolisch. Hierbei kann der Benutzer allerdings weder einen Gelenkwinkeloffset noch passive Transformationen spezifizieren. Außerdem kann nicht für jeden kinematisch nicht–redundanten Roboter eine Lösung gefunden werden. Es können nur Gleichungstypen gelöst werden, deren Lösung in Form von prozeduralen Wissen bekannt ist. SRAST benötigt zur Bestimmung des kinematischen Modells eines Unimate Puma 600 sehr viel Rechenzeit. Für die symbolische Multiplikation von sechs homogenen 4×4 Transformationsmatrizen, um die direkte Kinematik dieses Roboters zu lösen, werden mehr als fünf Minuten benötigt. Andere Komponenten einer Bewegungssteuerung, wie zum Beispiel Komponenten zur Dynamikberechnung oder Trajektoriengenerierung, werden in diesem Projekt nicht berücksichtigt. Inzwischen wurde das Projekt eingestellt und ist nicht mehr erhältlich.

3.2.3 Robotran

ROBOTRAN [50, 51, 152, 164] ist ein symbolischer Modellgenerator für Mehrkörpersysteme. Es wurde am Forschungszentrum Mechatronik (CEREM) an der Katholischen Universität von Louvain–la–Neuve (UCL) in Belgien entwickelt und in der Programmiersprache C implementiert. Um Terme symbolisch umzuformen, wird ein selbst entwickeltes Computeralgebrasystem verwendet, das speziell auf Dynamikberechnungen ausgelegt ist. Als Gelenkarten sind Rotations– und Translationsgelenke sowie unbewegliche Gelenke (engl.: locked joint) verfügbar. ROBOTRAN erwartet als Eingabe eine Textdatei, die nach einer ganz bestimmten Struktur aufgebaut sein muß und in der die Topologie eines mechanischen Mehrkörpersystems beschrieben wird. Ausgehend von dieser Beschreibung kann die direkte und inverse Dynamik von offenen und geschlossenen Kinematiken sowie von Baumstrukturen mit dem rekursiven Newton–Euler Verfahren bestimmt werden. Um die Gleichungen effizienter zu machen und Mehrfachberechnungen zu vermeiden, werden bei der Generierung Hilfsvariablen eingeführt. ROBOTRAN verwendet *nicht* die Denavit–Hartenberg Konvention und löst die inverse Kinematik durch eine MATLAB–Routine nur numerisch. Es kann ausschließlich über einen Webbrowser bedient werden.

3.2.4 Robotica for Mathematica

Ein weiteres Projekt in diesem Kontext ist ROBOTICA [121, 122, 123, 153], das das kommerzielle Computeralgebrasystem MATHEMATICA 2.1 für symbolische Berechnungen verwendet. ROBOTICA wurde am Coordinated Science Laboratory an der Universität von Illinois in Urbana–Champaign entwickelt. Man kann mit diesem Paket die Vorwärtstransformation, die Jacobi–Matrix und die dynamischen Gleichungen nach Lagrange anhand der Denavit–Hartenberg Parameter [34] eines seriellen Manipulators und weiterer Eingabedaten symbolisch berechnen. Die Eingabe und Bedienung erfolgt über eine graphische Oberfläche, dem ROBOTICA FRONT END. Es kann bei den DH–Parametern jedoch kein Gelenkwinkeloffset angegeben werden. Die Rückwärtstransformation kann nicht berechnet werden und es werden auch keine Funktionalitäten zur automatischen Trajektoriengenerierung angeboten. Ferner können keine physikalischen Simulationen bezüglich des dynamischen Verhaltens von Robotern durchgeführt werden. In der Zwischenzeit wird dieses Projekt nicht mehr länger verfolgt oder unterstützt. Da die derzeitige MATHEMATICA Version 5.2 ist, kann dieses Projekt als etwas veraltet angesehen werden.

3.2.5 JFKengine: Jacobian and Forward Kinematics Generator

JFKENGINE [81, 141] ist ein System, um die Jacobi–Matrix und die direkte Kinematik von seriellen Manipulatoren analytisch zu bestimmen und wurde am Oak Ridge National Laboratorium in Knoxville, Tennessee entwickelt. Als Implementierungssprache wurde C++ gewählt und diese Sprache um Funktionen zum symbolischen Rechnen erweitert. Als Gelenkarten sind Rotations– und Translationsgelenke möglich. Die Funktionalitäten von JFKENGINE werden in einer Programmierschnittstelle zur Verfügung gestellt. Die Methode *getForwardKinematics* berechnet auf Basis der Denavit–Hartenberg Parameter eines Manipulators eine homogene Transformationsmatrix, die die Position und Orientierung des Endeffektors im Basiskoordinatensystem angibt. Die Methode *getJacobian* berechnet die Jacobi–Matrix eines Manipulators durch Differentiation und Simplifikation der Ausdrücke der direkten Kinematik. Durch die Option *includeOrientation* kann angegeben werden, ob auch die Orientierungsanteile berechnet werden sollen. Falls ja, wird als Ergebnis eine $6 \times n$ Matrix und andernfalls eine $3 \times n$ Matrix zurückgegeben. Mit der Methode *getJointLocations* wird die direkte Kinematik anhand konkreter Zahlen für die Gelenkvariablen ausgewertet. Die inverse Kinematik kann mit JFKENGINE nicht bestimmt werden. Das System wurde unter der LINUX–Distribution REDHAT 8.0 entwickelt.

3.2.6 Linkage Designer

LINKAGE DESIGNER [45, 46] ist ein MATHEMATICA Zusatzpaket, um mechanische Verbindungen, die entweder seriell oder baumförmig aufgebaut sind, zu analysieren und zu simulieren. Es wurde von G. Erdõs an der Ungarischen Akademie der Wissenschaften in Budapest entwickelt. Mit dem MATHEMATICA Paket DYNAMIC VISUALIZER oder einem VRML–Browser können Gelenkverbindungen visualisiert und animiert werden. Als Gelenkarten sind Rotations–, Translations–, Universal–, Planar–, Scharnier– und Kugelgelenke sowie passive Gelenke verfügbar. Für beliebige Gelenke können Geschwindigkeiten, Beschleunigungen und Ableitungen höherer Ordnung in geschlossener Darstellung bestimmt werden. LINKAGE DESIGNER ist ein kommerzielles Produkt, das MATHEMATICA 5.0 oder höher benötigt. Es läuft unter den Plattformen WINDOWS, MACOS und LINUX. Zur Berechnung des dynamischen Robotermodells kann dieses Paket nicht verwendet werden.

3.2.7 SpaceLib in Maple

SpaceLib [88, 89] ist eine Softwarebibliothek für die kinematische und dynamische Analyse von Festkörpersystemen und wurde von G. Legnani an der Universität von Brescia in Italien entwickelt. SpaceLib basiert auf der Denavit–Hartenberg Konvention. Allerdings wurden in SpaceLib zusätzliche 4×4 Matrizen eingeführt. Die Linear– und Winkelgeschwindigkeit eines Körpers hinsichtlich eines Referenzkoordinatensystems wird in der Geschwindigkeitsmatrix angegeben. Hierzu entsprechend wird die Beschleunigungsmatrix definiert, die die Linear– und Winkelbeschleunigung eines Körpers bezüglich eines Koordinatensystems enthält. Die Kräfte, die auf einen Körper wirken, wird durch die schiefsymmetrische Aktionsmatrix beschrieben. Entsprechend werden die auf einen Körper wirkenden Impulse in der schiefsymmetrischen Impulsmatrix zusammengefasst. Die Massenverteilung eines Körpers wird in der Trägheitsmatrix dargestellt. Mit diesen Matrizen wird die Dynamik nach dem Lagrange–Verfahren bestimmt. Beispielsweise wird die kinetische Enerige eines Körpers als Funktion der Geschwindigkeits– und Trägheitsmatrix berechnet. Von SpaceLib gibt es in der Zwischenzeit insgesamt drei Versionen. Eine Version rechnet symbolisch und benötigt Maple 9.5 oder höher. Darüber hinaus gibt es zwei Versionen, die in der Programmiersprache C beziehungsweise in Matlab rein numerisch rechnen. Der Quellcode ist frei verfügbar und darf nicht–kommerziell genutzt werden.

3.3 Mehrkörpersimulationssysteme

Mehrkörpersimulationssysteme werden eingesetzt, um die dynamischen Eigenschaften von mechanischen Systemen, die in starre Körper, Gelenke und Kraftelemente aufgeteilt werden, zu untersuchen. Zu den Mehrkörpersimulationssystemen, die sich in ihrer Funktionalität und der Art wie Bewegungsgleichungen aufgestellt und gelöst werden, unterscheiden, gehören zum Beispiel Dymola, DynaFlexPro, NewEul, MBSymba und Adams.

3.3.1 Dymola: Dynamic Modelling Laboratory

Dymola [42, 43, 44, 128] ist eine Modellierungs– und Simulationssystem, das auf der objektorientierten Modellierungssprache Modelica basiert. Es besteht aus einer Reihe von Bibliotheken um beispielsweise Fahrzeuge oder elektrische Schaltungen zu modellieren. Die *Mehrkörpersystembibliothek* enthält grundlegende Modellklassen, wie zum Beispiel Festkörper, rotatorische und translatorische Gelenke inklusive vieler Reibelemente sowie Animationselemente, um Roboter und andere mechatronische Systeme zu modellieren. Die geometrische Modellierung erfolgt *nicht* nach der Denavit–Hartenberg Konvention. Stattdessen werden in einem graphischen Editor Grundobjekte aus Bibliotheken ausgewählt, parametriert und über physikalische Konnektoren zu Modellen miteinander verbunden. Jedes Grundobjekt wird durch einen Satz von mathematischen Gleichungen beschrieben. Durch die Verbindung von Grundobjekten werden zusätzliche Gleichungen erstellt und entsprechend der Verschaltungsstruktur miteinander kombiniert. Es gibt Algorithmen, um diese Modellgleichungen symbolisch in Zustandsform zu transformieren und C–Code zu generieren. Außerdem stehen viele numerische Integrationsverfahren für die Lösung von Differentialgleichungen zur Verfügung, beispielsweise *Euler–*, *Runge–Kutta–* und *DASSL–Verfahren (Differential Algebraic System Solver)* [138]. Das kinematische Robotermodell wird von Dymola nicht berechnet. Seit 1992 wird es von der schwedischen Firma Dynasim AB in Lund für Windows und Linux Betriebssysteme kommerziell vertrieben.

3.3.2 DynaFlexPro

DynaFlexPro [10, 113] ist ein Maple Zusatzpaket für die Modellierung und Simulation der Dynamik von mechanischen Mehrkörpersystemen, die an der Universität von Waterloo in Kanada entwickelt wurde. Die Erstellung von Systemmodellen erfolgt mit dem DynaFlexPro Model–Builder, einer graphischen Benutzeroberfläche zur Verschaltung von elementaren Blöcken zu Blockdiagrammen. Die Geometrie eines Roboters wird mit der Denavit–Hartenberg Konvention beschrieben. Allerdings können diese Parameter nicht direkt in den DynaFlexPro Model–Builder eingegeben werden. Stattdessen wird ein Manipulator durch mit Linien verbundenen Blöcke dargestellt, wobei jeder Block ein Gelenk darstellt. Die Denavit–Hartenberg Paramter werden in eine interne Parameterdarstellung konvertiert, um diese Werte dann jedem Block zuzuweisen. Aus dem Systemmodell werden automatisch die symbolischen Bewegungsgleichungen generiert. Hierfür werden intern Projektionsmethoden verwendet, das heißt das Newton–Euler Verfahren und Methoden aus der linearen Graphentheorie. Die Differentialgleichungen können mit zwölf verschiedenen Verfahren integriert werden. Darunter ist auch das Runge–Kutta–Fehlberg Verfahren. Reibungskräfte sind bislang nicht implementiert, sollen aber in einer zukünftigen Version mit einbezogen werden. Während die direkte Kinematik automatisch gelöst werden kann, kann eine geschlossene Lösung der inversen Kinematik nicht bestimmt werden. Das Newton–Raphson Verfahren wird ebenfalls nicht unterstützt. DynaFlexPro ist ein kommerzielles Produkt und kostet mehrere Tausend Euro. Die aktuelle Version 3.0 benötigt Maple 10 oder höher und läuft unter Windows, Linux und MacOS.

3.3.3 NewEul

NewEul [71] ist ein System, um die Dynamik von Mehrkörpersystemen mit dem Newton–Euler Verfahren und den Prinzipien von d'Alembert and Jourdain zu bestimmen. Es wurde am Institut für Technische und Numerische Mechanik an der Universität Stuttgart entwickelt und in der Programmiersprache Fortran77 implementiert. Anhand einer Eingabedatei, die aus den Abschnitten *Allgemeine Angaben*, *Koordinatensysteme*, *Massengeometrische Größen* und *Eingeprägte Kräfte und Momente* besteht, werden die Bewegungsgleichungen symbolisch erzeugt. Das dynamische Verhalten kann durch Integration der Bewegungsgleichungen mit dem Modul NewSim simuliert werden. In NewEul wird die Denavit–Hartenberg Konvention *nicht* verwendet und es werden keine Gelenktypen unterschieden. Allerdings erlaubt das System sowohl eine relative als auch eine absolute Definition von Koordinatensystemen. Jedes Koordinatensystem kann bis zu sechs Freiheitsgrade haben. Das Modell wird aus einer Verknüpfung von Koordinatensystemen aufgebaut. Mehrkörpersysteme können ketten– oder baumförmig aufgebaut sein und es werden auch geschlossene Strukturen unterstützt. Ferner besteht die Möglichkeit, lange Gleichungen zu komprimieren, die bei sehr komplizierten Mehrkörpersystemen generiert werden. Der Benutzer kann zwischen sieben verschiedenen Komprimierungsstufen auswählen, um Gleichungen unterschiedlich stark komprimieren zu lassen. Während in der nullten Stufe keine Komprimierung vorgenommen wird, wird in der sechsten Stufe eine maximale Komprimierung durchgeführt, wodurch allerdings trigonometrische Vereinfachungen nicht mehr möglich sind. Diese Stufe kommt deswegen einer ausschießlich numerischen Berechnung sehr nahe. Reibungskräfte werden in NewEul nicht berücksichtigt. Eine eingeschränkte Demoversion für Mehrkörpersysteme mit bis zu vier Freiheitsgraden ist für Windows und Linux erhältlich. Eine kommerzielle Lizenz kostet mehrere Hundert Euro.

3.3.4 MBSymba: Symbolic Modelling of Multibody Systems

MBSymba [91] ist eine Maple Erweiterung zur symbolischen Modellierung von Mehrkörpersystemen und wurde von R. Lot an der Universität Padua in Italien entwickelt. Es besteht aus einer Reihe von Methoden für die Beschreibung eines Systems, für die Herleitung der Bewegungsgleichungen und für die Analyse der Systemgleichungen. Manipulatoren können *nicht* nach der Denavit–Hartenberg Konvention beschrieben werden. Stattdessen erfolgt die Beschreibung eines Systems mit einer Reihe von Prozeduren, um Objekte (Punkte, Vektoren, Körper, Kräfte und Momente) und Objektbeziehungen zu definieren. Als Berechnungsverfahren für die Dynamik werden sowohl die Lagrange'schen Gleichungen als auch der Newton–Euler–Formalismus verwendet. Weiterhin gibt es Prozeduren für die Linearisierung der Gleichungen, ihre Konversion in den Zustandsraum und die Berechnung der Systemantwort. Die Software unterliegt den GNU GPL–Bedingungen.

3.3.5 Adams: Automatic Dynamic Analysis of Mechanical Systems

Das Simulationspaket Adams [151] besteht aus einer Vielzahl von Modulen, um Modelle von Mehrkörpersystemen zu erstellen, deren physikalisches Verhalten zu simulieren und die Ergebnisse zu interpretieren. Die Denavit–Hartenberg Konvention wird *nicht* verwendet, um Roboter zu modellieren. Das Modul Adams View dient zur Erstellung von Mehrkörpermodellen und zur Visualisierung von Bewegungsabläufen. Neben der Neuentwicklung von Modellen können Geometriedaten auch aus den meisten CAD–Systemen importiert werden. Die Bewegungsgleichungen werden dann nach dem Lagrange Verfahren aufgestellt, wobei es auch eine Option gibt, um Reibungskräfte definieren zu lassen. Mit dem Modul Adams Solver werden die erzeugten Gleichungen numerisch gelöst. Das Modul Adams Postprozessor ist für die Nachbearbeitung von Daten. Außerdem gibt es spezielle Module, die auf bestimmte Anwendungsbereiche zugeschnitten sind, wie Adams Car für Kraftfahrzeuge oder Adams Rail für Schienenfahrzeuge. Das kinematische Modell kann von Adams nicht hergeleitet werden, sondern muß vom Benutzer selbst implementiert werden. Im Gegensatz zu anderen Mehrkörpersimulationssystemen kann Adams nur numerisch und nicht symbolisch rechnen. Die Software wird von der Firma MSC.Software vertrieben und ist für Unix und Windows Betriebssysteme erhältlich.

3.4 Physik–Engines

Zu den bisher beschriebenen Projekten sind physikalische Simulatoren verwandt, mit denen das dynamische Verhalten von Robotern unter Verwendung von Differentialgleichungen in einer Simulation untersucht wird. In der Simulation werden meistens auch Kollisionen und Reibungskräfte berücksichtigt. Es kommen verschiedene Algorithmen zur Integration von Differentialgleichungen zum Einsatz, die sich in der Rechengeschwindigkeit und Rechengenauigkeit sowie im Konvergenzverhalten unterscheiden. Bei *Einschrittverfahren* werden keine zeitlich weiter zurückliegende Werte benutzt, während bei *Mehrschrittverfahren* auch Vergangenheitswerte in die Integration miteinbezogen werden. Außer in der Robotik werden physikalische Simulatoren und Physik–Engines auch in der Automobil– und Luftfahrttechnik sowie in der Computeranimation eingesetzt. Die meisten dieser physikalischen Simulatoren, wie DynaMo, MBDyn, ODE und DynaMechs, rechnen ausschließlich numerisch. EasyDyn rechnet hingegen teilweise auch symbolisch.

3.4.1 DynaMo: Dynamic Motion Library

DynaMo [7] ist eine Bibliothek zur Simulation physikalischer Systeme, die in einer Dissertation an der Technischen Universiteit Eindhoven entwickelt wurde. Körper werden als punktförmige Massen und dazugehörige Trägheitstensoren dargestellt. Für die Dynamiksimulation müssen die angreifenden Kräfte und Drehmomente angegeben werden. DynaMo bietet für die Integration von Differentialgleichungen vier verschiedene numerische Integratoren an, und zwar zwei verschiedene Euler–Verfahren sowie das Runge–Kutta–Verfahren der Ordnung zwei und vier. Für die Kollisionserkennung wird die ebenfalls an der Technischen Universiteit Eindhoven entwickelte und frei erhältliche Bibliothek FreeSolid (Software Library for Interference Detection) [172] verwendet. Nachdem eine Kollision erkannt wurde, berechnet DynaMo die resultierenden Reaktionskräfte. Die Bibliothek ist vollständig in der Programmiersprache C/C++ geschrieben und als GNU LGPL–basierte Software veröffentlicht. Reibungskräfte werden von dieser Bibliothek nicht abgedeckt.

3.4.2 MBDyn: MultiBody Dynamics Software

MBDyn [58, 59, 117] wurde an der Polytechnischen Universität Mailand entwickelt. Es können sowohl offene und geschlossene Kinematiken als auch baumförmige kinematische Strukturen nach dem Newton–Euler Verfahren modelliert werden. Viele verschiedene Gelenkarten sind verfügbar. Reibungskräfte können auf Basis des *Coulomb–Modells* oder des *LuGre–Modells* berechnet werden. Die Integration der Differentialgleichungen erfolgt durch das *BDF–Verfahren (Backward Difference Formula)* [67]. Der Quellcode ist als GNU GPL erhältlich und kann unter Unix–Betriebssystemen ausgeführt werden. Mit der Erweiterung RT–MBDyn [5], die unter dem Betriebssystem RTAI läuft, können harte Echtzeitanforderungen erfüllt werden. Im Gegensatz zu anderen Physik–Engines handelt es sich nicht um eine Bibliothek, sondern um ein kommandozeilenbasiertes Programm.

3.4.3 ODE: Open Dynamics Engine

ODE [33] ist eine von dem Physiker R. Smith in der Programmiersprache C/C++ entwickelte Softwarebibliothek für die Bewegungssimulation von mechanischen Körpern. Sie ist als BSD–Lizenz (Berkeley Software Distribution) und optional als GNU LGPL erhältlich. Die Bibliothek unterstützt verschiedene Gelenktypen, zum Beispiel Kugel–, Scharnier– und Translationsgelenke sowie fixierte Verbindungen, Kollisionserkennung zwischen Objekten sowie ein Berührungs– und Reibungsmodell. Die Orientierung von Körpern wird entweder als Rotationsmatrix oder als Quaternion dargestellt. Jedem Gelenk werden drei Gleichungen zugeordnet, die die zulässigen Positionen und Orientierungen der beiden Körper, die das Gelenk miteinander verbindet, definieren. Zur Integration der Differentialgleichungen wird das einstufige *Euler–Verfahren* verwendet. Für die Kollisionserkennung ist es erforderlich, dass jeder Körper eine geometrische Form hat. Kollisionsobjekte können beispielsweise Kugeln, Quader, Zylinder oder Strahlen sein. In jedem Simulationsschritt werden Kollisionserkennungsfunktionen aufgerufen, die bei Berührungen von Körpern eine Liste von Kontaktpunkten zurückliefern. Kontaktpunkte werden durch Position, Richtungsvektor und Eindringtiefe angegeben. Die Visualisierung des zu simulierenden Systems erfolgt mit OpenGL (Open Graphics Library). Durch die Bibliothek ODEJava können die Funktionen auch in Java–Programmen verwendet werden. Die inverse Kinematik kann von ODE nicht gelöst werden. ODE kann unter Windows, MacOS und Unix genutzt werden. Echtzeitfähige Betriebssysteme werden nicht unterstützt.

3.4.4 DynaMechs: Dynamics of Mechanisms

An der Universität von Ohio entwickelte S. McMillan die Softwarebibliothek DynaMechs [109, 110, 111, 112] für die Dynamiksimulation von mechanischen Systemen. Es können sowohl offene kinematische Ketten als auch komplizierte Baumstrukturen simuliert werden. D. Marhefka [104] erweiterte das System, so dass nun auch bestimmte geschlossene Kinematiken simuliert werden können. Gelenke mit einem Freiheitsgrad (Translations– und Rotationsgelenke) werden nach der Denavit–Hartenberg Konvention beschrieben. Darüber hinaus werden auch Kugelgelenke (engl.: ball–and–socket joints) unterstützt, die durch drei Eulerwinkel und einen Positionsvektor beschrieben werden. Die Bibliothek bietet auch eine laufzeitoptimierte Kollisionserkennung an. Um die Laufzeit der Kollisionserkennung bei der Dynamiksimulation zu reduzieren, wird vor jedem Integrationsschritt nicht die gesamte Oberfläche von mechanischen Systemen auf Kollisionen überprüft, sondern nur ausgewählte Kontaktpunkte. Die numerische Integration der Bewegungsgleichungen erfolgt wahlweise mit dem *Euler–Verfahren* oder dem *Runge–Kutta–Verfahren vierter Ordnung mit fester oder adaptiver Schrittweite*. Bei adaptiver Schrittweite wird dieser Wert automatisch an das Verhalten der Lösung angepasst und dabei vorgegebene Genauigkeitsanforderungen berücksichtigt. Da im Gegensatz zu anderen Bibliotheken mit DynaMechs auch mechanische Systeme unter Wasser simuliert werden können, wird die Bibliothek hauptsächlich für den Entwurf von Unterwasserrobotern eingesetzt. Die Bibliothek ist in C/C++ implementiert und der Quellcode ist unter der GNU GPL erhältlich. Es werden die Plattformen Linux, Solaris, Irix und Windows NT unterstützt.

3.4.5 EasyDyn: Easy Simulation of Dynamic Problems

EasyDyn [174, 175, 176] ist eine C/C++ Bibliothek zur Simulation von Mehrkörpersystemen, die mathematisch mit Differentialgleichungen 2. Ordnung beschrieben werden. Die Bibliothek wurde von O. Verlinden an der Polytechnischen Fakultät von Mons in Belgien entwickelt und ist als kostenlos nutzbare Open–Source–Software entsprechend den Bedingungen der GPL erhältlich. Für numerische Berechnungen wird die Mathematikbibliothek GNU Scientific Library (GSL) verwendet. EasyDyn ist in vier Module gegliedert. Das Modul *vec* enthält Operationen für das Rechnen mit dreidimensionalen Vektoren und homogenen Transformationsmatrizen. Das Modul *sim* enthält Routinen for die numerische Integration von Differentialgleichungen 2. Ordnung, die auf dem *Newmark–Verfahren* [124] basieren. Das Modul *mbs* berechnet numerisch die Bewegungsgleichungen aus einem C/C++ Programm, in dem die Positionen, Geschwindigkeiten und Beschleunigungen der Massenschwerpunkte der Teilkörper eines mechanischen Systems und die extern wirkenden Kräfte angegeben sind. Die Bewegungsgleichungen werden durch die Anwendung des *d'Alembert–Prinzips* [2] konstruiert. Dieses Modul liefert also die Eingaben für das Modul *sim*. Das Modul *visu* dient zur Generierung von Dateien mit denen dreidimensionale Objekte visualisiert und animiert werden können. Die eigentliche Visualisierung erfolgt mit dem System EasyAnim, das auch von O. Verlinden entwickelt wurde. Das Projekt rechnet mit einer Ausnahme ausschließlich numerisch. Ausdrücke für Geschwindigkeiten und Beschleunigungen können anhand homogener Transformationsmatrizen mit Hilfe des Paderborner Computeralgebrasystems MuPad (Multi Processing Algebra Data Tool) [119] symbolisch hergeleitet werden. Die Herleitung erfolgt mit dem Skript *CAGeM (Computer Aided Generation of Motion)*. EasyDyn läuft unter Windows und Linux Betriebssystemen und wurde zum Einsatz in Lehre und Ausbildung entwickelt.

3.5 3D–Robotersimulationssysteme

Die Zielsetzung von Robotersimulationssystemen ist im Vergleich zu den bisher beschriebenen Projekten etwas anders gelagert. Das Haupteinsatzgebiet von diesen Systemen ist die Offline–Programmierung von Industrierobotern und die Layoutplanung von Roboterzellen. Hierzu gehören unter anderem Untersuchungen bezüglich Zykluszeit, Erreichbarkeit sowie Kollisions– und Singularitätsfreiheit. Eine wichtige Anforderung an diese Systeme ist, dass der virtuelle Roboter in der Simulation sich mit denselben Gelenkkonfigurationen bewegt, wie später der reale Roboter in der Anlage. Ansonsten können Kollisionen auftreten. Die Dynamik wird in diesen Systemen größtenteils *nicht* berücksichtigt. Es gibt zwei Arten von Robotersimulationssystemen und zwar herstellerspezifische und herstellerübergreifende Systeme. Mit der KUKA SIM FAMILIE und ABB ROBOTSTUDIO können nur die Roboter des jeweiligen Herstellers programmiert werden. Für Roboter anderer Hersteller oder für selbst definierte Kinematiken muss die Steuerungssoftware vom Benutzer implementiert werden. In EASY–ROB, EM–WORKPLACE, ROBOMOSP, COSIMIR, IGRIP und MICROSOFT ROBOTICS STUDIO sind hingegen auch Roboter von verschiedenen Herstellern verfügbar. Im folgenden werden ausschließlich die herstellerübergreifenden Robotersimulationssysteme betrachtet, weil diese Systeme allgemeiner ausgerichtet sind.

3.5.1 EASY–ROB 3D Robot Simulation Tool

In dem System EASY–ROB [3] der gleichnamigen Firma aus Frankfurt am Main sind mathematische Lösungen bezüglich Kinematik, Bewegungsplanung und Ausführung für die industrielle Robotik implementiert. Zu den Anwendungsbereichen gehören die Simulation und Offline–Programmierung von Robotern sowie die Layoutplanung von Roboterzellen. Es können auch eigene Kinematiken erstellt und simuliert werden. Kinematische Ketten werden nach der Denavit–Hartenberg Konvention beschrieben. Der Benutzer kann die Lösung der inversen Kinematik aus einer Bibliothek von vorhandenen Robotern laden beziehungsweise selbst in der Programmiersprache C/C++ implementieren und über eine Anwendungsprogrammierschnittstelle einbinden. Außerdem kann die inverse Kinematik näherungsweise mit der Jacobi–Matrix berechnet werden. Bei mehr als sechs Gelenkachsen wird die Pseudoinverse verwendet und diese Matrix mittels Singulärwert–Zerlegung berechnet. In EASY–ROB werden die gängigen Programmiersprachen für Industrieroboter unterstützt. Als Interpolationsarten stehen synchrone Punkt–zu–Punkt Bewegungen sowie lineare und zirkulare Bahninterpolation zur Verfügung. EASY–ROB benutzt die Grafikbibliothek OPENGL zur Visualisierung von Robotern und läuft unter MICROSOFT WINDOWS und unter Betriebssystemen, die es erlauben, WINDOWS Programme auszuführen. Es ist keine Open–Source–Software und für die Nutzung muß eine Lizenz erworben werden. Darüber hinaus kann eine Demoversion kostenfrei heruntergeladen werden.

3.5.2 eM–Workplace / Robcad

EM–WORKPLACE [41, 79], das früher ROBCAD hieß, ist eine Robotersimulationssoftware der Firma TECNOMATIX. Für die Modellierung von Fertigungszellen stehen umfangreiche Roboter–, Maschinen–, Werkzeug– und Bestückungsbibliotheken zur Verfügung. Darüber hinaus kann der Benutzer weitere Roboter und Fertigungseinrichtungen modellieren. Unterschiedliche Bewegungen können simuliert und Kollisionsuntersuchungen durchgeführt werden. Als Betriebssysteme werden WINDOWS 2000 und XP sowie UNIX unterstützt.

3.5.3 Robomosp: Robotics Modelling and Simulation Platform

ROBOMOSP [78] ist ein System für die Modellierung und Simulation von Manipulatoren und wurde von der Forschungsgruppe Robotik und Automation an der Universität Javeriana in Kolumbien entwickelt. Aus Performanzgründen wurden die meisten Komponenten in ANSI C implementiert. Die graphische Benutzeroberfläche wurde in der Skriptsprache TCL/TK (TOOL COMMAND LANGUAGE/TOOL KIT) geschrieben und für die Visualisierung von Robotern wurde die Graphikbibliothek OPENGL verwendet. Die mathematische Beschreibung serieller Kinematiken erfolgt nach der Denavit–Hartenberg Konvention. Der Benutzer kann zwischen rotatorischen und translatorischen Gelenktypen auswählen. Die inverse Kinematik wird im allgemeinen näherungsweise mit Hilfe der Jacobi–Matrix berechnet. Darüber hinaus kann der Benutzer eigene numerische oder analytische Lösungen über eine Anwendungsprogrammierschnittstelle in das System integrieren, um die vorhandene Lösung zu ersetzen. Die Dynamik wird mit dem rekursiven Newton–Euler Verfahren berechnet und die Differentialgleichungen mit dem Runge–Kutta–Verfahren integriert. Die Offline–Programmierung von Robotern erfolgt in der standardisierten Sprache INDUSTRIAL ROBOT LANGUAGE (IRL) und wird durch die graphische Simulationsumgebung und dreidimensionale CAD–Modelle unterstützt. Gegenwärtige Arbeiten beinhalten die Erweiterung des Systems um verschiedene Roboterprogrammiersprachen und um Kollisionserkennungsverfahren sowie die Erstellung einer Bibliothek mit Robotern von verschiedenen Herstellern. ROBOMOSP läuft unter LINUX, MACOS und WINDOWS Betriebssystemen.

3.5.4 Cosimir: Cell Oriented Simulation of Industrial Robots

COSIMIR [54, 55, 56] ist ein Robotersimulationssystem, das am Institut für Roboterforschung an der Universität Dortmund entwickelt wurde und gegenwärtig von FESTO CORPORATION vertrieben wird. Dieses System ist primär für die Simulation von Arbeitsabläufen mit Robotern entworfen worden. Es unterstützt mehrere Roboterprogrammiersprachen und enthält eine Bibliothek mit vielen verschiedenen Robotertypen von namhaften Herstellern. Die Rückwärtstransformation wird für alle Roboter in dieser Bibliothek berechnet. Darüber hinaus können Roboter aus Rotations– und Translationsgelenken zusammengesetzt und mit Hilfe von Denavit–Hartenberg Matrizen kinematisch beschrieben werden. Es werden dann sehr viele Aufbaumöglichkeiten von der Kinematikerkennung automatisch detektiert. Die erkannten Kinematiken können von der enthaltenen analytischen Universalsteuerung bewegt werden. Für nicht erkannte Kinematiken besteht die Möglichkeit, sie von der Universalsteuerung auf Einzelachsebene zu bewegen. In diesem Fall sind nur PTP–Bewegungen ohne Achssynchronisation möglich. Wenn eine externe Steuerung für eine nicht erkannte Kinematik in Software oder in Hardware existiert, kann diese Steuerung über die OPC–Schnittstelle (Object Linking and Embedding for Process Control) verwendet werden. Die Bewegungsgleichungen, die von der internen Steuerung verwendet werden, sind für den Benutzer nicht zugreifbar. Außerdem kann COSIMIR nicht um selbst entwickelte Kinematikmodelle ergänzt werden, das heißt es gibt keine Anwendungsschnittstelle, die Zugriff auf die Universalsteuerung und deren Bahnplanungsmodul ermöglicht. Das dynamische Robotermodell wird von COSIMIR nicht berechnet. Bei der Trajektoriengenerierung wird Überschleifen unterstützt, allerdings keine Spline– oder Nurbs–Bahnen. COSIMIR läuft auf WINDOWS–Systemen ab Version 2000. Es ist ist keine Open–Source–Software, sondern ein kommerzielles Produkt für den industriellen Einsatz. Der Preis richtet sich nach den für die jeweilige Aufgabenstellung benötigten Modulen.

3.5.5 Igrip: Interactive Graphics Robot Instruction Program

IGRIP [24] ist eine Simulationsumgebung mit dreidimensionaler Darstellung für den Entwurf und die Offline–Programmierung von Roboterzellen, die von der Firma DELMIA entwickelt und vertrieben wird. In diesem System sind auch Funktionen zur automatischen Kollisionserkennung und zur Überwachung von Mindestabständen integriert. Eine Bibliothek enthält über 500 Industrierobotermodelle sowie zusätzliche Geräte, wie zum Beispiel Laufschienen, Portalkräne und Spannvorrichtungen für Werkstücke. Der Benutzer kann Bauteile aus gängigen CAD–Systemen nach IGRIP importieren und daraus eigene Geräte erstellen sowie aus mehreren Geräten eine Werkzelle aufbauen. Die Programmierung von Simulationen erfolgt mit der GRAPHIC SIMULATION LANGUAGE, die eine PASCAL–ähnliche Sprache mit zusätzlichen Datentypen wie Positionen und Pfaden ist. Technische Prozesse wie Punktschweißen, Lackieren und Materialabtrag können simuliert werden.

3.5.6 Microsoft Robotics Studio

MICROSOFT ROBOTICS STUDIO [118] ist eine WINDOWS–basierte Entwicklungsumgebung zur Simulation und Ansteuerung von Robotern, die MICROSOFT als Standard in der Robotik durchsetzen möchte. Es umfasst eine graphische, datenflußbasierte Programmierumgebung *(Visual Programming Language, VPL)* mit der Dienste *(Services)* und Kontrollflußelemente *(Basic Activities)* zu einem Programm verschaltet werden können. Diese Umgebung wurde von MICROSOFT eingeführt, um Software für Roboter zu entwickeln, ohne Programmcode schreiben zu müssen. Darüber hinaus kann die Robotersoftware auch in einigen Programmiersprachen wie zum Beispiel C# oder VB.NET geschrieben werden. Ein Konfigurationseditor *(Decentralized Software Services Manifest Editor, DSSME)* soll die Konfiguration der Dienste für verschiedene Roboterplattformen erleichtern. Die Kommunikation zwischen Diensten erfolgt ausschließlich durch Nachrichtenaustausch. Hierfür wird ein Kommunikationsprotokoll *(Decentralized Software Services Protocol, DSSP)* verwendet, bei dem Nachrichten und Daten zuerst in XML serialisiert und dann mit dem SOAP–Protokoll an einen anderen Dienst gesendet werden. Jeder Dienst befindet sich in einem Zustand und schickt, sobald sein Zustand sich ändert, eine Nachricht darüber an alle Dienste, die sich bei ihm angemeldet haben. Dienste können auf demselben oder auf verschiedenen Rechnern ablaufen, beispielsweise um mehrere Roboter anzusteueren. Inzwischen werden auch Dienste für die Spracherkennung und das maschinelle Sehen angeboten. Ferner gehört zu MICROSOFT ROBOTICS STUDIO eine serviceorientierte Laufzeitumgebung *(Microsoft Robotics Studio Runtime)* mit der Anwendungen parallel und verteilt ausgeführt werden können. Insbesondere können Sensordaten verarbeitet und die Motorik von Robotern angesteuert werden. Die Laufzeitumgebung unterstützt inzwischen auch WINDOWS MOBILE 6 und WINDOWS EMBEDDED CE 6.0. Dadurch können auch eingebettete Anwendungen erstellt werden. Bevor die Software auf realen Robotern eingesetzt wird, kann sie zunächst auch nur simuliert werden. MICROSOFT ROBOTICS STUDIO enthält hierfür eine Simulationsumgebung *(Visual Simulation Environment, VSE)* in der die entwickelten Steuerungsprogramme und selbst entworfene Roboter virtuell getestet werden können. Für die 3D–Simulation von Robotern und die Nachbildung physikalischer Effekte wie Gravitation, Reibung und Schattierungen wird die *AGEIA PhysX Technology–Engine* verwendet. Im Dezember 2006 ist die Version 1.0 und im Juli 2007 ist die aktuelle Version 1.5 von MICROSOFT ROBOTICS STUDIO erschienen. Für nicht–kommerzielle Zwecke ist das Herunterladen von MICROSOFT ROBOTICS STUDIO bislang kostenlos.

3.6 Bewertung und Vergleich der Systeme

In diesem Abschnitt werden die Eigenschaften der in den vorherigen Abschnitten vorgestellten Systeme miteinander verglichen und einander gegenüber gestellt. In Tabelle 3.1 sind die wichtigsten Merkmale der Systeme in einer Übersicht dargestellt. Die vorliegende Arbeit unterscheidet sich in einer Reihe von Kriterien von diesen Systemen.

Nutz– / Verfügbarkeit: Die Projekte ROBOTECT, SRAST und ROBOTICA wurden in der Zwischenzeit eingestellt und die Software ist deswegen auch nicht mehr erhältlich. Im Unterschied zu den anderen hier vorgestellten Systemen können ROBOTDRAW und ROBOTRAN nur per Webbrowser benutzt werden. Da eine lokale Installation nicht möglich ist, sind diese Systeme für den Benutzer sehr restriktiv. Von OSCAR und JFKENGINE sind ebenfalls keine Download–Versionen erhältlich, da die Software nur intern verwendet und nicht an Dritte weitergegeben wird. Manche Projekte werden nicht gepflegt. Beispielsweise verwendet MATLAB ROBOTICS TOOLBOX eine inzwischen veraltete MATLAB Version und wurde nicht nach neueren MATLAB Versionen portiert. Für eine Reihe von Projekten werden Lizenzgebühren erhoben, die teilweise sehr hoch sind, und der Quellcode ist nicht verfügbar, zum Beispiel LINKAGE DESIGNER, DYNAFLEXPRO und IGRIP.

Echtzeitanforderungen: Obwohl die Steuerung von Robotern die genaue Einhaltung zeitlicher Vorgaben erfordert und die Berechnungen in wenigen Millisekunden durchgeführt werden müssen, können von diesen Systemen nur OROCOS, MBDYN und MICROB auf einem Echtzeitbetriebssystem eingesetzt werden. Für zeitkritische Steuerungsaufgaben sind Physik–Engines wie zum Beispiel ODE nicht geeignet. Deswegen können reale Roboter mit den meisten dieser Systeme nicht gesteuert werden.

Denavit–Hartenberg Konvention: Viele Projekte verwenden die Denavit–Hartenberg Konvention, um die Geometrie eines Roboters zu beschreiben. Eine Einschränkung bei den meisten Projekten ist dabei, dass kein *Gelenkwinkeloffset* angegeben werden kann. Beispielsweise kann bei OSCAR, MATLAB ROBOTICS TOOLBOX, RTSS, MANDY, MICROB, SYNTHETICA, ROBOTECT, ROBOTDRAW, SRAST, ROBOTICA, JFKENGINE, LINKAGE DESIGNER, SPACELIB, DYNAFLEXPRO, DYNAMECHS, EASY–ROB und ROBOMOSP kein Offsetwert spezifiziert werden, der stets zu einem Gelenkwinkel addiert werden soll. Hingegen kann bei ROBOOP der Benutzer einen Gelenkwinkeloffset angeben. Bei vielen Projekten fehlen ebenso *passive* Gelenkachsen und Transformationen (kein Freiheitsgrad). Ein passive Transformation tritt beispielsweise auf, wenn ein Roboter an der Wand oder an der Decke montiert ist, und deswegen zunächst das Referenzkoordinatensystem, dessen z–Achse üblicherweise anti–parallel zum Gravitätsvektor definiert ist, entsprechend gedreht werden muß. Ein weiteres Beispiel für eine passive Transformation ist die Umrechnung zwischen Flansch– und Werkzeugkoordinatensystem. Bei Austausch des Werkzeugs am Roboterflansch ändert sich üblicherweise die Lage der Werkzeugspitze und damit auch diese Transformation. Aus diesen Gründen können eine Reihe von Industrierobotern nicht mit diesen Projekten modelliert werden und eine allgemeine Anwendbarkeit ist nicht gegeben. In dieser Arbeit wird auch die Denavit–Hartenberg Konvention verwendet. Es werden sowohl passive Transformationen als auch Gelenkwinkeloffsets unterstützt.

Kinematisches Robotermodell: Viele dieser Projekte können anhand von Denavit–Hartenberg Parametern die direkte Kinematik berechnen. Beispielsweise können OSCAR, ROBOOP und MATLAB ROBOTICS TOOLBOX die direkte Kinematik numerisch berechnen, während ROBOTICA die direkte Kinematik sowohl numerisch als auch symbolisch

berechnen kann. Jedoch kann keines dieser Projekte eine geschlossene Lösung der inversen Kinematik für beliebige, nicht kinematisch redundante Manipulatoren bestimmen. Eine näherungsweise Lösung mit Hilfe der *Newton–Raphson Technik* [63] ist bei OSCAR, ROBOOP, MATLAB ROBOTICS TOOLBOX, RTSS und MANDY implementiert. Nachteil dieses Verfahrens ist, dass es nur konvergiert, wenn keine Singularität auftritt und die Start– und Zielkonfiguration genügend dicht beeinander liegen. Deswegen müssen bei Punkt–zu–Punkt Bewegungen Anfangs– und Endpunkt entsprechend dicht gelegt werden. SYNTHETICA verwendet den Levenberg–Marquardt Algorithmus, der allerdings nicht so schnell wie das Newton–Raphson Verfahren konvergiert. Bei manchen Projekten kann der Benutzer das kinematische Robotermodell von Hand bestimmen und über eine Anwendungsprogrammierschnittstelle in das System einbinden, zum Beispiel bei den Projekten MICROB, ROBOTECT, EASY–ROB und ROBOMOSP. Bei der manuellen Herleitung besteht allerdings die Gefahr, dass Lösungen fehlerhaft sind oder übersehen werden. Außerdem ist diese Herleitung sehr aufwändig und kompliziert.

Dynamisches Robotermodell: Eine Reihe von Projekten verwenden Blockschaltbildeditoren, um das dynamische Modell aufzustellen, zum Beispiel DYMOLA und DYNAFLEXPRO. Bei diesen Projekten ist in einer Bibliothek eine große Bandbreite an Funktionsblöcken und Automatisierungskomponenten integriert. Durch das Plazieren und Verschalten von Blöcken kann der Benutzer das System konfigurieren. Blockschaltbilder sind ein gutes Mittel, um Signalflüsse und Zusammenhänge zwischen Aus– und Eingangsgrößen graphisch zu beschreiben. Bei größeren Systemen werden Blockschaltbilder allerdings sehr unübersichtlich. Außerdem lässt sich die mechanische Struktur von Robotern mit Blockschaltbildern nur schwer beschreiben. Für die Erstellung dieser Beschreibung sind Tabellen und Formulare besser geeignet. In mehreren Projekten, die die Dynamik berechnen, sind keine Reibungskräfte vorgesehen, zum Beispiel in OSCAR, DYNAFLEXPRO und NEWEUL.

Trajektoriengenerierung: In manchen Projekten, zum Beispiel in ROBOTICA oder DYNAFLEXPRO, sind keine Trajektoriengenerierungsalgorithmen verfügbar, sondern müssen vom Benutzer manuell hinzugefügt werden. OSCAR, MATLAB ROBOTICS TOOLBOX und RTSS implementieren Algorithmen zur Trajektoriengenerierung im Gelenkwinkelraum auf Basis von Polynomfunktionen, wie sie beispielsweise von Craig [31] beschrieben werden. Nachteilig ist hierbei, dass Interpolationspolynome mit hohem Grad bei vielen Stützpunkten zum Überschwingen neigen. Außerdem muss der Benutzer die Zeitdauer der Bewegung vorgeben. Wird die Zeitdauer als zu hoch gewählt, so werden die Maximalgeschwindigkeiten und –beschleunigungen der Antriebe nicht ausgenutzt. Wird die Zeitdauer als zu klein gewählt, kann die Trajektorie nicht durchgeführt werden. Algorithmen, die zeitoptimale und ruckbegrenzte Trajektorien generieren, sind deswegen vorzuziehen. In manchen Projekten ist das Geschwindigkeitsprofil trapezförmig, zum Beispiel in MANDY und MICROB. Der Nachteil ist hierbei, dass die zeitliche Ableitung der Beschleunigung unendlich groß wird und deswegen der Manipulator sprunghaft beschleunigt und abgebremst wird. Um ruckfreie Bewegungen zu generieren, sollte nicht das Geschwindigkeits–, sondern das Beschleunigungsprofil trapezförmig sein.

Bewegungssteuerung: Viele dieser Projekte konzentrieren sich nur auf Teilaspekte einer Bewegungssteuerung und berücksichtigen die notwendigen Funktionalitäten nicht in ihrer Gesamtheit. Beispielsweise beschränken sich DYNAMECHS und NEWEUL auf die Bestimmung des dynamischen Modells von Mehrkörpersystemen. Ebenfalls fehlt bei den meisten Projekten ein Interpreter für Roboterprogramme.

Tabelle 3.1 Gegenüberstellung der Systeme

		Open–Source–Software	Nutz– / Verfügbarkeit	Echtzeitanforderungen	Denavit–Hartenberg	Newton–Raphson	Lagrange Verfahren	Newton–Euler Verfahren	Roboter Interpreter
1.	OROCOS	X	X	X					
2.	OSCAR				X	X	X	X	
3.	ROBOOP	X	X		X	X		X	
4.	ROBOTICS TOOLBOX	X	X		X	X		X	
5.	RTSS	X	X		X	X		X	
6.	PMT	X	X				X		
7.	MANDY	X	X		X	X		X	X
8.	MICROB	X	X	X	X				
9.	SYNTHETICA	X	X		X	X			
10.	ROBOTECT				X			X	
11.	ROBOTDRAW		X		X				
12.	IKAN	X	X						
13.	SRAST				X				
14.	ROBOTRAN		X					X	
15.	ROBOTICA				X		X		
16.	JFKENGINE				X				
17.	LINKAGE DESIGNER		X		X				
18.	SPACELIB	X	X		X		X		
19.	DYMOLA		X				X		
20.	DYNAFLEXPRO		X		X			X	
21.	NEWEUL		X					X	
22.	MBSYMBA	X	X				X	X	
23.	ADAMS		X				X		
24.	DYNAMO	X	X						
25.	MBDYN	X	X	X				X	
26.	ODE	X	X						
27.	DYNAMECHS	X	X		X				
28.	EASYDYN	X	X						
29.	EASY–ROB		X		X	X			X
30.	EM–WORKPLACE		X						X
31.	ROBOMOSP		X		X			X	X
32.	COSIMIR		X		X				X
33.	IGRIP		X						X
34.	MS ROBOTICS STUDIO		X						

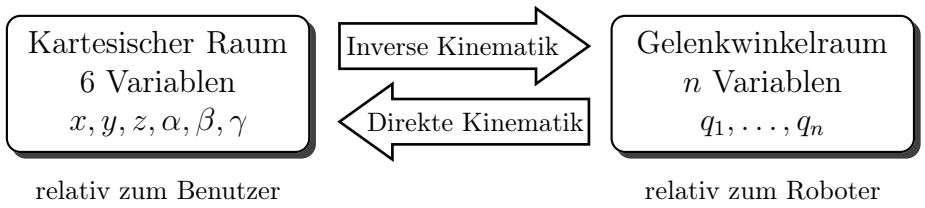
Kapitel 4

Bestimmung des kinematischen Robotermodells

In diesem Kapitel wird beschrieben, wie die Herleitung des kinematischen Modells für eine beliebige serielle Kinematik automatisch durchgeführt wird. Zunächst wird der Manipulator mit Denavit–Hartenberg Parametern beschrieben (Abschnitt 4.1) und das direkte kinematische Problem gelöst (Abschnitt 4.2). Anschließend werden wissensbasierte Methoden (Abschnitt 4.3) und algebraische Eliminationsmethoden (Abschnitt 4.4) angewendet, um das inverse kinematische Problem systematisch zu lösen. Darüber hinaus wird das kinematische Modell auch geometrisch bestimmt (Abschnitt 4.5). Dadurch können die erzeugten mathematischen Modelle sehr gut miteinander verglichen und validiert werden. Außerdem werden durch die Reduktion des Problems auf geometrische Teilprobleme anschaulichere Lösungen generiert als bei der Lösung eines nichtlinearen Gleichungssystems. Schließlich wird in Abschnitt 4.6 die Jacobi–Matrix für allgemeine Kinematiken bestimmt.

Kinematik (griech.: kinema = Bewegung) ist die mathematische Beschreibung von Bewegungsabläufen ohne Berücksichtigung der Kräfte, die die Bewegung verursachen. Dabei gibt es zwei kinematische Teilprobleme. Mit der *direkten Kinematik* beziehungsweise mit der *Vorwärtstransformation* wird anhand eines n–dimensionalen Gelenkvariablenvektors die Position und Orientierung des Endeffektors im Basiskoordinatensystem des Roboters bestimmt. Für das direkte Problem einer seriellen Kinematik gibt es immer eine eindeutige Lösung, die sich in der Regel relativ einfach durch Matrizenmultiplikation bestimmen läßt. Das hierzu inverse Problem ist für serielle Kinematiken im Allgemeinen sehr kompliziert zu bestimmen, da es sich um ein nichtlineares Inversionsproblem handelt (Abbildung 4.1). Für die *inverse Kinematik* beziehungsweise für die *Rückwärtstransformation* kann es keine Lösung geben, zum Beispiel liegt die Zielpose außerhalb des Arbeitsraums, eine Lösung, zum Beispiel liegt die Zielpose am Rande des Arbeitsraums, endlich viele Lösungen oder unendlich viele Lösungen. Die genaue Anzahl von Lösungen hängt sowohl von Hindernissen, der Manipulatorgeometrie als auch von der gegebenen Pose des Endeffektors ab.

Abbildung 4.1 Direkte und inverse Kinematik



Tsai und Morgan [169, 180] waren die ersten Autoren, die annahmen, dass die inverse Kinematik eines allgemeinen, seriellen Roboters mit sechs Freiheitsgraden im kartesischen Raum bis zu 16 Lösungen haben kann. Sie konvertieren zunächst die kinematischen Gleichungen in Polynomialgleichungen und transformierten dann diese Gleichungen in acht quadratische Gleichungen mit acht Unbekannten. Die Nullstellen dieser Gleichungen bestimmten sie mit der sogenannten Fortsetzungsmethode [1] numerisch. Motiviert durch dieses Ergebnis hat Primrose [144] wissenschaftlich bewiesen, dass die obere Anzahl von geschlossenen Lösungen tatsächlich 16 ist. Außerdem zeigte er, dass es neben 16 realen auch 16 imaginäre Lösungen geben kann. Lee und Liang [87] zeigten, dass das Problem der inversen Kinematik eines sechsachsigen Roboters in ein univariates Polynom vom Grade 16 transformiert werden kann. Die Nullstellen dieses Polynoms können nur numerisch bestimmt werden. Raghaven und Roth [147] haben das Problem ebenfalls in eine Menge von Polynomgleichungen umgewandelt. Eine weitere Lösung stammt von Manocha and Canny [94, 95, 96]. Sie transformierten das Problem in ein Problem zur Bestimmung der Eigenwerte und Eigenvektoren einer Matrix. Manseur und Doty [100] präsentierten den ersten Manipulator der tatsächlich die maximale Anzahl von Lösungen erreicht. Dadurch haben sie den Beweis, dass 16 die maximale Anzahl von Lösungen für Roboterarme mit sechs Freiheitsgraden ist, abgeschlossen. Im Bereich der Parallelkinematiken zeigte Raghavan [146] durch Anwendung der Fortsetzungsmethode, dass es 40 Lösungen für die direkte Kinematik einer allgemeinen Stewart–Gough–Plattform gibt. Husty [77] kam zu demselben Ergebnis, indem er eine univariate Polynomgleichung vom Grade 40 ermittelte.

Die inverse Kinematik kann numerisch oder analytisch bestimmt werden. Bei *numerischen Verfahren* treten Probleme auf. Sie sind sehr rechenintensiv und ermitteln immer nur eine Konfiguration gegebenenfalls unter Einhaltung von Nebenbedingungen. Die Anzahl der erforderlichen Iterationen und damit die Anzahl der nötigen arithmetischen Operationen kann stark variieren und nicht im Voraus bestimmt werden. Numerische Verfahren bestimmen häufig nur die zum gewählten Startpunkt nächstliegende Lösung und das Auffinden einer Lösung ist oft nicht garantiert. Numerische Verfahren sind sehr anfällig für Singularitäten, das heißt dass sie in bestimmten Fällen numerisch instabil sind und oszillieren, obwohl es unendlich viele Lösungen geben kann. Bei einer numerischen Lösung ist es möglich, dass bei der Ausführung eines Roboterprogramms in einer Simulationsumgebung es zu keiner Kollision kommt, und bei der Ausführung desselben Programms auf einer Robotersteuerung es eine Kollision gibt. Die Ursache hierfür ist, dass jeweils andere Gelenkkonfigurationen auftreten. Numerische Methoden wurden von Kelmar und Khosla [86], Höpler und Otter [74, 75] sowie in einigen Projekten (Abschnitt 3.1) implementiert.

Analytische Verfahren zur Bestimmung der Rückwärtstransformation werden angestrebt, da sie exakte Ergebnisse liefern und ihr Berechnungsaufwand niederer als der von numerischen Lösungen ist. Da im Voraus die erforderliche Anzahl arithmetischer Operationen bestimmt werden kann und diese Anzahl sich nicht ändert, sind die Rechenzeiten für geschlossene Lösungen konstant. Geschlossene Lösungen sind daher für Echtzeitsysteme besser geeignet. Zudem liefern sie alle Konfigurationen, das heißt alle möglichen Gelenkstellungen bei gleicher Pose des Endeffektors. Es können detaillierte Aussagen über das Gelenkverhalten getroffen und die Bahnplanung optimiert werden. Dabei ist zu beachten, dass eine ausgewählte Konfiguration nur an geeigneten Stellen gewechselt werden kann. Zur Erkennung von Kollisionen sind analytische Lösungen ebenfalls günstiger. Bei analytischen Lösungen kann es im Gegensatz zu numerischen Lösungen keine Approximierungsfehler geben. Rundungsfehler können nur bei einer numerischen Ausführung auftreten.

Analytische Verfahren werden in geometrische und algebraische Verfahren eingeteilt. *Geometrische Verfahren* basieren auf geometrischen Überlegungen und trigonometrischen Beziehungen, beispielsweise der Anwendung von Kosinussatz und Sinussatz. Sie erfordern allerdings eine hohe Intuition und Erfahrung von Seiten des Entwicklers und bei der Herleitung sind viele geometrische Fallunterscheidungen nötig. Diese graphischen Verfahren werden aufwändig von Hand durchgeführt und sind nur für spezielle Robotertypen geeignet. Bei *algebraischen Verfahren* muss ein hochgradig nichtlineares Gleichungssystem mit bis zu sechs unbekannten Gelenkvariablen gelöst werden. Dadurch erhält man wie beim geometrischen Verfahren eine geschlossene Lösung und auch etwaige Mehrfachlösungen. Die Schwierigkeit beim algebraischen Verfahren besteht in der Auflösung der Gleichungen nach einer oder zwei der unbekannten Gelenkvariablen. Allerdings müssen diese *transzendenten* Gleichungen nur einmal und offline für einen gegebenen Robotertyp gelöst werden.

Bei einer seriellen Kinematik mit sechs Bewegungsachsen gibt es in der Regel unterschiedlich viele Gelenkstellungen bei gleicher Lage des Endeffektors (Tabelle 4.1). Bei einem Gelenkarmroboter mit Zentralhand kann es bis zu acht unterschiedliche Gelenkkonfigurationen für eine bestimmte Endeffektorlage geben. Bei einem kinematisch redundanten Roboter gibt es keine geschlossene Lösung für die inverse Kinematik, da das Positions– und Orientierungsproblem auf unendlich verschiedene Weisen lösbar ist. Kinematisch redundant sind zum Beispiel Roboter mit zwei parallelen Translationsachsen oder mit drei Translationsachsen, die in einer Ebene liegen oder Roboter mit vier Translationsachsen.

Tabelle 4.1 Analytische Lösbarkeit der inversen Kinematik eines seriellen Manipulators

Robotertyp	obere Anzahl an Lösungen
kinematisch redundant	∞
6R, 5R1T	16
4R2T, 6R mit Zentralhand	8
3R3T	2

Um die Entwicklung von Steuerungssoftware zu vereinfachen, werden die Gelenkachsen heutiger Industrieroboter meistens parallel oder rechtwinklig und nicht windschief zueinander angeordnet. Manseur and Doty [101, 102] zeigten, dass Roboter mit fünf Rotationsachsen geschlossen gelöst werden können, sofern mindestens zwei aufeinanderfolgende Achsen sich schneiden oder parallel zueinander sind. Ein allgemeiner sechsachsiger Roboter kann nicht geschlossen gelöst werden, allerdings können bestimmte Geometrien analytisch gelöst werden. Pieper [140] bestimmte in seiner Dissertation solche günstige Robotergeometrien bei denen die Gelenkachsen voneinander entkoppelt sind. Mathematisch betrachtet lassen sich dadurch unbekannte Gelenkvariablen gut isolieren. Hierzu gehören beispielsweise Roboter bei denen drei aufeinander folgende Achsen sich in einem Punkt schneiden, wie zum Beispiel bei Robotern mit Zentralhand oder Roboter bei denen drei aufeinander folgende Achsen parallel sind, wie zum Beispiel bei SCARA–Robotern. Die meisten Industrieroboter werden heute so entworfen, dass sie mindestens eines der beiden Kriterien erfüllen. Tourassis und Ang [168] bestimmten weitere Robotergeometrien, bei denen die Gelenkachsen entkoppelt sind und somit eine geschlossene Lösung der inversen Kinematik ermittelt werden kann. Hollerbach [73] zeigte, wie die Gelenkachsen von redundanten Robotern mit sieben Gelenkachsen optimal angeordnet werden können.

4.1 Modellierung offener, kinematischer Ketten

Um einen Roboter kinematisch zu beschreiben, wird die Denavit–Hartenberg Konvention (DH) [34, 35], die erstmals 1955 vorgeschlagen wurde, verwendet. Die DH–Konvention ist die kompakteste Darstellung der Geometrie eines Manipulators, da lediglich vier Parameter pro Gelenkachse notwendig sind. Hierbei wird nur die räumliche Lage der Gelenkachsen zueinander betrachtet. Von der konkreten Form der Gelenkglieder wird abstrahiert.

Zunächst beschreibt der Benutzer mit dem graphischen Konfigurationswerkzeug (Abschnitt 6.2), das im Rahmen dieser Arbeit entwickelt wurde, die mechanische Struktur eines Roboters. Durch das Ausfüllen der dafür vorgesehenen Felder kann er beispielsweise die Anzahl und Typen der Gelenkachsen sowie deren Anordnung festlegen. In jede Gelenkachse wird anhand dieser Beschreibung automatisch ein körperfestes, orthonormales Koordinatensystem nach bestimmten Regeln gelegt. Der Manipulator wird hierzu in der Nullstellung aller Gelenke betrachtet. Eine gewisse Wahlfreiheit bleibt bei der Zuordnung der Koordinatensysteme zu den Gelenkachsen dennoch erhalten. In die Basis des Manipulators wird ein ortsfestes Ausgangskoordinatensystem gelegt. Die Gelenkachsen werden von 1 bis n durchnummeriert. In einem iterativen Prozess wird das Koordinatensystem KS_i mit Hilfe von KS_{i-1} bestimmt. Dabei wird unterschieden, ob die Gelenkachsen parallel oder rechtwinklig zueinander angeordnet sind und sich schneiden beziehungsweise nicht schneiden. Die z_i–Achse wird in die Gelenkachse $i+1$, welches Glied i und $i+1$ verbindet, gelegt. Für ein Translationsgelenk ist die Richtung der z_i–Achse in Richtung der Bewegung weg vom Gelenk definiert. Für ein Rotationsgelenk ist die Richtung der z_i–Achse durch die positive Drehrichtung um die Drehachse definiert.

- Festlegung der x_i–Achse bei sich schneidenden Gelenkachsen:
  Wenn sich die Achsen schneiden, existiert keine eindeutige gemeinsame Normale, und die x_i–Achse wird parallel oder anti–parallel zum Kreuzprodukt der z–Achsen des letzten und des aktuellen Gliedes gelegt. In vielen Fällen zeigt dann die x_i–Achse in die gleiche Richtung wie die x_{i-1}–Achse des vorhergehenden Gliedes. Der Ursprung von KS_i wird in den Schnittpunkt der Achsen z_i und z_{i-1} gelegt.

- Festlegung der x_i–Achse bei sich nicht schneidenden, orthogonalen Gelenkachsen:
  Wenn sich die Gelenkachsen z_i und z_{i-1} nicht schneiden und auch nicht parallel verlaufen, wird die gemeinsame Normale (kürzeste Verbindungsstrecke) zwischen diesen beiden Gelenkachsen gesucht. Die x_i–Achse wird in Richtung der gemeinsamen Normalen und von KS_{i-1} wegweisend gelegt. Der Ursprung von KS_i wird in den Schnittpunkt der Normalen mit der Achse z_i gelegt.

- Festlegung der x_i–Achse bei echt parallelen Gelenkachsen:
  Für diesen Fall ist die Richtung der x_i–Achse eindeutig definiert. Die x_i–Achse wird entlang der gemeinsamen Normalen von z_i und z_{i-1}, und von z_{i-1} wegweisend gelegt. Oft ist dies die Richtung der mechanischen Verbindung des Armteiles zwischen den Gelenkachsen i und $i+1$. Das Gemeinlot ist nicht eindeutig bestimmt und der Ursprung von KS_i auf der z_i–Achse ist frei wählbar.

Durch diese Festlegungen steht die x_i–Achse senkrecht zur z_{i-1}–Achse und schneidet die z_{i-1}–Achse. Unter Umständen wird auch eine Transformation vom Flansch– zum Endeffektorkoordinatensystem benötigt. Der Ursprung von KS_n wird in den *Tool Center Point (TCP)* gelegt. Die y_i–Achse wird so definiert, dass sich ein rechtshändiges kartesisches

Koordinatensystem ergibt. Sie wird durch das Kreuzprodukt ermittelt. Für den Fall, dass die Achsen z_i und z_{i-1} „zusammen fallen“, das heißt die beiden Achsen haben unendlich viele Punkte gemeinsam, ist der Manipulator kinematisch redundant. Da diese Achsanordnung nicht sinnvoll ist, wird dieser Fall von dem Konfigurationswerkzeug abgefangen. Für den Spezialfall, dass die Achsen z_i und z_{i-1} windschief zueinander liegen, sind die Koordinatensysteme von Hand zu bestimmen. Allerdings kommt dieser Fall eher selten vor. Im nächsten Schritt werden anhand dieser Zuordnung Relativ–Transformationen zwischen benachbarten Koordinatensystemen bestimmt. Hierfür wird jedes Gelenk mit einem Satz von vier Parametern beschrieben $(\theta_i, d_i, a_i, \alpha_i)$, der eindeutig die Lage des i–ten Koordinatensystems bezüglich des $(i-1)$–ten angibt. Zwei Parameter beziehen sich auf Rotationen und zwei auf Translationen.

- Der *Gelenkwinkel* θ_i (engl.: joint angle) ist der Winkel, um den man die erste Achse x_{i-1} um die z_{i-1}–Achse drehen muss, damit sie parallel zur zweiten Achse x_i wird. Manchmal wird ein konstruktionsbedingter konstanter Wert (Gelenkwinkeloffset) hinzuaddiert, um in der Nullstellung die Achse x_{i-1} in die Achse x_i zu überführen.

- Die *Hebelverschiebung* d_i (engl.: link offset) ist der kürzeste Abstand der Achsen x_{i-1} und x_i entlang der z_{i-1}–Achse.

- Die *Hebellänge* a_i (engl.: link length) ist die gemeinsame Normale der beiden Achsen z_{i-1} und z_i, also der kürzeste Abstand der Achsen z_{i-1} und z_i entlang der x_i–Achse.

- Die *Hebeldrehung* beziehungsweise *Armelementverwindung* α_i (engl.: link twist) ist der Winkel, um den man die erste Achse z_{i-1} um die x_i–Achse drehen muss, damit sie parallel zur zweiten Achse z_i wird. Dies ergibt also den Neigungswinkel der beiden Achsen am Glied i.

Eine Translation entlang beziehungsweise Rotation um die y_{i-1}– oder y_i–Achse wird nicht ausgeführt. Um die DH–Parameter automatisch zu bestimmen, wurde folgendes Expertenwissen in Regelform gebracht.

- Bei einem aktiven Gelenk ist ein DH–Parameter variabel, während die anderen konstant sind.

- Bei einem passiven Gelenk sind alle DH–Parameter konstant.

- Ist das Gelenk i ein Rotationsgelenk, so ist der Drehwinkel θ_i die veränderliche Gelenkvariable.

- Ist das Gelenk i ein Translationsgelenk, so ist der Translationsweg d_i die veränderliche Gelenkvariable.

- Die DH–Parameter a_i und α_i sind immer konstante Konstruktionsparameter.

- Ist das Gelenk i nicht rotatorisch und die Achsen x_{i-1} und x_i parallel zueinander, so ist $\theta_i = 0°$ (beziehungsweise $\theta_i = 180°$, wenn die beiden Achsen anti–parallel zueinander sind).

- Ist das Gelenk i nicht translatorisch und die Achsen x_{i-1} und x_i schneiden sich, so ist $d_i = 0$.

- Schneiden sich die Achsen z_{i-1} und z_i, so ist $a_i = 0$.

- Sind die Achsen z_{i-1} und z_i parallel zueinander, so ist $\alpha_i = 0°$ (beziehungsweise $\alpha_i = 180°$, wenn die beiden Achsen anti–parallel zueinander sind).

Nachdem die DH–Parameter anhand der Lage der lokalen Koordinatensysteme und mit Hilfe dieser Regeln bestimmt wurden, werden sie in eine Tabelle eingetragen. Als nächstes wird eine Menge von Transformationsmatrizen bestimmt. Die Transformation von Koordinatensystem KS_{i-1} zu KS_i wird als homogene 4×4 Transformationsmatrix dargestellt, bestehend aus einer 3×3 Rotationsmatrix und einem 3×1 Verschiebungsvektor [106]. Die i–te Transformationsmatrix überführt mit vier nacheinander ausgeführten elementaren Transformationen das Koordinatensystem KS_{i-1} in das nachfolgende Koordinatensystem KS_i. Zuerst wird dabei mit dem Winkel θ_i um die z_{i-1}–Achse gedreht und danach entlang der z_{i-1}–Achse um den Wert d_i verschoben. Anschließend wird entlang der x_i–Achse um den Wert a_i verschoben und schließlich mit dem Winkel α_i um die x_i–Achse gedreht.

$$T_{i-1,i} = Rot(z_{i-1}, \theta_i) * Trans(z_{i-1}, d_i) * Trans(x_i, a_i) * Rot(x_i, \alpha_i) \tag{4.1a}$$

$$= \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.1b}$$

$$= \begin{pmatrix} \cos\theta_i & -\cos\alpha_i \cdot \sin\theta_i & \sin\alpha_i \cdot \sin\theta_i & a_i \cdot \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cdot \cos\theta_i & -\sin\alpha_i \cdot \cos\theta_i & a_i \cdot \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.1c}$$

Diese Transformationsmatrix ist vergleichsweise einfach aufgebaut und ist immer invertierbar. Die entsprechende inverse Transformation wird durch folgende Matrix bestimmt.

$$T_{i,i-1} = \begin{pmatrix} \cos\theta_i & \sin\theta_i & 0 & -a_i \\ -\cos\alpha_i \cdot \sin\theta_i & \cos\alpha_i \cdot \cos\theta_i & \sin\alpha_i & -d_i \cdot \sin\alpha_i \\ \sin\alpha_i \cdot \sin\theta_i & -\sin\alpha_i \cdot \cos\theta_i & \cos\alpha_i & -d_i \cdot \cos\alpha_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

4.2 Bestimmung der direkten Kinematik

Mit der Vorwärtstransformation wird die Position und Orientierung des Endeffektors im Basiskoordinatensystem des Roboters in Abhängigkeit der Gelenkvariablen bestimmt. Für das direkte kinematische Problem gibt es bei seriellen Kinematiken immer eine geschlossene und eindeutige Lösung. Die Gesamttransformation wird durch Multiplikation der homogenen 4×4–Transformationsmatrizen berechnet [155]. Diese Matrixmultiplikationen können sowohl symbolisch als auch numerisch durchgeführt werden. In dieser Arbeit wird die resultierende Matrix symbolisch berechnet und die Terme weitestgehend vereinfacht, um eine geschlossene und kompakte Darstellung zu erhalten. Die Lage des Endeffektors wird bei gegebenem Gelenkvariablenvektor $\vec{q}$ durch Gleichung (4.3) bestimmt.

$$T_{0,n}(\vec{q}) = \prod_{i=1}^{n} T_{i-1,i}(q_i) = \begin{pmatrix} \vec{n} & \vec{o} & \vec{a} & \vec{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

Dabei wird die Position des Endeffektors durch den Vektor $\vec{p}$ und die Orientierung des Endeffektors durch die drei orthogonalen Einheitsvektoren $\vec{n}, \vec{o}$ und $\vec{a}$ angegeben.

4.3 Regelbasierte Lösung der inversen Kinematik

Die Rückwärtstransformation berechnet anhand einer vorgegebenen kartesischen Position und Orientierung des Endeffektors die entsprechenden Gelenkkoordinaten zur Ansteuerung der Antriebe eines Roboters. Diese Funktion nimmt eine zentrale Stellung in der Realisierung der Bewegungssteuerung ein. Die Lösung dieses Problems ist bei seriellen Kinematiken sehr viel schwieriger als die Lösung des direkten kinematischen Problems. Bei kinematisch redundanten Robotern gibt es keine geschlossene Lösung.

Zur Bestimmung der inversen Kinematik müssen die Gleichungen der direkten Kinematik nach den unbekannten Gelenkkoordinaten aufgelöst werden. Diese transzendenten Gleichungen sind jedoch untereinander gekoppelt und können aufgrund ihrer starken Nichtlinearität nicht direkt gelöst werden. Deswegen besteht das weitere Vorgehen darin, die Gleichungen der Vorwärtstransformation in vielen verschiedenen Möglichkeiten anzuordnen, um einfacher zu lösende Gleichungen zu erhalten und unbekannte Gelenkvariablen zu isolieren. Obwohl es sich um zwölf Gleichungen mit bis zu sechs unbekannten Gelenkvariablen handelt, können diese Gleichungen auf sehr viele unterschiedliche Arten dargestellt werden. Dieses Vorgehen wurde erstmals von Paul, Renaud und Stevenson vorgeschlagen [134]. Die einzelnen Transformationsmatrizen werden invertiert ($T_{i-1,i}^{-1} = T_{i,i-1}$) und von rechts und links an die Gleichungen der Vorwärtstransformation multipliziert. Für einen Manipulator mit sechs Freiheitsgraden und den Transformationsmatrizen $T_{01}, T_{12}, T_{23}, T_{34}, T_{45}$ und T_{56} sowie der homogenen Matrix R, die die Endeffektorlage in kartesischen Koordinaten angibt, wird automatisch das Gleichungssystem in Abbildung 4.2 aufgestellt und dann regelbasiert gelöst. Hierbei handelt es sich natürlich um redundante Transformationen der Gleichungen der direkten Kinematik.

Die linke Seite jeder dieser Matrixgleichungen hängt von den unbekannten Gelenkvariablen $q_i, \ldots, q_j$ ab. Die rechte Seite hängt von den verbleibenden Gelenkvariablen $q_1, \ldots, q_{i-1}$ und $q_{j+1}, \ldots, q_n$ sowie von der Position und Orientierung des Endeffektors ab. Jede dieser Matrixgleichungen liefert zwölf nichtlineare transzendente Gleichungen, die von den oberen drei Zeilen der 4×4 Matrizen stammen. Die vierte Matrizenzeile ist immer trivial und somit vernachlässigbar. Während drei dieser Gleichungen von den Positionstermen stammen, stammen die restlichen neun von den Orientierungstermen. Insgesamt werden bei n Transformationsmatrizen $12 \cdot n \cdot (n+1)/2$ Gleichungen generiert. Bei einem sechsachsigen Roboter entsteht auf diese Weise ein Gleichungssystem mit insgesamt 252 Gleichungen.

Dieses Vorgehen reduziert die Anzahl unbekannter Variablen in mehreren Gleichungen und damit auch die symbolische Komplexität der resultierenden Gleichungen. In einigen Gleichungen taucht bei den meisten Industrierobotern nur noch eine unbekannte Variable auf. Dennoch sind diese Gleichungen in der Regel kompliziert zu lösen, da unbekannte Gelenkwinkel in trigonometrischen Funktionen vorkommen. Das Hauptproblem ist nun günstige Gleichungen auszuwählen, die eine Lösung nach einer unbekannten Gelenkvariablen zulassen. Das hierfür notwendige Expertenwissen über analytische Lösungen von transzendenten Gleichungen wird in Form von deklarativen Regeln ausgedrückt und in einer Wissensbasis in Form von Wenn/Dann–Regeln gespeichert. Jede Produktionsregel ist einheitlich aufgebaut. Der Bedingungsteil ist ein Muster (engl.: Pattern), bestehend aus einer oder mehreren transzendenten Gleichungen. Der Aktionsteil ist eine Operation, die ausgeführt wird, wenn die Regel feuert. Die Operation besteht darin, eine Lösung mit einer unbekannten Gelenkvariablen zu assoziieren und auf Basis dieser Lösung die kinematischen Gleichungen zu vereinfachen. Anders als die If/Then–Anweisungen in

Abbildung 4.2 Automatische Generierung des kinematischen Gleichungssystems ($n = 6$)

$$
\begin{array}{rcl}
T_{01} * T_{12} * T_{23} * T_{34} * T_{45} * T_{56} = & & R \\
T_{01} * T_{12} * T_{23} * T_{34} * T_{45} = & & R * T_{65} \\
T_{01} * T_{12} * T_{23} * T_{34} = & & R * T_{65} * T_{54} \\
T_{01} * T_{12} * T_{23} = & & R * T_{65} * T_{54} * T_{43} \\
T_{01} * T_{12} = & & R * T_{65} * T_{54} * T_{43} * T_{32} \\
T_{01} = & & R * T_{65} * T_{54} * T_{43} * T_{32} * T_{21} \\
T_{12} * T_{23} * T_{34} * T_{45} * T_{56} = & T_{10} * R & \\
T_{12} * T_{23} * T_{34} * T_{45} = & T_{10} * R * T_{65} & \\
T_{12} * T_{23} * T_{34} = & T_{10} * R * T_{65} * T_{54} & \\
T_{12} * T_{23} = & T_{10} * R * T_{65} * T_{54} * T_{43} & \\
T_{12} = & T_{10} * R * T_{65} * T_{54} * T_{43} * T_{32} & \\
T_{23} * T_{34} * T_{45} * T_{56} = & T_{21} * T_{10} * R & \\
T_{23} * T_{34} * T_{45} = & T_{21} * T_{10} * R * T_{65} & \\
T_{23} * T_{34} = & T_{21} * T_{10} * R * T_{65} * T_{54} & \\
T_{23} = & T_{21} * T_{10} * R * T_{65} * T_{54} * T_{43} & \\
T_{34} * T_{45} * T_{56} = & T_{32} * T_{21} * T_{10} * R & \\
T_{34} * T_{45} = & T_{32} * T_{21} * T_{10} * R * T_{65} & \\
T_{34} = & T_{32} * T_{21} * T_{10} * R * T_{65} * T_{54} & \\
T_{45} * T_{56} = & T_{43} * T_{32} * T_{21} * T_{10} * R & \\
T_{45} = & T_{43} * T_{32} * T_{21} * T_{10} * R * T_{65} & \\
T_{56} = & T_{54} * T_{43} * T_{32} * T_{21} * T_{10} * R &
\end{array}
$$

konventionellen Programmiersprachen haben Regeln keinen Else–Zweig und dürfen nicht ineinander geschachtelt werden. Außerdem handelt es sich bei der Bedingung um ein Pattern und nicht um einen booleschen Ausdruck. Dieses regelbasierte System zur Lösung der kinematischen Gleichungen ist modular aufgebaut. Die einzelnen Komponenten sind:

Wissensbasis: Spezielles Wissen über die Lösung transzendenter Gleichungen in Form einer Sammlung von Produktionsregeln. Jede Regel hat einen Bedingungsteil (engl.: left hand side) und einen Aktionsteil (engl.: right hand side) und ist unabhängig vom aktuell betrachteten Robotertyp gültig. Die Wissensbasis kann leicht um zusätzliche Regeln erweitert werden, um mehr Gleichungstypen lösen zu können.

Arbeitsspeicher: Globale Datenbasis, die die Fakten enthält. Die Fakten repräsentieren in symbolischer Form die kinematischen Gleichungen des momentan betrachteten Roboters. Fakten und Regeln sind daher voneinander getrennt. Durch Anwendung des Aktionsteils von Regeln wird der Inhalt des Arbeitsspeichers geändert.

Regelinterpreter / Inferenzmaschine: Komponente, die die in der Wissensbasis gespeicherten Regeln identifiziert, deren Bedingungsteil hinsichtlich der Gleichungen im Arbeitsspeicher erfüllt ist. Ist der Bedingungsteil einer Regel vollständig erfüllt, so wird eine sogenannte *Instantiierung* in die Konfliktmenge aufgenommen. Eine Instantiierung ist die Kombination einer Regel mit den Fakten, die sie erfüllt. Diese Inferenzmethode, die ausgehend von Fakten durch Symbolverarbeitung entsprechende Schlußfolgerungen zieht, wird auch als *Vorwärtsverketttung* (engl.: forward chaining) beziehungsweise *datengetriebene Inferenz* (engl.: data driven inference) bezeichnet.

Konfliktmenge / Agenda: Ungeordnete Menge von Regelinstantiierungen, die vom Regelinterpreter erzeugt wird. Insbesondere enthält die Konfliktmenge alle die Regeln, die potentiell angewendet werden können. Die Agenda kann auch leer sein.

Konfliktlösungsstrategie: Komponente, die eine Instantiierung aus der Konfliktmenge auswählt. Für diesen Zweck sind unterschiedliche Strategien denkbar und ergeben sich aus der Anwendung. Beispielsweise werden Instantiierungen in der Reihenfolge ihrer Aufnahme in die Agenda, nach der Spezifität der Bedingungsteile der instantiierten Regeln, nach dem Zufallsprinzip oder durch übergeordnetes Metawissen ausgewählt. In diesem System wird die nächste anzuwendende Regel auf der Grundlage von Prioritäten (engl.: salience) bestimmt. Der Aktionsteil der anwendbaren Regel mit der höchsten Priorität wird vor den Aktionsteilen von Regeln mit niederer Priorität ausgeführt. Das Anwenden einer Regel wird auch als *Feuern* bezeichnet.

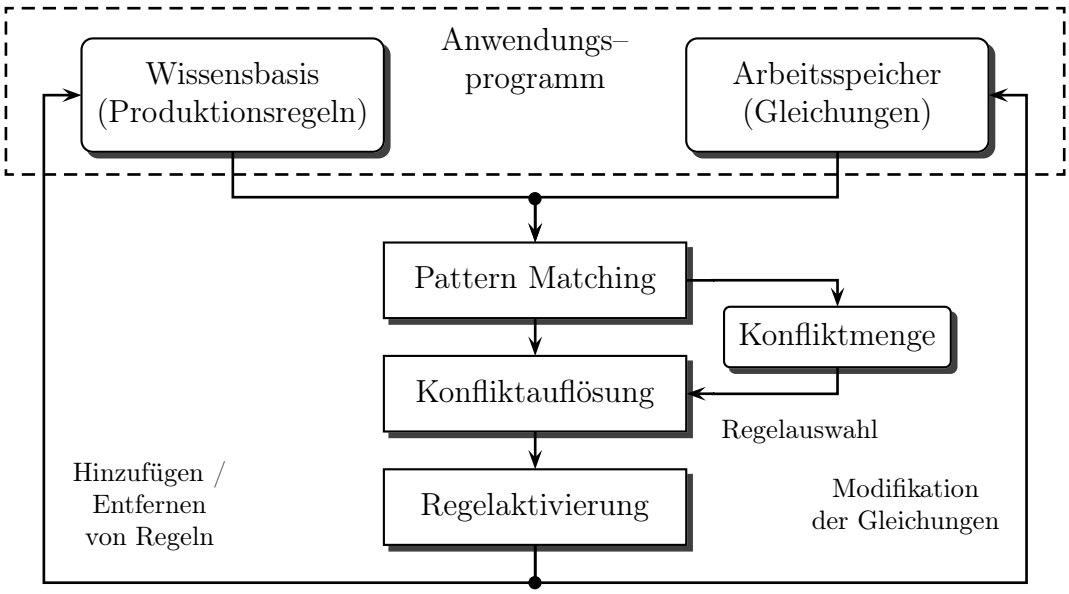
Erklärungskomponente: Komponente über die der Anwender bei Bedarf Erklärungen über das Zustandekommen der Lösungen erhält. Indem die angewendeten Regeln und die Reihenfolge ihrer Anwendung angezeigt und als LaTeX–Dokumentation ausgegeben werden, kann der Benutzer nachverfolgen, wie eine Lösung bestimmt wurde. Die Arbeitsweise des Systems wird hierdurch dem Benutzer transparent gemacht.

Benutzerschnittstelle: Steuerung des Dialogs mit dem Benutzer. Der Benutzer liefert als Eingabe eine Beschreibung der mechanischen Struktur eines Roboters. Die Dialogkomponente ist zweckmäßigerweise mit der Erklärungskomponente gekoppelt.

Damit ergibt sich der folgende Ausführungszyklus zur Lösung der kinematischen Gleichungen (Abbildung 4.3). Zunächst werden die Gleichungen im Arbeitsspeicher in eine kanonische Darstellung transformiert. Es werden diejenigen Regeln gesammelt, deren Prämisse durch Gleichungen im Arbeitsspeicher vollständig erfüllt ist. Als nächstes werden Regelinstantiierungen durch die Kombination einer Regel mit den Gleichungen, die sie erfüllt, gebildet und in die Konfliktmenge getan. Dieser Vorgang wird auch als *Pattern Matching* bezeichnet. Sofern sich mehr als eine Instantiierung in der Konfliktmenge befindet, so wird mit Hilfe von Konfliktlösungsstrategien die nächste anzuwendende Regel ausgewählt. In jedem Durchlauf wird dabei genau eine aktivierte Regel zur Ausführung gebracht. Nach der Ausführung einer Regel wird die entsprechende Regelinstantiierung aus der Konfliktmenge gelöscht, um eine erneute Ausführung zu verhindern. Mit Hilfe der gefundenen Lösung werden die Gleichungen im Arbeitsspeicher vereinfacht. In dem vereinfachten Gleichungssystem wird wiederum nach Gleichungen gesucht, die nach einer unbekannten Gelenkvariablen auflösbar sind. Diese Schritte werden solange wiederholt bis für alle Gelenkvariablen eine Lösung ermittelt werden konnte oder keine Regel mehr anwendbar ist. Dieser Algorithmus terminiert immer für alle seriellen Robotertypen.

Jeder Regel ist eine *Priorität* zugewisen, um Konflikte zwischen Regeln aufzulösen. Regeln, die eine eindeutige Lösung liefern, ist eine höhere Priorität assoziiert, als Regeln die zwei Lösungen liefern. Ein weiterer Grund für die Verwendung von Prioritäten ist, dass gewisse Regeln Lösungen produzieren, die in Echtzeit effizienter ausgewertet werden können als die von anderen Regeln. Können zwei Regeln mit gleicher Priorität feuern, so erfolgt die Konfliktauflösung nach größter *Aktualität*, das heißt dass die zuletzt instantiierte Regel höher eingestuft wird als alle vorherig instantiierten Regeln derselben Priorität. Diese Konfliktlösungsstrategie wird auch als *Depth* bezeichnet. Bei der Strategie *Breadth* werden die als erstes instantiierten Regeln höher eingestuft als die später instantiierten Regeln mit der gleichen Priorität. Der Ablauf der Regelauswertung ist somit deterministisch.

Abbildung 4.3 Regelbasierte Architektur zur Lösung der kinematischen Gleichungen



4.3.1 Entwicklung einer Wissensbasis für trigonometrische Gleichungen

In dieser Arbeit wurde eine Wissensbasis entwickelt, die die geschlossenen Lösungen von häufig vorkommenden kinematischen Gleichungen enthält. Mit Hilfe dieser Wissensbasis werden die mathematischen Lösungen von den einzelnen Gelenkvariablen automatisch gewonnen. Paul [133] und Wolovich [191] haben die Lösungen einer Reihe von trigonometrischen Gleichungen theoretisch bestimmt. Diese und weitere Lösungen wurden in Form von Lösungsregeln in die Wissensbasis aufgenommen. Insgesamt enthält die Wissensbasis zwölf Lösungsgleichungen, von denen elf sich auf trigonometrische Gleichungen beziehen. Bei der zwölften Regel handelt es sich um die Lösung von Polynomen ersten Grades.

Unbekannte Gelenkwinkel dürfen in den Produktionsregeln nicht über den Arcussinus oder Arcuscosinus berechnet werden, da der Arcussinus nur Werte im Intervall $[-\pi/2, \pi/2]$ und der Arcuscosinus nur Werte im Intervall $[0, \pi]$ liefert. Außerdem ist der Definitionsbereich dieser beiden Arcusfunktionen auf das Intervall $[-1, 1]$ beschränkt. Der Arcustangens liefert nur Werte im Intervall $(-\pi/2, \pi/2)$, da die Tangensfunktion für Funktionswerte von $\pm\pi/2$ nicht umkehrbar ist. Um Lösungen nicht zu verlieren und beliebige Orientierungen des Endeffektors ausdrücken zu können, werden unbekannte Gelenkwinkel in den Lösungsregeln über den *erweiterten Arcustangens* berechnet. Diese Funktion hat zwei Argumente, wobei das zweite Argument eigentlich der Nenner ist, welcher aber Null sein darf. Durch Auswertung der Vorzeichen der beiden Argumente dieser Funktion wird der Quadrant des Ergebnisses bestimmt. Der erweiterte Arcustangens liefert so Werte im Intervall $(-\pi, \pi]$. Falls beide Argumente der Arcustangens Funktion Null sind, so ist die Funktion mathematisch nicht definiert und diese Konfiguration für den Manipulator nicht zulässig.

Bei der Rückwärtstransformation kann es auch zwei Lösungen für einen unbekannten Gelenkwinkel geben. Diese Mehrfachlösungen liegen oft symmetrisch, das heißt ein Gelenk kann spiegelbildlich nach links oder nach rechts einknicken, um dieselbe Pose zu erreichen. Der $\pm$–Operator in einigen Lösungen spiegelt diese Mehrdeutigkeit wieder. Falls bei der

Berechnung der inversen Kinematik ein Radikand, das heißt der Wert unter einer Quadratwurzel, negativ wird oder eine Division durch Null stattfindet, reduziert sich die Anzahl der möglichen Gelenkkonfigurationen für diese kartesische Pose. Falls alle Lösungen verworfen werden müssen, liegt eine unerreichbare Zielpose vor. Die in der Wissensbasis gespeicherten transzendenten Gleichungen zusammen mit ihren symbolischen Lösungen werden nun nach wachsender Komplexität aufgelistet und bewiesen.

Satz 4.1. *Gegeben sind die beiden folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d, wobei $b \neq 0$ oder $d \neq 0$ gilt. Unter der Randbedingung $a \neq 0$ und $c \neq 0$ existiert eine eindeutige Lösung für die unbekannte Gelenkvariable θ_i.*

$$\begin{aligned} a \cdot \sin\theta_i &= b \\ c \cdot \cos\theta_i &= d \end{aligned} \quad \Longrightarrow \quad \theta_i = \arctan_2\left(\frac{b}{a}, \frac{d}{c}\right)$$

Beweis: $\theta_i = \arctan_2(\sin\theta_i, \cos\theta_i)$ □

Satz 4.2. *Gegeben sind die beiden folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d, wobei $c \neq 0$ oder $d \neq 0$ gilt. Unter der Randbedingung $a \neq 0$ und $b \neq 0$ existiert eine eindeutige Lösung für die unbekannte Gelenkvariable θ_i.*

$$\begin{aligned} a \cdot \cos\theta_i - b \cdot \sin\theta_i &= c \\ a \cdot \sin\theta_i + b \cdot \cos\theta_i &= d \end{aligned} \quad \Longrightarrow \quad \theta_i = \arctan_2(a \cdot d - b \cdot c, a \cdot c + b \cdot d)$$

Beweis: Da $a^2 + b^2 = c^2 + d^2 \neq 0$ gilt, können diese Gleichungen folgendermaßen umgeschrieben werden:

$$\begin{pmatrix} a & -b \\ b & a \end{pmatrix} \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} \Longleftrightarrow \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix} = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}^{-1} \begin{pmatrix} c \\ d \end{pmatrix} = \frac{1}{a^2 + b^2} \begin{pmatrix} a \cdot c + b \cdot d \\ a \cdot d - b \cdot c \end{pmatrix}$$

□

Satz 4.3. *Gegeben ist die folgende transzendente Gleichung mit den bekannten Termen a, b. Unter der Voraussetzung $a \neq 0$ gibt es zwei Lösungen für die unbekannte Gelenkvariable θ_i.*

$$a \cdot \sin\theta_i = b \quad \Longrightarrow \quad \theta_i^{(1,2)} = \arctan_2\left(\frac{b}{a}, \pm\sqrt{1 - \left(\frac{b}{a}\right)^2}\right)$$

Beweis: $\cos\theta_i = \pm\sqrt{1 - \sin^2\theta_i}$ □

Satz 4.4. *Gegeben ist die folgende transzendente Gleichung mit den bekannten Termen a, b. Unter der Voraussetzung $a \neq 0$ gibt es zwei Lösungen für die unbekannte Gelenkvariable θ_i.*

$$a \cdot \cos\theta_i = b \quad \Longrightarrow \quad \theta_i^{(1,2)} = \arctan_2\left(\pm\sqrt{1 - \left(\frac{b}{a}\right)^2}, \frac{b}{a}\right)$$

Beweis: $\sin\theta_i = \pm\sqrt{1 - \cos^2\theta_i}$ □

Satz 4.5. *Gegeben ist die folgende transzendente Gleichung mit den bekannten Termen a, b, c, wobei $c \neq 0$ gilt. Unter der Randbedingung $a \neq 0$ und $b \neq 0$ gibt es zwei Lösungen für die unbekannte Gelenkvariable θ_i.*

$$a \cdot \cos\theta_i + b \cdot \sin\theta_i = c \quad \Longrightarrow \quad \theta_i^{(1,2)} = \arctan_2(c, \pm\sqrt{a^2 + b^2 - c^2}) - \arctan_2(a, b)$$

Beweis: Es wird eine neue Variable $\phi = \arctan_2(a, b)$ eingeführt und dann die Termersetzungen $r = \sqrt{a^2 + b^2}$, $a = r \cdot \sin\phi$ und $b = r \cdot \cos\phi$ angewendet.

$$\begin{aligned}
& a \cdot \cos\theta_i + b \cdot \sin\theta_i = c \\
\Leftrightarrow \quad & \sin\phi \cdot \cos\theta_i + \cos\phi \cdot \sin\theta_i = c/r \\
\Leftrightarrow \quad & \sin(\phi + \theta_i) = c/r \\
\Leftrightarrow \quad & \phi + \theta_i = \arctan_2(c/r, \pm\sqrt{1 - (c/r)^2}) \\
\Leftrightarrow \quad & \theta_i^{(1,2)} = \arctan_2(c, \pm\sqrt{a^2 + b^2 - c^2}) - \arctan_2(a, b)
\end{aligned}$$

□

Satz 4.6. *Gegeben ist die folgende transzendente Gleichung mit den bekannten Termen a, b. Unter der Randbedingung $a \neq 0$ und $b \neq 0$ gibt es zwei um $180°$ versetzte Lösungen für die unbekannte Gelenkvariable θ_i.*

$$a \cdot \cos\theta_i + b \cdot \sin\theta_i = 0 \quad \Longrightarrow \quad \theta_i^{(1,2)} = \arctan_2(b, a) \pm \pi/2$$

Beweis: Analog zur vorherigen Beweisführung. □

Satz 4.7. *Gegeben sind die zwei folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d, wobei $c \neq 0$ oder $d \neq 0$ gilt und den unbekannten Gelenkvariablen θ_i und θ_j. Unter der Randbedingung $a \neq 0$ existieren zwei Lösungen für die Gelenkvariable θ_i, wobei in der Lösungsgleichung die Hilfsvariable $h = (a^2 - b^2 + c^2 + d^2)/(2 \cdot a)$ verwendet wird.*

$$\begin{aligned} a \cdot \cos\theta_i + b \cdot \cos\theta_j &= c \\ a \cdot \sin\theta_i + b \cdot \sin\theta_j &= d \end{aligned} \quad \Longrightarrow \quad \theta_i^{(1,2)} = \arctan_2(h, \pm\sqrt{c^2 + d^2 - h^2}) - \arctan_2(c, d)$$

Beweis: Der Term $a \cdot \cos\theta_i$ wird in der ersten Gleichung und der Term $a \cdot \sin\theta_i$ wird in der zweiten Gleichung auf die rechte Seite gebracht. Die umgeformten Gleichungen werden zuerst quadriert und danach addiert. Auf diese Weise entsteht eine neue Gleichung in der nur noch θ_i, aber nicht mehr θ_j vorkommt.

$$\begin{aligned}
& b^2 = a^2 + c^2 + d^2 - 2 \cdot a \cdot c \cdot \cos\theta_i - 2 \cdot a \cdot d \cdot \sin\theta_i \\
\Leftrightarrow \quad & c \cdot \cos\theta_i + d \cdot \sin\theta_i = (a^2 - b^2 + c^2 + d^2)/(2 \cdot a)
\end{aligned}$$

Die zwei Lösungen dieser Gleichung werden mit Hilfe von Satz 4.5 bestimmt. □

Satz 4.8. *Gegeben sind die zwei folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d und den unbekannten Gelenkvariablen θ_i und θ_j, wobei $c \neq 0$ oder $d \neq 0$ gilt. Unter der Randbedingung $a \neq 0$ und $b \neq 0$ existieren zwei Lösungen für die Gelenkvariable θ_j, wobei in der Lösungsgleichung die Hilfsvariable $h = (c^2 + d^2 - a^2 - b^2)/(2 \cdot a \cdot b)$ verwendet wird. Für die Gelenkvariable θ_i existiert eine Lösung, wobei in der Lösungsgleichung die Hilfsvariablen $h_1 = a \cdot \cos\theta_j + b$ und $h_2 = a \cdot \sin\theta_j$ verwendet werden.*

$$\begin{aligned} a \cdot \cos(\theta_i + \theta_j) + b \cdot \cos\theta_i &= c \\ a \cdot \sin(\theta_i + \theta_j) + b \cdot \sin\theta_i &= d \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \theta_j^{(1,2)} &= \arctan_2(\pm\sqrt{1 - h^2}, h) \\ \theta_i &= \arctan_2(d \cdot h_1 - c \cdot h_2, c \cdot h_1 + d \cdot h_2) \end{aligned}$$

Beweis: Um θ_j zu bestimmen, werden beide Seiten der transzendenten Gleichungen quadriert und anschließend addiert. Dadurch ergibt sich eine neue Gleichung, die nur noch θ_j, aber nicht mehr θ_i enthält.

$$\begin{aligned} & a^2 + 2 \cdot a \cdot b \cdot \cos\theta_j + b^2 = c^2 + d^2 \\ \Leftrightarrow \quad & \cos\theta_j = (c^2 + d^2 - a^2 - b^2)/(2 \cdot a \cdot b) \\ \Leftrightarrow \quad & \theta_j^{(1,2)} = \arctan_2(\pm\sqrt{1-h^2}, h) \end{aligned}$$

Nachdem θ_j bekannt ist, wird θ_i bestimmt. Die beiden transzendenten Gleichungen werden durch Anwendung der Additionstheoreme und Substitution der Hilfsvariablen h_1 und h_2 umgeformt. Dadurch werden die Gleichungen in die zwei äquivalenten Gleichungen $h_1 \cdot \cos\theta_i - h_2 \cdot \sin\theta_i = c$ und $h_1 \cdot \sin\theta_i + h_2 \cdot \cos\theta_i = d$ umgeformt. Durch Anwendung des Satzes 4.2 auf diese Gleichungen wird die Lösung für θ_i umittelbar abgelesen. □

Satz 4.9. *Gegeben sind die zwei folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d, e und den unbekannten Gelenkvariablen θ_i und θ_j, wobei $d \neq 0$ oder $e \neq 0$ gilt. Unter der Randbedingung $a \neq 0$ oder $b \neq 0$ und $c \neq 0$ existieren zwei Lösungen für die Gelenkvariable θ_j, wobei in der Lösungsgleichung die Hilfsvariable $h = (d^2 + e^2 - a^2 - b^2 - c^2)/(2 \cdot c)$ verwendet wird. Für die Gelenkvariable θ_i existiert eine Lösung, wobei in der Lösungsgleichung die Hilfsvariablen $h_1 = a \cdot \sin\theta_j + b \cdot \cos\theta_j + c$ und $h_2 = b \cdot \sin\theta_j - a \cdot \cos\theta_j$ verwendet werden.*

$$\begin{aligned} & a \cdot \sin(\theta_i + \theta_j) + b \cdot \cos(\theta_i + \theta_j) + c \cdot \cos\theta_i = d \\ & b \cdot \sin(\theta_i + \theta_j) - a \cdot \cos(\theta_i + \theta_j) + c \cdot \sin\theta_i = e \\ \Longrightarrow \quad & \theta_j^{(1,2)} = \arctan_2(h, \pm\sqrt{a^2 + b^2 - h^2}) - \arctan_2(b, a) \\ & \theta_i = \arctan_2(e \cdot h_1 - d \cdot h_2, d \cdot h_1 + e \cdot h_2) \end{aligned}$$

Beweis: Die beiden transzendenten Gleichungen werden quadriert und dann addiert. Dadurch kann aus diesen Gleichungen eine neue Gleichung, in der nur noch eine unbekannte Variable auftaucht, ermittelt werden.

$$\begin{aligned} & a^2 + 2 \cdot a \cdot c \cdot \sin\theta_j + b^2 + 2 \cdot b \cdot c \cdot \cos\theta_j + c^2 = d^2 + e^2 \\ \Leftrightarrow \quad & b \cdot \cos\theta_j + a \cdot \sin\theta_j = (d^2 + e^2 - a^2 - b^2 - c^2)/(2 \cdot c) \end{aligned}$$

Die zwei Lösungen von θ_j werden mit Hilfe von Satz 4.5 bestimmt. Anschließend wird die Lösung von θ_i in Abhängigkeit von θ_j bestimmt. Durch Anwendung der Additionstheoreme und Substitution der Hilfsvariablen h_1 und h_2 werden die zwei transzendenten Gleichungen in die Gleichungen $h_1 \cdot \cos\theta_i - h_2 \cdot \sin\theta_i = d$ und $h_1 \cdot \sin\theta_i + h_2 \cdot \cos\theta_i = e$ überführt. Auf diese beiden Gleichungen wird der Satz 4.2 angewendet, wodurch man eine eindeutige Lösung für θ_i erhält. □

Satz 4.10. *Gegeben sind die zwei folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d, e und den unbekannten Gelenkvariablen θ_i, θ_j und θ_k, wobei $d \neq 0$ oder $e \neq 0$ gilt. Unter der Randbedingung $d^2 + e^2 >= c^2$ existieren zwei Lösungen für die Gelenkvariable θ_i, wobei in der Lösungsgleichung die Hilfsvariable $h = \pm\sqrt{d^2 + e^2 - c^2}$ verwendet wird.*

$$\begin{aligned} & a \cdot \cos\theta_i \cdot \cos(\theta_j + \theta_k) + b \cdot \cos\theta_i \cdot \cos\theta_j - c \cdot \sin\theta_i = d \\ & a \cdot \sin\theta_i \cdot \cos(\theta_j + \theta_k) + b \cdot \sin\theta_i \cdot \cos\theta_j + c \cdot \cos\theta_i = e \\ \Longrightarrow \quad & \theta_i^{(1,2)} = \arctan_2\left(h \cdot e - c \cdot d, h \cdot d + c \cdot e\right) \end{aligned}$$

Beweis: Durch Quadrieren und Addieren der beiden transzendenten Gleichungen entsteht die Gleichung $(a\cdot\cos(\theta_j+\theta_k)+b\cdot\cos\theta_j)^2 = d^2+e^2-c^2$. Die Gleichungen $h\cdot\cos\theta_i - c\cdot\sin\theta_i = d$ und $h\cdot\sin\theta_i + c\cdot\cos\theta_i = e$ sind deshalb zu den ursprünglichen transzendenten Gleichungen äquivalent. Die Lösung für θ_i wird aus diesen Gleichungen mit Satz 4.2 bestimmt. □

Satz 4.11. *Gegeben sind die drei folgenden transzendenten Gleichungen mit den bekannten Termen a, b, c, d, e und den unbekannten Gelenkvariablen θ_i, θ_j und θ_k. Unter der Randbedingung $a \neq 0$ und $b \neq 0$ existieren zwei Lösungen für die Gelenkvariable θ_k, wobei in der Lösungsgleichung die Hilfsvariable $h = (c^2 + d^2 + e^2 - a^2 - b^2)/(2 \cdot a \cdot b)$ verwendet wird.*

$$\begin{aligned} a \cdot \cos\theta_i \cdot \cos\theta_j \cdot \sin\theta_k - a \cdot \sin\theta_i \cdot \cos\theta_k - b \cdot \sin\theta_i &= c \\ a \cdot \sin\theta_i \cdot \cos\theta_j \cdot \sin\theta_k + a \cdot \cos\theta_i \cdot \cos\theta_k + b \cdot \cos\theta_i &= d \\ a \cdot \sin\theta_j \cdot \sin\theta_k &= e \\ \Longrightarrow \quad \theta_k^{(1,2)} = \arctan_2(\pm\sqrt{1-h^2}, h) & \end{aligned}$$

Beweis: Die drei transzendenten Gleichungen werden quadriert und dann miteinander addiert. Dadurch ergibt sich eine neue Gleichung, in der nur noch θ_k vorkommt.

$$\begin{aligned} & a^2 + 2 \cdot a \cdot b \cdot \cos\theta_k + b^2 = c^2 + d^2 + e^2 \\ \Leftrightarrow \quad & \cos\theta_k = (c^2 + d^2 + e^2 - a^2 - b^2)/(2 \cdot a \cdot b) \end{aligned}$$

Die zwei Lösungen von θ_k werden mit Hilfe von Satz 4.4 bestimmt. □

4.3.2 Pattern Matching mit dem RETE–Algorithmus

Den größten Teil der Laufzeit bei der Abarbeitung von regelbasierten Anwendungen (Abbildung 4.3) macht das Pattern Matching aus. Es wurde geschätzt, daß hierfür über 90% der Laufzeit benötigt wird. Ein sehr einfacher Pattern Matching Algorithmus ist, in jeder Iteration die Liste der Regeln zu durchlaufen und die linke Seite jeder Regel auf Erfüllbarkeit hinsichtlich des gesamten Arbeitsspeichers zu überprüfen. Dieser naive Algorithmus hat exponentielle Komplexität und führt deswegen im allgemeinen zu unakzeptablen langen Laufzeiten. Um zu überprüfen, ob für eine Gleichung oder eine Kombination von Gleichungen eine Lösung in der Wissensbasis vorkommt, können verschiedene Algorithmen verwendet werden. Miranker [114, 115] hat beispielsweise die sogenanten TREAT–Strukturen und Batory [8, 9] den LEAPS–Algorithmus (Lazy Evaluation Algorithm for Production Systems) entwickelt. Ein weiterer Algorithmus mit einem besseren Laufzeitverhalten als bei TREAT und LEAPS ist der RETE–Algorithmus (lat.: Netz), der deswegen in dieser Arbeit verwendet wird. Der RETE–Algorithmus wurde in den achtziger Jahren von C. Forgy [52, 53] an der Carnegie Mellon Universität entwickelt und ist die Basis für viele *regelbasierte Expertensysteme*. Mit dem RETE–Algorithmus werden zur Lösung des Vergleichsproblems, das bei der Regelauswertung entsteht, relativ kurze Laufzeiten auf Kosten des Speicherbedarfs erzielt. Der Algorithmus verbessert die Zeitkomplexität des naiven Algorithmus von $\mathcal{O}(R\cdot F^P)$ nach $\mathcal{O}(R\cdot F\cdot P)$, wobei R die Anzahl der Regeln in der Wissensbasis, F die Anzahl der Fakten im Arbeitsspeicher und P die mittlere Anzahl von Bedingungen pro Regel sind. Diese Effizienzsteigerung wird durch die beiden folgenden Einschränkungen erreicht.

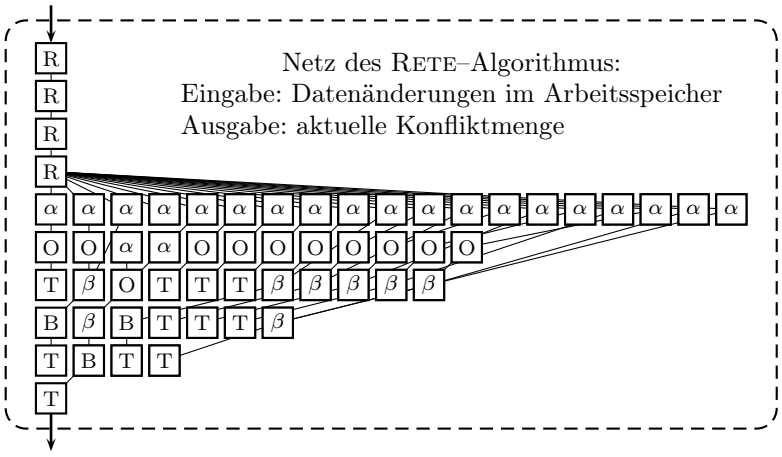
1. Einschränkung der zu überprüfenden Fakten. Durch Anwendung einer Regel ändert sich meistens nur ein Teil des Arbeitsspeichers. Die Ergebnisse des vorangegangenen

Vergleichs des Inhalts des Arbeitsspeichers und der Bedingungen von Regeln werden gespeichert. Ein Vergleich der Bedingungsteile mit den unveränderten Fakten des Arbeitsspeichers lässt die Konfliktmenge gleich. Deswegen werden nur geänderte Fakten im Arbeitsspeicher daraufhin geprüft, ob Regeln, die bisher nicht angewendet werden konnten, jetzt angewendet werden können beziehungsweise Regeln, die bisher angewendet werden konnten, jetzt nicht mehr angewendet werden können.

2. Einschränkung der zu überprüfenden Bedingungen. Haben mehrere Regeln Prämissen gemeinsam, so werden diese Bedingungen nur einmal ausgewertet. Da die Bedingungsteile der Regeln im allgemeinen nicht disjunkt sind, werden gleichartige Bedingungen zusammengefasst und dadurch der Auswerteaufwand reduziert.

Die Bedingungen der in Abschnitt 4.3.1 definierten Regeln werden automatisch in das RETE–Netzwerk übersetzt. Die Aktionen der Regeln werden nicht in das RETE–Netzwerk aufgenommen. Als Eingabe erhält das RETE–Netzwerk die Änderungen des Arbeitsspeichers und als Ergebnis wird die aktuelle Konfliktmenge ausgegeben (Abbildung 4.4).

Abbildung 4.4 Generiertes RETE–Netzwerk für die Wissensbasis



Das RETE–Netzwerk repräsentiert die Bedingungen der Regeln in der Wissensbasis in Form eines gerichteten, azyklischen Graphen aus Kanten und verschiedenen Typen von Knoten. Jede Änderung des Arbeitsspeichers wird an den Wurzelknoten des RETE–Netzwerks übergeben, wo sie entlang der Kanten weitergeleitet und Operationen durchgeführt werden. Zur Vermeidung redundanter Operationen werden bereits berechnete Vergleichsergebnisse und insbesondere feuerbereite Prämissen und Fakten zwischengespeichert. Das RETE–Netzwerk besteht aus mehreren Typen von Knoten, die die eigentlichen Operationen repräsentieren.

R–Knoten: Wurzelknoten des Netzwerkes. Vorverarbeitung der Eingaben.

***α*–Knoten:** Selektionsbedingung, die sich auf einzelne Fakten des Arbeitsspeichers bezieht. Jede Teilbedingung einer Regel wird durch einen solchen Knoten dargestellt, wobei identische Teilbedinungen in verschiedenen Regeln durch denselben Knoten repräsentiert werden (engl.: node sharing). Die Ausgabe des Knotens wird in einem α–Speicher abgelegt. Hier werden diejenigen Fakten gespeichert, die die Bedingung des Knotens erfüllen. Dieser Knotentyp wird auch als Prämissenknoten (engl.: one–input node) bezeichnet.

O–Knoten: Verbindung der ersten Teilbedingung einer Regel mit dem Netzwerk.

β–Knoten: Verbund– beziehungsweise Verschmelzungsbedingung, die α– und β–Knoten über Junktoren und Variablenbindungen miteinander verknüpfen. Durch β–Knoten werden Einzelbedingungen einer Regel miteinander verbunden, so dass eine konsistente Variablenbelegung entsteht. Jeder β–Knoten hat zwei Eingänge in die die Ausgaben der Vorgängerknoten eingelesen werden. Die Ausgabe des Knotens wird in einem β–Speicher abgelegt. Dieser Knotentyp wird auch als Verbindungsknoten (engl.: two–input node) bezeichnet.

B–Knoten: Seperater Knoten, der einen booleschen Ausdruck testet.

T–Knoten: Terminalknoten, der angibt, dass alle Bedingungen einer Regel erfüllt sind.

Beim ersten Durchlauf werden sämtliche Prämissenknoten evaluiert und die Ergebnisse gespeichert. Ab dem zweiten Durchlauf werden nur diejenigen Prämissenknoten ausgewertet, die sich auf geänderte Fakten beziehen. Der Inhalt in den α– und β–Speichern wird dann gegebenenfalls überschrieben.

4.3.3 Experimentelle Ergebnisse

Die automatisierte Lösung des kinematischen Problems wurde in dem Produktionssystem Jess (Java Expert System Shell) [57] in der aktuellen Version 7.1 implementiert. Jess wurde gewählt, da es den Rete–Algorithmus in die Programmiersprache Java einbettet und für Lehr– und Forschungszwecke frei erhältlich ist. Außerdem steht eine umfangreiche Dokumentation mit verschiedenen Beispielprogrammen zur Verfügung. Jess wird seit 1995 von E. Friedman–Hill an den Sandia National Laboratories in Kalifornien entwickelt. Anfangs orientierte sich Jess an der Expertensystemschale Clips (C Language Integrated Production System) [62], die von G. Riley in der Programmiersprache C entwickelt wurde. Jess ist daher weitestgehend kompatibel zu dieser Regelsprache, übertrifft inzwischen allerdings die Performanz dieser Sprache in einer Reihe von Experimenten. Da es sich bei Jess nicht um ein Computeralgebrasystem handelt, wurde es in dieser Arbeit um Funktionalitäten für das symbolische Rechnen, beispielsweise das Vereinfachen algebraischer Ausdrücke und das analytische Lösen nichtlinearer Gleichungen erweitert.

Der zeitliche Aufwand zum Aufstellen und Lösen des kinematischen Gleichungssystems ist für einige ausgewählte Robotertypen in Tabelle 4.2 wiedergegeben. Rotationsgelenke werden mit R und Translationsgelenke mit T abgekürzt. Es wurde ein Pentium Prozessor mit 2.0 GHz Taktrate und das Betriebssystem Linux verwendet. Der Zeitbedarf zum Aufstellen der kinematischen Gleichungen hängt von der Anzahl der Gelenkachsen ab. Mit steigender Zahl an kinematischen Lösungen steigt auch der Zeitbedarf zum Lösen der Gleichungen. Obwohl es sich bei Jess um interpretierten und nicht um kompilierten Code handelt, können die symbolischen Lösungen in wenigen Sekunden hergeleitet werden. Aus diesen Lösungen wird ausführbarer C/C++ Code automatisch generiert.

Zu einer Endeffektorlage werden auch eventuelle Mehrfachlösungen bestimmt. Für einen üblichen sechsachsigen Gelenkarmroboter mit Zentralhand werden acht Gelenkkonfigurationen für eine vorgegebene Lage des Endeffektors ermittelt. Diese Konfigurationen werden als „Arm links“ oder „Arm rechts“ beziehungsweise „Ellbogen oben“ oder „Ellbogen unten“ bezeichnet. Falls es mehrere Lösungen gibt, so werden bei der Auswahl einer geeigneten Lösung Gelenkanschläge und mögliche Hindernisse beachtet. Die ermittelten Lösungen

Tabelle 4.2 Zeitbedarf zum Aufstellen und Lösen der kinematischen Gleichungen

Roboter	Konfiguration	Zeitbedarf (min:sec) Aufstellen	Lösen	Anzahl der Lösungen
Kartesischer Roboter	TTT RR	00:03	00:02	1
Scara I	RRT R	00:01	00:03	2
Scara II	TRR R	00:01	00:03	2
Zylindrischer Roboter	RTT RRR	00:03	00:04	4
Stanford Arm	RRT RRR	00:04	00:07	8
Gelenkarmroboter	RRR RRR	00:04	00:10	8

werden überprüft, ob sie zwischen dem minimalen und maximalen Gelenkvariablenwert liegen und sofern dies nicht der Fall ist, verworfen. Um Stetigkeit zu erzielen wird von den verbleibenden Lösungen üblicherweise die naheste Lösung ausgewählt, das heißt die Lösung bei der die Gelenke minimal bewegt werden. Gegebenenfalls werden hierbei die Änderungen bei den verschiedenen Gelenkachsen unterschiedlich gewichtet.

Die kinematischen Gleichungen wurden außerdem nach der Anzahl unbekannter Gelenkvariablen eingestuft (Tabelle 4.3). Gleichungen in denen keine unbekannten Variablen vorkommen, tragen nicht zu einer Lösung bei und werden verworfen. Die Anzahl Gleichungen mit einer oder zwei Unbekannten nimmt ab und die Anzahl Gleichungen mit fünf oder sechs Unbekannten nimmt zu, wenn Roboter mit zunehmend komplexeren Geometrien betrachtet werden. Handelt es sich bei einer unbekannten Gelenkvariable um ein Rotationsgelenk, so kommt die Variable als Argument von Sinus– und Cosinusfunktionen vor. Dadurch taucht diese Unbekannte in sehr viel mehr Gleichungen auf und ist schwieriger zu ermitteln, als wenn es sich um die Variable eines Translationsgelenks handelt.

Tabelle 4.3 Komplexität der zu lösenden kinematischen Gleichungssysteme

Roboter	Anzahl Gleichungen mit $x = 0, \ldots, 6$ Unbekannten							
	0	1	2	3	4	5	6	$\sum$
Kartesischer Roboter	11	129	92	5	10	5	–	252
Scara I	26	25	23	46	0	–	–	120
Scara II	32	17	28	43	0	–	–	120
Zylindrischer Roboter	0	19	59	119	50	5	0	252
Stanford Arm	0	8	11	75	116	42	0	252
Gelenkarmroboter	0	6	4	47	64	89	42	252

Bei der Auswertung in Echtzeit müssen die kinematischen Modellgleichungen in genau festgelegten Zeitintervallen berechnet werden. Die primären Kriterien bei der Evaluation der generierten kinematischen Modelle sind die Anzahl der erforderlichen arithmetischen Operationen und der hierfür erforderliche Rechenaufwand. Die Anzahl Operationen der direkten Kinematik kann leicht anhand der Transformationsmatrix $T_{0,n}$ bestimmt werden. Bei der inversen Kinematik ist zu berücksichtigen, dass, falls es mehrere Lösungen gibt, ein Baum zur Berechnung aller Konfigurationen durchlaufen werden muß. Anhand eines

Beispielroboters wird dieses Vorgehen im Anhang dargestellt (Abbildung B.1).

Die erforderlichen Operationen der generierten kinematischen Modelle sind in Tabelle 4.4 dargestellt. Zur Minimierung der Rechenzeit werden redundante Berechnungen bei der Generierung automatisch eliminiert. Im Falle von zwei aufeinanderfolgenden, parallelen Rotationsachsen werden die Gleichungen durch Anwendung der Additionstheoreme vereinfacht. Dadurch entstehen Terme, in denen die Summe beziehungsweise Differenz zweier Gelenkwinkel auftauchen. Für die Berechnung aller Lösungen des inversen kinematischen Modells eines sechsachsigen Gelenkarmroboters wurde in einer Simulationsumgebung eine Rechenzeit von weniger als fünf Mikrosekunden gemessen. Deswegen genügen die generierten Modelle den strengen Echtzeitanforderungen einer Bewegungssteuerung. Der Rechenaufwand kann zudem weiter reduziert werden. Bei der Ausführung in Echtzeit müssen nicht in jedem Interpolationstakt alle Lösungen berechnet werden, da normalerweise nicht zwischen verschiedenen Konfigurationen gesprungen werden darf. Andernfalls würde der Roboter sich ruckartig bewegen und die Mechanik zu sehr beansprucht.

Tabelle 4.4 Evaluation des kinematischen Robotermodells für verschiedene Roboter

	Anzahl arithmetische Operatoren der direkten Kinematik								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	0	0	4	0	2	2	0	0	0
Scara I	8	5	4	0	4	4	0	0	0
Scara II	12	1	4	0	4	4	0	0	0
Zylindrischer Roboter	8	10	55	0	4	4	0	0	0
Stanford Arm	19	16	123	0	5	5	0	0	0
Gelenkarmroboter	54	18	128	0	21	22	0	0	0

	Anzahl arithmetische Operatoren der inversen Kinematik								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	0	0	0	0	0	0	2	0	0
Scara I	6	6	7	1	1	1	4	1	7
Scara II	6	6	7	1	1	1	4	1	7
Zylindrischer Roboter	5	12	28	0	2	2	4	0	0
Stanford Arm	17	15	70	0	3	3	6	1	3
Gelenkarmroboter	36	18	67	1	11	13	7	1	7

4.4 Algebraische Lösung der inversen Kinematik

Manipulatoren bei denen sich die Handachsen in einem Punkt schneiden, haben den Vorteil, dass die Orientierung von der Position des Endeffektors entkoppelt ist. Parallele Rotationsachsen führen dazu, dass sich Terme in den kinematischen Gleichungen zusammenfassen lassen. Falls eines dieser beiden Kriterien nicht erfüllt ist, so erhöht sich die Komplexität der zu lösenden Gleichungen. In Tabelle 4.5 sind die Denavit–Hartenberg Parameter eines Manipulators (Typ I) aufgeführt, bei dem es keinen gemeinsamen Schnittpunkt der drei Handachsen gibt. In der Tabelle sind außerdem die Denavit–Hartenberg Parameter eines Manipulators (Typ II) ohne parallele Gelenkachsen aufgeführt. Beide

Manipulatoren haben jeweils insgesamt sechs Rotationsgelenke.

Tabelle 4.5 Denavit–Hartenberg Parameter zweier Manipulatoren (Typ I und Typ II)

Gelenk i	θ_i	Offset	d_i	a_i	α_i
1	θ_1	0°	0	0	90°
2	θ_2	0°	0	a_2	0°
3	θ_3	0°	0	a_3	0°
4	θ_4	0°	0	a_4	−90°
5	θ_5	0°	0	a_5	90°
6	θ_6	0°	0	a_6	0°

Gelenk i	θ_i	Offset	d_i	a_i	α_i
1	θ_1	0°	0	0	−90°
2	θ_2	0°	d_2	0	90°
3	θ_3	−90°	0	$-a_3$	90°
4	θ_4	0°	d_4	0	−90°
5	θ_5	0°	0	0	90°
6	θ_6	0°	d_6	0	0°

Zur Bestimmung einer geschlossenen Lösung des kinematischen Robotermodells werden die folgenden Schritte durchgeführt. Zuerst werden die homogenen Transformationsmatrizen und das direkte kinematische Modell bestimmt und dann das kinematische Gleichungssystem aufgestellt (Abbildung 4.2), um die unbekannten Gelenkvariablen zu isolieren. Um eine möglichst allgemeine Lösung zu erhalten, werden keine konkreten Zahlen für die Roboterabmessungen und das Endeffektorkoordinatensystem verwendet, sondern mit symbolischen Variablen gerechnet. Die generierten Gleichungen werden entsprechend der Anzahl unbekannter Variablen gruppiert (Tabelle 4.6). Für diese beiden Manipulatoren entstehen jedoch keine Gleichungen, die nur noch eine Unbekannte enthalten. Insbesondere werden keine geeigneten Gleichungen generiert, die ausschließlich mit den wissensbasierten Methoden aus Abschnitt 4.3 gelöst werden können.

Tabelle 4.6 Komplexität der zu lösenden kinematischen Gleichungssysteme

	Anzahl Gleichungen mit $x = 0, \ldots, 6$ Unbekannten							
Roboter	0	1	2	3	4	5	6	$\sum$
Typ I	0	0	29	38	32	93	60	252
Typ II	0	0	12	15	79	112	34	252

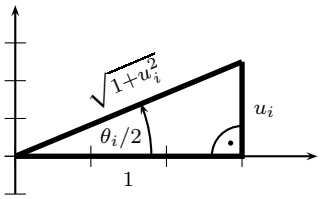
Deswegen wird im folgenden versucht, diese Gleichungen in einfacher zu lösende Gleichungen zu überführen und insbesondere eine Gleichung mit nur einer Unbekannten zu bestimmen. Hierfür wird das trigonometrische Gleichungssystem auf eine Menge von Polynomgleichungen zurückgeführt und anschließend die *Gröbnerbasis* berechnet.

4.4.1 Konvertierung in Polynomialgleichungen

Um ein trigonometrisches Gleichungssystem in ein polynomiales Gleichungssystem umzuwandeln, sind zwei verschiedene Transformationen möglich. Die erste Transformation ist die sogenannte *Halbwinkelsubstitution* (Abbildung 4.5) bei der für jede unbekannte Gelenkwinkelvariable θ_i eine neue Variable u_i eingeführt wird und die folgenden Substitutionen durchgeführt werden.

$$\cos\theta_i := \frac{1-u_i^2}{1+u_i^2} \qquad \sin\theta_i := \frac{2\cdot u_i}{1+u_i^2} \qquad \Longrightarrow \qquad \theta_i := 2\cdot\arctan(u_i) \tag{4.4}$$

Abbildung 4.5 Geometrische Veranschaulichung der Halbwinkelsubstitution



$$\cos\left(\frac{\theta_i}{2}\right) = \frac{1}{\sqrt{1+u_i^2}} \qquad \cos\theta_i = \cos^2\left(\frac{\theta_i}{2}\right) - \sin^2\left(\frac{\theta_i}{2}\right) = \frac{1-u_i^2}{1+u_i^2}$$

$$\sin\left(\frac{\theta_i}{2}\right) = \frac{u_i}{\sqrt{1+u_i^2}} \qquad \sin\theta_i = 2\cdot\cos\left(\frac{\theta_i}{2}\right)\cdot\sin\left(\frac{\theta_i}{2}\right) = \frac{2\cdot u_i}{1+u_i^2}$$

Die resultierenden Bruchgleichungen werden mit dem jeweiligen Hauptnenner erweitert, um so eine Polynomialdarstellung zu erhalten. Aufgrund des Faktors zwei in der Lösungsgleichung für θ_i, ist es nicht notwendig den erweiterten Arcustangens zu verwenden, um eine eindeutige Lösung für die unbekannte Gelenkvariable zu erhalten.

Eine weitere Transformation ist es, für jede unbekannte Gelenkwinkelvariable θ_i zwei Variablen c_i und s_i sowie eine zusätzliche Gleichung einzuführen. Die Kosinus– und Sinusterme werden durch die neuen Variablen substituiert. Die Polynomialgleichung, die für jede unbekannte Gelenkwinkelvariable eingeführt wird, stellt die trigonometrische Beziehung zwischen den neuen Unbekannten c_i und s_i her.

$$\cos\theta_i := c_i \qquad \sin\theta_i := s_i \qquad c_i^2 + s_i^2 = 1 \qquad \Longrightarrow \qquad \theta_i := \arctan_2(s_i, c_i) \tag{4.5}$$

Während bei der ersten Transformation die Anzahl der Gleichungen gleich bleibt, erhöht sich bei der zweiten Transformation die Anzahl der Gleichungen. Andererseits ist bei der ersten Transformation der Grad der Gleichungen höher als bei der zweiten Transformation.

4.4.2 Bestimmung der Gröbnerbasis

Das mit diesen Transformationen (Abschnitt 4.4.1) erhaltene multivariate Gleichungssystem soll nun symbolisch gelöst werden. In dem Gleichungssystem $F = \{f_1, \ldots, f_m\}$ steht n für die Anzahl der unbekannten Variablen und m für die Anzahl der Polynomgleichungen.

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \qquad \Longrightarrow \qquad \begin{aligned} g_1(x_1) &= 0 \\ g_2(x_1, x_2) &= 0 \\ &\vdots \\ g_t(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

Die *Gröbnerbasis* ist eine Menge von Polynomialgleichungen $GB = \{g_1, \ldots, g_t\}$, die zu den ursprünglichen Gleichungen äquivalent sind, allerdings von der Form ist, dass die

Lösungen der Unbekannten nacheinander bestimmt werden kann. Insbesondere erhält man, sofern eine solche Gleichung existiert, eine Gleichung mit nur einer Unbekannten. Dieses univariate Polynom wird benutzt, um alle möglichen Werte eines unbekannten Gelenkwinkels zu ermitteln. Die Nullstellen von Polynomen bis zum vierten Grad können analytisch bestimmt werden, während sie bei höhergradigen Polynomen nur noch numerisch bestimmbar sind. Die restlichen unbekannten Gelenkvariablen können dann wiederum mit den wissensbasierten Methoden aus Abschnitt 4.3 gelöst werden. Ist das Gleichungssystem nicht geschlossen lösbar, so ergibt sich die Gröbnerbasis zu $GB = \{1\}$. Zur Bestimmung der Gröbnerbasis ist die Vorgabe einer Variablenreihenfolge erforderlich. Eine Variable mit einer höheren Ordnung taucht früher in der Gröbnerbasis auf. Die benötigte Rechenzeit zur Bestimmung einer Gröbnerbasis hängt entscheidend von der Variablenreihenfolge ab. Außerdem wird bei einer günstigen Variablenreihenfolge die Gröbnerbasis sehr viel kleiner als bei einer ungünstigen. Cox [30] zeigte durch polynomiale Reduktion des $\mathcal{NP}$–vollständigen 3–Färbbarkeitsproblem (engl.: graph coloring) auf die Berechnung von Gröbnerbasen, dass dieses Problem $\mathcal{NP}$–hart ist. Es wurde mehrere Algorithmen entwickelt, um Gröbnerbasen zu bestimmen, die im folgenden kurz vorgestellt werden.

Buchberger Algorithmus

Der erste Algorithmus wurde von Buchberger in seiner Dissertation entwickelt [16, 17, 18, 19, 20, 21]. Die Idee des Algorithmus besteht darin, jeweils zwei Polynome derart zu kombinieren, so dass sich die führenden Monome bezüglich einer Termordnung gegenseitig wegheben. Hierbei sind verschiedene Termordnungen möglich, zum Beispiel lexikographische, graduiert–lexikographische oder graduiert–invers–lexikographische Ordnung. Die *lexikographische Ordnung* bezüglich zweier Monome ist wie folgt definiert:

$$x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n} \succ_{lex} x_1^{\beta_1} x_2^{\beta_2} \dots x_n^{\beta_n} :\Longleftrightarrow \exists j \in \{1, \dots, n\} : (\alpha_i = \beta_i \ \forall i < j) \wedge \alpha_j > \beta_j \quad (4.6)$$

Bei dieser Ordnung werden die beiden Monome von links nach rechts durchlaufen und miteinander verglichen. Diese Ordnung ist der alphabetischen Sortierung von Zeichenketten sehr ähnlich. Bei der *graduiert–lexikographischen Ordnung* werden Monome zunächst nach ihrem Gesamtgrad sortiert und bei Gleichheit die lexikographische Ordnung angewendet. Termordnungen können auch mit Hilfe von Matrizen definiert werden. Das bezüglich einer Ordnung als das höchste einzuordnende Monom einer Polynomfunktion f wird als Leitmonom $lm(f)$ und der Koeffizient des Leitmonoms als Leitkoeffizient $lk(f)$ bezeichnet. Das Produkt von Leitkoeffizient und Leitmonom wird als Leitterm $lt(f) = lk(f) \cdot lm(f)$ bezeichnet. Das *S–Polynom* zweier Polynomfunktionen f_1 und f_2, ist so definiert ist, dass sich die Leitterme der beiden Funktionen herausheben.

$$Spoly(f_1, f_2) = \frac{lcm(lm(f_1), lm(f_2))}{lt(f_1)} \cdot f_1 - \frac{lcm(lm(f_1), lm(f_2))}{lt(f_2)} \cdot f_2 \quad (4.7)$$

Der Operator lcm ist definiert als $lcm(x^\alpha, x^\beta) = x_1^{\max\{\alpha_1,\beta_1\}} \dots x_n^{\max\{\alpha_n,\beta_n\}}$ und berechnet das kleinste gemeinsame Vielfache (engl.: least common multiple) der beiden Operanden. Nachfolgend ist der Buchberger Algorithmus in seiner Grundform aufgeführt (Algorithmus 4.1). Mit der Funktion $NormalForm$ werden die S–Polynome durch alle Elemente der bisher berechneten Gröbnerbasis dividiert. Falls der Rest ungleich Null ist, wird er in die Gröbnerbasis aufgenommen. Das Abbruchkriterium des Algorithmus ist erfüllt, wenn der Rest der Division aller möglichen S–Polynome durch die Gröbnerbasis Null ist.

Algorithmus 4.1 Buchberger Algorithmus zur Berechnung von Gröbnerbasen

Input: Menge von Polynomgleichungen $F = \{f_1, f_2, \ldots, f_m\}$
Output: Gröbnerbasis $GB = \{g_1, g_2, \ldots, g_t\}$

$GB := F$;
$M := \{(f_i, f_j) \mid f_i, f_j \in GB \text{ und } 1 \leq i < j \leq m\}$;
while $M \neq \{\}$ **do**
    choose $\{(p, q)\} \in M$;
    $M := M \backslash \{(p, q)\}$;
    $h := NormalForm(Spoly(p, q), GB)$;
    if $h \neq \{\}$ **then**
        $M := M \cup \{(g, h) \mid g \in GB\}$;
        $GB := GB \cup \{h\}$;
return GB;

Die Nachteile dieses Algorithmus liegen in der Laufzeit und in dem Speicherbedarf. Für kompliziertere Gleichungen und bei ungünstiger Variablenreihenfolge kann die Gröbnerbasis nicht in annehmbarer Zeit berechnen. Die Elimination einer Unbekannten kann im schlechtesten Fall die Anzahl der Polynomgleichungen quadrieren. Bei n Ungleichungen und m Unbekannten ergibt sich doppelt exponentielle Komplexität $\mathcal{O}(n^{2^m})$ [107, 108].

XL Algorithmus

Im Jahr 2000 entwickelten Shamir, Patarin, Courtois und Klimov [28, 29] einen effizienteren Algorithmus als der von Buchberger, den XL–Algorithmus (Extended Linearization). Neue nichtlineare Gleichungen werden nicht durch Kombination von Polynomgleichungen hergeleitet, wie es beim Buchberger Algorithmus der Fall ist, sondern durch Multiplikation von Gleichungen mit allen möglichen unbekannten Monomen eines bestimmten Grades. In den hinzugekommenen Gleichungen werden die vorkommenden Terme als neue unbekannte Variablen betrachtet und diese Gleichungen mittels Gauß–Elimination gelöst.

Algorithmen F4 und F5

Eine weitere Verbesserung des Buchberger Algorithmus stellen die Algorithmen F4 [47] und F5 [48, 49] dar, die von Faugère eingeführt wurden. Diese beiden Algorithmen verwenden Methoden aus der linearen Algebra. Das Kernstück der Algorithmen besteht in der Erstellung und Triangulierung einer Matrix [93], wobei jedes Polynom als Matrixzeile dargestellt wird. Diese Matrix hat in der Regel sehr viel mehr Spalten als Zeilen. Simulationen haben mittlerweile gezeigt, dass der F4 Algorithmus für dieselben Polynomialgleichungen weniger große Matrizen erzeugt und Zeit verbraucht, als der XL Algorithmus [4].

4.4.3 Experimentelle Ergebnisse

Für diejenigen Roboter, deren geometrische Eigenschaften in den Tabellen 4.5 und 4.6 angegeben sind, werden die kinematischen Gleichungen mit zwei unbekannten Gelenkvariablen in Polynomialdarstellung gebracht und dann die äquivalente Gröbnerbasis bestimmt. Der Buchberger Algorithmus benötigt hierfür nur einige Sekunden an Rechenzeit. Als erstes wird der Robotertyp I betrachtet und das folgende, nichtlineare Gleichungssystem $F = \{f_1, f_2, f_3, f_4, f_5, f_6\}$ durch Substitution gemäß Gleichung (4.5) aufgestellt. Unbekannte Variablen sind s_1, c_1, s_5, c_5, s_6 und c_6, Denavit–Hartenberg Parameter sind a_5 und

a_6 und Variablen, die die Effektorlage angeben, sind n_x, n_y, o_x, o_y, a_x, a_y, p_x und p_y.

$$
\begin{aligned}
f_1 &= a_x \cdot s_1 - a_y \cdot c_1 - c_5 & f_4 &= c_1^2 + s_1^2 - 1 \\
f_2 &= -n_y \cdot c_1 \cdot s_6 - o_y \cdot c_1 \cdot c_6 + n_x \cdot s_1 \cdot s_6 + o_x \cdot c_6 \cdot s_1 & f_5 &= c_5^2 + s_5^2 - 1 \\
f_3 &= (-a_6 \cdot n_y + p_y) \cdot c_1 + (a_6 \cdot n_x - p_x) \cdot s_1 & f_6 &= c_6^2 + s_6^2 - 1 \\
&\quad + a_5 \cdot (-n_y \cdot c_1 \cdot c_6 + o_y \cdot c_1 \cdot s_6 + n_x \cdot c_6 \cdot s_1 - o_x \cdot s_1 \cdot s_6)
\end{aligned}
$$

Um diese Gleichungen zu entkoppeln und eine Polynomialgleichung mit einer Unbekannten zu erhalten, wird die Gröbnerbasis berechnet. Durch Anwendung der lexikographischen Ordnung mit $s_6 > c_6 > s_1 > c_1 > s_5 > c_5$ werden einfachere Gleichungen bestimmt. In dieser Variablenreihenfolge sind die Sinus– und Cosinusvariablen bezüglich der Gelenkvariable θ_6 führend. In Experimenten konnte keine Gröberbasis berechnet werden, wenn die Variablen bezüglich θ_1 oder θ_5 als führend gewählt werden. In den folgenden Gröbnerbasisgleichungen $GB = \{g_1, g_2, g_3, g_4, g_5, g_6\}$ werden die Hilfsvariablen k_{ij} verwendet, die Terme bestehend aus konstanten DH–Parametern und der Endeffektorlage (engl.: Tool Center Point, TCP) symbolisieren.

$$
\begin{aligned}
g_1 &= k_{11} \cdot s_6^2 + k_{12} \cdot s_6 & g_4 &= k_{41} \cdot s_1 \cdot s_6 + k_{42} \cdot s_1 + k_{43} \cdot c_1 \\
g_2 &= k_{21} \cdot c_6 + k_{22} \cdot s_6 + k_{23} & g_5 &= k_{51} \cdot s_5^2 + k_{52} \cdot s_6 + k_{53} \\
g_3 &= k_{31} \cdot s_1^2 + k_{32} \cdot s_6 + k_{33} & g_6 &= k_{61} \cdot s_1 \cdot s_6 + k_{62} \cdot c_5 + k_{63} \cdot s_1
\end{aligned}
$$

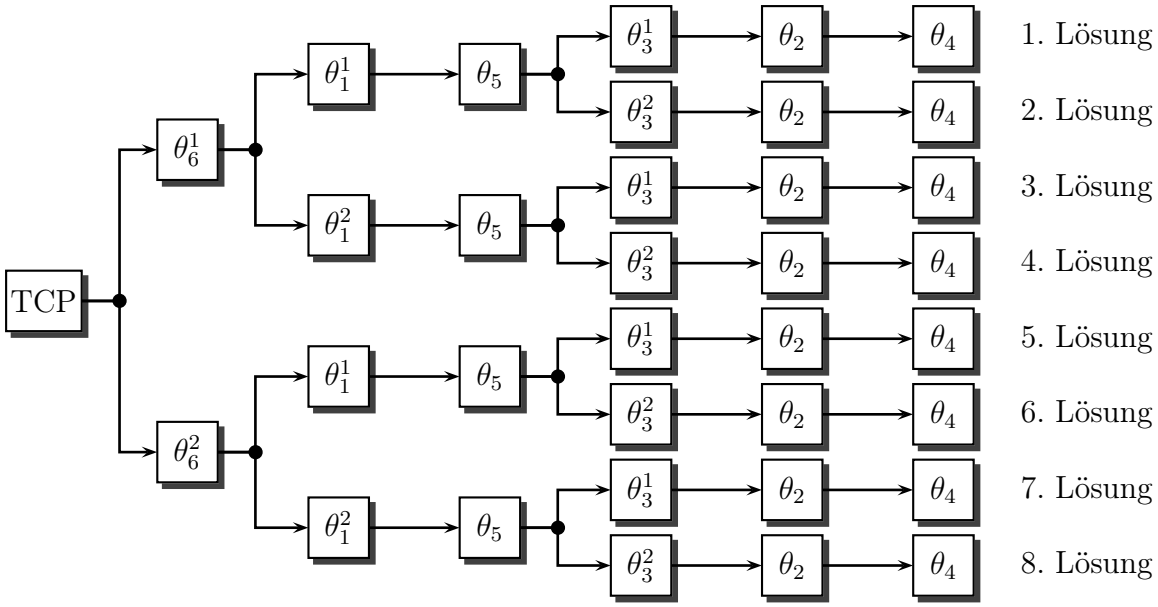
Diese Gleichungen haben dieselben Lösungen wie die ursprünglichen Kinematikgleichungen. Da diese Gleichungen *trianguliert* sind, können die Lösungen mittels Rücksubstitution ermittelt werden. Die erste Gleichung hängt nur von der Unbekannten s_6 ab. Die zweite enthält die Unbekannten c_6 und s_6. In den darauffolgenden Gleichungen kommt jeweils eine Unbekannte hinzu. Die Gleichung g_2 ist eine Prototypgleichung (Satz 4.5), die zwei Lösungen hat. Deswegen kann die inverse Kinematik geschlossen gelöst werden und für θ_6 gibt es zwei Lösungen. Die restlichen Unbekannten werden dann in Abhängigkeit von θ_6 durch Pattern Matching mit der Wissensbasis gelöst. Für θ_1 gibt es zwei Lösungen, für θ_5 eine Lösung, für θ_3 zwei Lösungen, für θ_2 eine Lösung und für θ_4 eine Lösung. Im allgemeinen gibt es für diesen Manipulator insgesamt acht Konfigurationen, um eine vorgegebene Pose im kartesischen Raum zu erreichen (Abbildung 4.6). Die Generierung einer Softwarekomponente schließt den Berechnungsvorgang für diesen Roboterarm ab.

Auf dieselbe Art wird der zweite Manipulatortyp gelöst. Die Gleichungen $\{f_1, f_2, f_3\}$ sind die in Polynomialdarstellung konvertierten kinematischen Gleichungen mit zwei Unbekannten und die zusätzlichen Gleichungen $\{f_4, f_5, f_6\}$ stellen die trigonometrischen Beziehungen bezüglich den neu eingeführten Variablen dar. Von diesen Gleichungen wird die Gröbnerbasis $GB = \{g_1, g_2, g_3, g_4, g_5, g_6\}$ mit der lexikographischen Ordnung $s_3 > c_3 > s_2 > c_2 > s_1 > c_1$ berechnet. Wichtig ist hierbei, mit den Variablen bezüglich θ_3 und nicht bezüglich θ_1 anzufangen. Andernfalls würde in der darauffolgenden Auswertung für θ_2 nicht eine Lösung ermittelt, sondern zwei Lösungen.

$$
\begin{aligned}
f_1 &= a_3 \cdot s_2 \cdot s_3 + d_4 \cdot s_2 \cdot c_3 + a_z \cdot d_6 - p_z & f_4 &= c_1^2 + s_1^2 - 1 \\
f_2 &= d_6 \cdot (a_y \cdot c_1 - a_x \cdot s_1) + p_x \cdot s_1 - p_y \cdot c_1 + a_3 \cdot c_3 - d_4 \cdot s_3 + d_2 & f_5 &= c_2^2 + s_2^2 - 1 \\
f_3 &= a_x \cdot d_6 \cdot c_1 \cdot s_2 - p_y \cdot s_1 \cdot s_2 - p_z \cdot c_2 + a_z \cdot d_6 \cdot c_2 - p_x \cdot c_1 \cdot s_2 & f_6 &= c_3^2 + s_3^2 - 1 \\
&\quad + a_y \cdot d_6 \cdot s_1 \cdot s_2
\end{aligned}
$$

Die Gleichung g_2 ist wiederum eine Prototypgleichung mit zwei Lösungen für θ_3. Insgesamt ergibt sich der folgende Auflösungsfluß für diesen Manipulator: zwei Lösungen für θ_3,

Abbildung 4.6 Auflösungsfluß bei der Berechnung der inversen Kinematik



zwei Lösungen für θ_1, eine Lösung für θ_2, zwei Lösungen für θ_4, eine Lösung für θ_6 und schließlich eine Lösung für θ_5. Falls sich die Gröbnerbasis zu $GB = \{1\}$ ergibt oder kein univariates Polynom enthält, ist es nicht möglich, eine geschlossene Lösung zu bestimmen. In diesem Fall wird eine numerische Approximation angewendet (Abschnitt 4.6.3).

$$g_1 = k_{11} \cdot s_3^2 + k_{12} \cdot s_3 \qquad g_4 = k_{41} \cdot c_2^2 + k_{42}$$
$$g_2 = k_{21} \cdot c_3 + k_{22} \cdot s_3 + k_{23} \qquad g_5 = k_{51} \cdot s_1 + k_{52} \cdot s_3 \cdot c_2 + k_{53} \cdot c_2 + k_{54}$$
$$g_3 = k_{31} \cdot s_2 + k_{32} \cdot s_3 + k_{33} \qquad g_6 = k_{61} \cdot c_1 + k_{62} \cdot s_3 \cdot c_2 + k_{63} \cdot c_2 + k_{64}$$

4.5 Geometrische Lösung der inversen Kinematik

Eine Alternative zu homogenen Koordinaten und homogenen 4×4 Transformationsmatrizen stellen sogenannte *Twist–Koordinaten* (engl.: Verwindung) dar, mit denen ebenfalls das kinematische Robotermodell vollautomatisch bestimmt wird. Durch diese diversitäre Entwicklung können sowohl die automatisch generierten Modelle verglichen und bewertet werden, als auch eine hohe Softwarequalität für die Bewegungssteuerung erzielt werden. Mit Hilfe von Twist–Koordinaten wird für eine serielle Kinematik genau dasselbe direkte kinematische Modell bestimmt, wie mit homogenen 4×4 Transformationsmatrizen (Abschnitt 4.2). Ein Vorteil des Verfahrens ist es, dass nicht jeder Gelenkachse ein lokales Koordinatensystem zugeordnet werden muß. Nur das Basis– und das Endeffektorkoordinatensystem des Roboters werden benötigt. Deswegen können Twist–Koordinaten sehr viel einfacher als Denavit–Hartenberg Parameter ermittelt werden. Brockett [11] zeigte, dass die Vorwärtstransformation einer kinematischen Kette durch die Berechnung der Exponenten von Twists und durch Multiplikation der sich hieraus ergebenden Matrizen bestimmt werden kann.

$$T_{0,n}(\vec{q}) = \left(\prod_{i=1}^{n} \exp(\hat{\xi}_i q_i) \right) * T_{0,n}(\vec{0}) \tag{4.8}$$

Die Transformationsmatrix $T_{0,n}(\vec{0})$ gibt die Transformation zwischen dem Basis– und dem Endeffektorkoordinatensystem an, wenn sich der Roboter in der Nullstellung befindet. Der Vektor $\vec{q} = (q_1, \ldots, q_n)^T$ gibt die Gelenkkonfiguration des Roboters wieder. Die Exponentialfunktion exp definiert eine surjektive Abbildung von der Lie–Algebra $se(3)$ nach der Lie–Gruppe $SE(3)$. Der Twist $\hat{\xi}_i$, der dem i–ten Gelenk zugeordnet wird, kann als Element von $se(3)$ geschrieben werden oder als sechsdimensionaler Vektor ξ_i, der auch als Twist–Koordinate bezeichnet wird (Abschnitt A.2).

$$\hat{\xi}_i = \begin{pmatrix} 0 & -\omega_3 & \omega_2 & \nu_1 \\ \omega_3 & 0 & -\omega_1 & \nu_2 \\ -\omega_2 & \omega_1 & 0 & \nu_3 \\ 0 & 0 & 0 & 0 \end{pmatrix} \longmapsto \xi_i = (\vec{\nu}_i, \vec{\omega}_i)^T \in \mathbb{R}^{6\times 1} \tag{4.9}$$

4.5.1 Bestimmung der Twist–Koordinaten eines Manipulators

Die systematische Berechnung der direkten Kinematik mit Hilfe von Twists wird in mehreren Schritten durchgeführt. Als erstes wird jeder Gelenkachse im Basiskoordinatensystem des Roboters ein Twist beziehungsweise Twist–Koordinate zugewiesen. Diese Zuweisung kann durch sehr einfache geometrische Überlegungen erfolgen. Für Translations– beziehungsweise Rotationsgelenke sind Twist–Koordinaten von der folgenden Form:

$$\xi_i^T = (\vec{\nu}_i, 0_{3\times 1})^T \qquad \xi_i^R = (-\vec{\omega}_i \times \vec{r}_i, \vec{\omega}_i)^T \tag{4.10}$$

Der Einheitsvektor $\vec{\nu}_i$ gibt bei Translationsgelenken die Richtung der Linearbewegung und der Einheitsvektor $\vec{\omega}_i$ bei Rotationsgelenken die Richtung der Rotationsachse an. Bei Rotationsgelenken zeigt der Vektor $\vec{r}_i$ vom Ursprung des Basiskoordinatensystems zu einem beliebig gewählten Punkt auf der Rotationsachse. Passiven Gelenkachsen und starren Verbindungen werden keine Twist–Koordinaten zugewiesen. Die homogene Matrix, die das Basiskoordinatensystem in das Endeffektorkoordinatensystem in der Nullstellung des Manipulators überführt, kann typischerweise durch Inspektion bestimmt werden. Alternativ können Twists auch anhand der Denavit–Hartenberg Parameter bestimmt werden. In diesem Fall werden die Twists durch folgende Matrixmultiplikationen ermittelt.

$$\hat{\xi}_i = M_1 * \ldots * M_{i-1} * P_i * (M_1 * \ldots * M_{i-1})^{-1} \tag{4.11}$$

Die Matrix M_i ist die i–te Denavit–Hartenberg Matrix (Gleichung 4.1), wobei die Gelenkvariable (θ_i beziehungsweise d_i) auf Null gesetzt ist. Die Matrix P_i hat für ein Translations– beziehungsweise Rotationsgelenk die folgende Gestalt:

$$P_i^T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad P_i^R = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.12}$$

Der nächste Schritt ist die Berechnung der Matrixexponenten der Twists. Im allgemeinen ist diese Berechnung sehr komplex, da die Exponentialfunktion als unendliche Reihe definiert ist und hier auf Matrizen angewendet wird. Im Falle von Starrkörperbewegungen konvergiert diese Reihe und es kann die folgende Formel verwendet werden:

$$\exp(\hat{\xi}_i q_i) = \exp\left(\begin{bmatrix} \tilde{\omega}_i & \vec{\nu}_i \\ 0_{3\times 1} & 0_{1\times 1} \end{bmatrix} q_i\right) = \begin{pmatrix} \exp(\tilde{\omega}_i q_i) & \vec{b}_i \\ 0_{3\times 1} & 1_{1\times 1} \end{pmatrix} \tag{4.13}$$

Für die Berechnung von Exponentialausdrücken der Form $\exp(\tilde{\omega}_i q_i)$, wird die Rotationsformel von Rodrigues (Gleichung A.14) [150] angewendet. Diese Formel wandelt eine schiefsymmetrische Einheitsmatrix $\tilde{\omega}_i$ (Kreuzproduktmatrix), die eine Rotationsachse beschreibt sowie einen Drehwinkel q_i in eine trigonometrische Rotationsmatrix um.

$$\exp(\tilde{\omega}_i q_i) = \begin{cases} I_{3\times3} + \tilde{\omega}_i \cdot \sin\theta_i + \tilde{\omega}_i^2 \cdot (1 - \cos\theta_i) & , q_i = \theta_i \\ I_{3\times3} & , q_i = d_i \end{cases} \tag{4.14}$$

$$\vec{b}_i = \begin{cases} (I_{3\times3} - \exp(\tilde{\omega}_i q_i)) \cdot (\vec{\omega}_i \times \vec{\nu}_i) & , q_i = \theta_i \\ d_i \cdot \vec{\nu}_i & , q_i = d_i \end{cases} \tag{4.15}$$

Nach Berechnung der Matrizen $\exp(\hat{\xi}_i q_i)$ für jedes Gelenk wird das direkte kinematische Problem durch Matrixmultiplikation nach Brockett (Gleichung 4.8) automatisch gelöst.

4.5.2 Bestimmung der Adjungierten von Twist–Koordinaten

Bei der Bestimmung der Vorwärtstransformation werden Twist–Koordinaten im Basiskoordinatensystem des Roboters angegeben. Manchmal ist es nötig, Twist–Koordinaten in ein anderes Koordinatensystem zu transformieren, beispielsweise bei der Bestimmung der Jacobi–Matrix mit Twist–Koordinaten [131]. Für eine homogene Transformationsmatrix $T \in SE(3)$ wird die *Adjungierte* Ad_T beziehungsweise *inverse Adjungierte* Ad_T^{-1} eingeführt, die Twist–Koordinaten zwischen unterschiedlichen Koordinatensystemen transformiert. Dabei bezeichnet R die Rotationsmatrix und $\vec{p}$ den Translationsvektor bezüglich der Transformation T und $\tilde{p}$ die durch $\vec{p}$ definierte antisymmetrische 3×3–Matrix.

$$Ad_T = \begin{pmatrix} R & \tilde{p} * R \\ 0 & R \end{pmatrix} \in \mathbb{R}^{6\times6} \qquad Ad_T^{-1} = \begin{pmatrix} R^T & -R^T * \tilde{p} \\ 0 & R^T \end{pmatrix} \in \mathbb{R}^{6\times6} \tag{4.16}$$

Für eine Twist–Koordinate ξ und eine homogene Transformation T gilt, dass $\xi' = Ad_T * \xi$ und $\xi'' = Ad_T^{-1} * \xi$ ebenfalls zwei Twist–Koordinaten sind mit $\exp(\hat{\xi}'q) = T * \exp(\hat{\xi}q) * T^{-1}$ und $\exp(\hat{\xi}''q) = T^{-1} * \exp(\hat{\xi}q) * T$.

4.5.3 Entwicklung einer Wissensbasis für Exponentialgleichungen

Die inverse Kinematik wird automatisch bestimmt, indem die Gleichungen der direkten Kinematik in eine Menge von einfacheren Gleichungen, die geschlossene Lösungen besitzen, transformiert werden. Diese kanonischen Teilprobleme basieren auf der Schraubentheorie [6] und resultieren aus geometrischen Überlegungen und algebraischen Umformungen. Die ersten drei Teilprobleme wurden von Kahan [83] und Paden [129, 130] eingeführt und geometrisch gelöst. Chen und Gao [23, 61] haben sich auch mit der geometrischen Lösung von Teilproblemen auseinandergesetzt, wobei sie die Lösungen unbekannter Gelenkvariablen durch Polynomfunktionen, die teilweise vom Grad 4 oder Grad 8 sind, ausdrückten. Mit ihren Lösungen bestimmten sie die inverse Kinematik von modularen Robotern ausschließlich numerisch. Im Gegensatz hierzu wird in dieser Arbeit die inverse Kinematik analytisch gelöst. Die Lösungen der Teilprobleme 1 sowie 3 bis 17 wurden im Rahmen dieser Arbeit unter Verwendung des Skalar–, Vektor– und Spatprodukts formal hergeleitet. Sämtliche Teilprobleme wurden in einer Wissensbasis abgespeichert.

Teilproblem 1: Rotation um eine einzelne Gelenkachse

Durch die Drehung einer Gelenkachse um den Winkel θ soll der Punkt $\vec{p}$ nach dem Punkt $\vec{q}$ rotiert werden. Der Gelenkachse ist die Twist–Koordinate $\xi = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ zugeordnet. Dann existiert eine eindeutige Lösung für den unbekannten Gelenkwinkel θ, um $\vec{p}$ nach $\vec{q}$ zu verschieben. Zunächst werden die beiden Hilfsvektoren $\vec{u} = \vec{p} - \vec{r}$ und $\vec{v} = \vec{q} - \vec{r}$ eingeführt. Als nächstes werden die beiden Vektoren auf die Ebene projeziert, die senkrecht zu $\vec{\omega}$ steht. Das Ergebnis sind die Vektoren $\vec{u}_2 = \vec{\omega} \times (\vec{u} \times \vec{\omega}) = \vec{u} \cdot (\vec{\omega}^T \cdot \vec{\omega}) - \vec{\omega} \cdot (\vec{\omega}^T \cdot \vec{u}) = \vec{u} - \vec{\omega} \cdot (\vec{\omega}^T \cdot \vec{u})$ und $\vec{v}_2 = \vec{\omega} \times (\vec{v} \times \vec{\omega}) = \vec{v} - \vec{\omega} \cdot (\vec{\omega}^T \cdot \vec{v})$. Der Winkel zwischen diesen beiden Vektoren ist die Lösung für θ. Mit dem Vektorprodukt wird der Sinus und mit dem Skalarprodukt der Cosinus von θ berechnet.

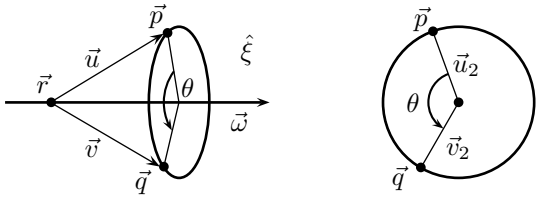
$$\exp(\hat{\xi}\theta) * \vec{p} = \vec{q} \quad \Longrightarrow \quad \vec{p} + \sin\theta \cdot (\vec{\omega} \times \vec{u}) - \vec{u}_2 + \cos\theta \cdot \vec{u}_2 = \vec{q} \tag{4.17}$$

$$\Longrightarrow \quad \sin\theta \cdot (\vec{\omega} \times \vec{u}_2) + \cos\theta \cdot \vec{u}_2 = \vec{v}_2 \tag{4.18}$$

$$\Longrightarrow \quad \sin\theta \cdot \vec{\omega} \cdot |\vec{u}_2|^2 = \vec{u}_2 \times \vec{v}_2 \ \wedge \ \cos\theta \cdot |\vec{u}_2|^2 = \vec{u}_2^T \cdot \vec{v}_2 \tag{4.19}$$

$$\Longrightarrow \quad \theta = \arctan_2(\vec{\omega}^T \cdot (\vec{u}_2 \times \vec{v}_2), \vec{u}_2^T \cdot \vec{v}_2) \tag{4.20}$$

Abbildung 4.7 Graphische Lösung des Teilproblems 1 (Paden–Kahan 1)



Teilproblem 2: Rotationen um zwei sich schneidende Gelenkachsen

Es soll ein Punkt $\vec{p}$ auf einen Punkt $\vec{q}$ verschoben werden, wobei zuerst mit dem Winkel θ_2 um die eine Gelenkachse gedreht wird und dann mit dem Winkel θ_1 um die andere Gelenkachse. Den beiden Gelenkachsen sind die Twist–Koordinaten $\xi_1 = (-\vec{\omega}_1 \times \vec{r}, \vec{\omega}_1)$ mit $|\vec{\omega}_1| = 1$ und $\xi_2 = (-\vec{\omega}_2 \times \vec{r}, \vec{\omega}_2)$ mit $|\vec{\omega}_2| = 1$ zugeordnet, wobei $\vec{r}$ der Schnittpunkt der beiden Gelenkachsen ist. Es gilt $|\omega_1 \times \omega_2| \neq 0$. Dieses Problem wird auf die Berechnung des Schnittpunkts zweier Kreise zurückgeführt. Es werden die Hilfsvektoren $\vec{u} = \vec{p} - \vec{r}$ und $\vec{v} = \vec{q} - \vec{r}$ sowie die Hilfsvariablen α, β und γ bestimmt.

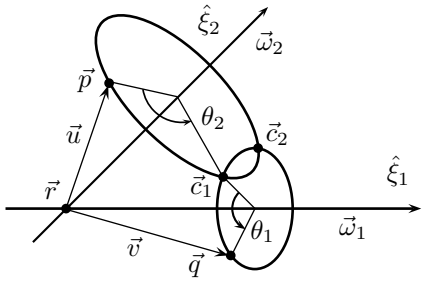
$$\alpha = \frac{(\vec{\omega}_1^T \cdot \vec{\omega}_2) \cdot (\omega_2^T \cdot \vec{u}) - (\vec{\omega}_1^T \cdot \vec{v})}{(\vec{\omega}_1^T \cdot \vec{\omega}_2)^2 - 1} \quad \beta = \frac{(\vec{\omega}_1^T \cdot \vec{\omega}_2) \cdot (\omega_1^T \cdot \vec{v}) - (\vec{\omega}_2^T \cdot \vec{u})}{(\vec{\omega}_1^T \cdot \vec{\omega}_2)^2 - 1} \tag{4.21}$$

$$\gamma^2 = \frac{|\vec{u}|^2 - \alpha^2 - \beta^2 - 2 \cdot \alpha \cdot \beta \cdot (\vec{\omega}_1^T \cdot \vec{\omega}_2)}{|\vec{\omega}_1 \times \vec{\omega}_2|^2} \tag{4.22}$$

Für den Schnittpunkt $\vec{c}$ der beiden Kreise kann es höchstens zwei Lösungen geben. Es gilt $\vec{c} = \vec{r} + \alpha \cdot \vec{\omega}_1 + \beta \cdot \vec{\omega}_2 \pm \gamma \cdot (\vec{\omega}_1 \times \vec{\omega}_2)$. Die beiden Lösungen für die unbekannten Gelenkwinkel θ_1 und θ_2 werden bestimmt, indem zweimal das Teilproblem 1 mit Hilfe von $\vec{c}$ gelöst wird.

$$\exp(\hat{\xi}_1\theta_1) * \exp(\hat{\xi}_2\theta_2) * \vec{p} = \vec{q} \quad \Longrightarrow \quad \exp(\hat{\xi}_1\theta_1) * \vec{c} = \vec{q} \quad \text{und} \quad \exp(\hat{\xi}_2\theta_2) * \vec{p} = \vec{c} \tag{4.23}$$

Abbildung 4.8 Graphische Lösung des Teilproblems 2 (Paden–Kahan 2)

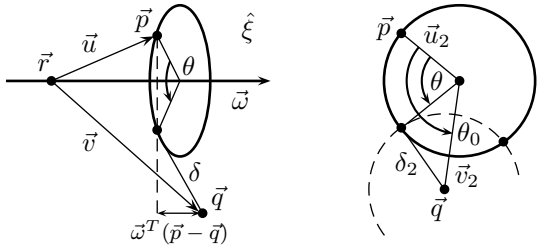


Teilproblem 3: Rotation um eine bestimmte Distanz

Es soll ein Punkt $\vec{p}$ um eine Rotationsachse mit dem Winkel θ gedreht werden, sodass nach der Drehung der Abstand zwischen den Punkten $\vec{p}$ und $\vec{q}$ dem Wert δ entspricht. Der Gelenkachse ist die Twist–Koordinate $\xi = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ zugeordnet. Die Vektoren $\vec{u}$, $\vec{v}$, $\vec{u}_2$ und $\vec{v}_2$ werden wie in Teilproblem 1 bestimmt. Der Winkel θ_0 ist der Winkel zwischen den Vektoren $\vec{u}_2$ und $\vec{v}_2$ und wird wie in Gleichung (4.20) bestimmt. Mit Hilfe der Variablen $\delta_2^2 = \delta^2 - (\omega^T \cdot (\vec{p} - \vec{q}))^2$ und durch Anwendung des Kosinussatzes werden die zwei Lösungen für den unbekannten Gelenkwinkel θ bestimmt.

$$
\begin{aligned}
|\exp(\hat{\xi}\theta) * \vec{p} - \vec{q}| = \delta \quad &\Longrightarrow \quad |\vec{p} - \vec{q}|^2 = |\vec{u}_2 - \vec{v}_2|^2 + (\omega^T \cdot (\vec{p} - \vec{q}))^2 && (4.24)\\
&\Longrightarrow \quad |\exp(\hat{\xi}\theta) * \vec{u}_2 - \vec{v}_2|^2 = \delta_2^2 && (4.25)\\
&\Longrightarrow \quad \sin^2\theta \cdot |\omega \times \vec{u}_2|^2 + \cos^2\theta \cdot |\vec{u}_2|^2 = |\vec{u}_2|^2 && (4.26)\\
&\Longrightarrow \quad (\omega \times \vec{u}_2) \cdot \vec{v}_2 = \cos(\theta_0 \pm \pi/2) \cdot |\vec{u}_2| \cdot |\vec{v}_2| && (4.27)\\
&\Longrightarrow \quad |\vec{u}_2|^2 - 2 \cdot \vec{u}_2^T \cdot \vec{v}_2 \cdot \cos(\theta \pm \theta_0) + |\vec{v}_2|^2 = \delta_2^2 && (4.28)\\
&\Longrightarrow \quad \theta^{(1,2)} = \theta_0 \pm \arccos\left(\frac{|\vec{u}_2|^2 + |\vec{v}_2|^2 - \delta_2^2}{2 \cdot \vec{u}_2^T \cdot \vec{v}_2}\right) && (4.29)
\end{aligned}
$$

Abbildung 4.9 Graphische Lösung des Teilproblems 3 (Paden–Kahan 3)



Teilproblem 4: Translation um eine bestimmte Distanz

Es soll ein Punkt $\vec{p}$ entlang einer Translationsachse mit dem Wert d verschoben werden, sodass nach der Verschiebung der Abstand zwischen den Punkten $\vec{p}$ und $\vec{q}$ dem Wert δ entspricht. Der Gelenkachse ist die Twist–Koordinate $\xi = (\vec{\nu}, \vec{0}_{3\times1})$ mit $|\vec{\nu}| = 1$ zugeordnet.

Für den unbekannten Translationsweg d gibt es höchstens zwei Lösungen.

$$|\exp(\hat{\xi}d) * \vec{p} - \vec{q}| = \delta \quad \Longrightarrow \quad |d \cdot \vec{\nu} + \vec{p} - \vec{q}| = \delta \tag{4.30}$$

$$\Longrightarrow \quad d^2 + 2 \cdot d \cdot (\vec{\nu}^T \cdot (\vec{p} - \vec{q})) + |\vec{p} - \vec{q}|^2 = \delta^2 \tag{4.31}$$

$$\Longrightarrow \quad d^{(1,2)} = -(\vec{\nu}^T \cdot (\vec{p} - \vec{q})) \pm \sqrt{\delta^2 - |\vec{p} - \vec{q}|^2 + (\vec{\nu}^T \cdot (\vec{p} - \vec{q}))^2} \tag{4.32}$$

Teilproblem 5: Translation eines Punkt–Vektors

Ein Punkt $\vec{p}$ soll entlang einer Translationsachse auf einen Punkt $\vec{q}$ verschoben werden. Die Translationsachse ist gegeben als $\xi = (\vec{\nu}, 0_{3\times1})$ mit $|\vec{\nu}| = 1$. Unter der Voraussetzung $|(\vec{q} - \vec{p}) \times \vec{\nu}| = 0$ gibt es eine eindeutige Lösung für den gesuchten Translationsweg d.

$$\exp(\hat{\xi}d) * \vec{p} = \vec{q} \quad \Longrightarrow \quad d \cdot \vec{\nu} = \vec{q} - \vec{p} \quad \Longrightarrow \quad d = (\vec{q} - \vec{p})^T \cdot \vec{\nu} \tag{4.33}$$

Teilproblem 6: Translation / Translation eines Punkt–Vektors

Ein Punkt $\vec{p}$ soll entlang zweier Translationsachsen auf einen Punkt $\vec{q}$ verschoben werden. Den Translationsachsen sind die Twist–Koordinaten $\xi_1 = (\vec{\nu}_1, 0_{3\times1})$ mit $|\vec{\nu}_1| = 1$ und $\xi_2 = (\vec{\nu}_2, 0_{3\times1})$ mit $|\vec{\nu}_2| = 1$ zugeordnet, wobei $|\nu_1 \times \nu_2| \neq 0$ und $(\vec{q} - \vec{p})^T \cdot (\nu_1 \times \nu_2) = 0$ gilt. Die Translationsachsen sind nicht parallel angeordnet. Dann gibt es eine eindeutige Lösung für die gesuchten Translationsvariablen d_1 und d_2.

$$\exp(\hat{\xi}_1 d_1) * \exp(\hat{\xi}_2 d_2) * \vec{p} = \vec{q} \quad \Longrightarrow \quad d_1 \cdot \vec{\nu}_1 + d_2 \cdot \vec{\nu}_2 = \vec{q} - \vec{p} \tag{4.34}$$

$$\Longrightarrow \quad d_1 = \frac{((\vec{q} - \vec{p}) \times \vec{\nu}_2)^T \cdot (\vec{\nu}_1 \times \vec{\nu}_2)}{|\vec{\nu}_1 \times \vec{\nu}_2|^2} \tag{4.35}$$

Nachdem d_1 bekannt ist, wird Teilproblem 5 verwendet, um d_2 zu bestimmen.

Teilproblem 7: Translation / Translation / Translation eines Punkt–Vektors

Ein Punkt $\vec{p}$ soll entlang dreier Translationsachsen auf einen Punkt $\vec{q}$ verschoben werden. Die Translationsachsen sind gegeben als $\xi_1 = (\vec{\nu}_1, 0_{3\times1})$ mit $|\vec{\nu}_1| = 1$, $\xi_2 = (\vec{\nu}_2, 0_{3\times1})$ mit $|\vec{\nu}_2| = 1$ und $\xi_3 = (\vec{\nu}_3, 0_{3\times1})$ mit $|\vec{\nu}_3| = 1$, wobei die Achsen paarweise nicht parallel sind. Es gibt eine eindeutige Lösung für die gesuchten Translationsvariablen d_1, d_2 und d_3.

$$\exp(\hat{\xi}_1 d_1) * \exp(\hat{\xi}_2 d_2) * \exp(\hat{\xi}_3 d_3) * \vec{p} = \vec{q} \quad \Longrightarrow \quad d_1 \cdot \vec{\nu}_1 + d_2 \cdot \vec{\nu}_2 + d_3 \cdot \vec{\nu}_3 = \vec{q} - \vec{p} \tag{4.36}$$

$$\Longrightarrow \quad d_1 = \frac{(\vec{q} - \vec{p})^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3)}{\vec{\nu}_1^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3)} \tag{4.37}$$

Nachdem d_1 bekannt ist, wird Teilproblem 6 verwendet, um d_2 und d_3 zu bestimmen.

Teilproblem 8: Translation / Rotation eines Punkt–Vektors

Zunächst wird ein Punkt $\vec{p}$ entlang einer Translationsachse mit dem Wert d verschoben und anschließend um eine Rotationsachse mit dem Winkel θ gedreht. Daraus ergibt sich der Zielpunkt $\vec{q}$. Der Translationsachse ist die Twist–Koordinate $\xi_2 = (\vec{\nu}, \vec{0}_{3\times1})$ mit $|\vec{\nu}| = 1$ und der Rotationsachse die Twist–Koordinate $\xi_1 = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ zugeordnet. Bei starren Körpern ist der Abstand zwischen einem Punkt $\vec{r}$, der auf der Rotationsachse liegt,

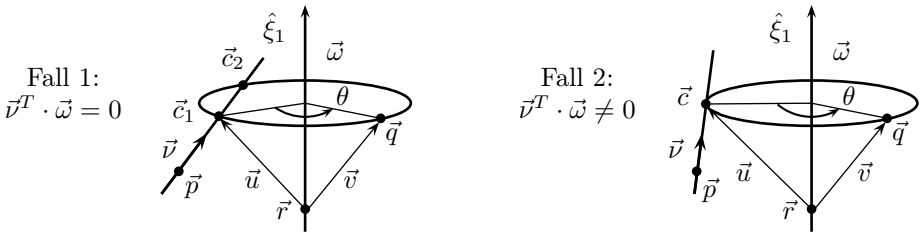
und dem Anfanspunkt der Rotation gleich dem Abstand zwischen $\vec{r}$ und dem Endpunkt der Rotation. Dadurch wird dieses Problem auf das Teilproblem 4 zurückgeführt.

$$\exp(\hat{\xi}_1\theta) * \exp(\hat{\xi}_2 d) * \vec{p} = \vec{q} \quad \Longrightarrow \quad d \cdot (\vec{\nu}^T \cdot \vec{\omega}) = (\vec{q} - \vec{p})^T \cdot \vec{\omega} \tag{4.38}$$

$$\Longrightarrow \quad |d \cdot \vec{\nu} + \vec{p} - \vec{r}| = |\vec{q} - \vec{r}| \tag{4.39}$$

Geometrisch betrachtet handelt es sich bei diesem Teilproblem um die Bestimmung der Schnittpunkte eines Kreises und einer Geraden. In der Lösung werden zwei Fälle unterschieden. Wenn die beiden Gelenkachsen rechtwinklig angeordnet sind, das heißt es gilt $\vec{\nu}^T \cdot \vec{\omega} = 0$, so gibt es höchstens zwei Lösungen für d, die anhand Gleichung (4.39) bestimmt werden. Ansonsten gibt es eine Lösung für den Translationsweg $d = ((\vec{q} - \vec{p})^T \cdot \vec{\omega})/(\vec{\nu}^T \cdot \omega)$. Anschließend wird das Teilproblem 1 verwendet, um θ zu bestimmen (Abbildung 4.10).

Abbildung 4.10 Graphische Lösung des Teilproblems 8



Teilproblem 9: Rotation / Translation eines Punkt–Vektors

Dieses Teilproblem ist dual zu Teilproblem 8. Zunächst wird ein Punkt $\vec{p}$ um eine Rotationsachse mit dem Winkel θ gedreht und anschließend entlang einer Translationsachse mit dem Wert d verschoben. Daraus ergibt sich der Zielpunkt $\vec{q}$. Der Rotationsachse ist die Twist–Koordinate $\xi_2 = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ und der Translationsachse ist die Twist–Koordinate $\xi_1 = (\vec{\nu}, \vec{0}_{3\times1})$ mit $|\vec{\nu}| = 1$ zugeordnet. Durch Umformung wird dieses Problem auf Teilproblem 8 zurückgeführt.

$$\exp(\hat{\xi}_1 d) * \exp(\hat{\xi}_2\theta) * \vec{p} = \vec{q} \quad \Longrightarrow \quad \exp(-\hat{\xi}_2\theta) * \exp(-\hat{\xi}_1 d) * \vec{q} = \vec{p} \tag{4.40}$$

Teilproblem 10: Rotation / Rotation eines Punkt–Vektors

Zunächst wird ein Punkt $\vec{p}$ um eine Gelenkachse mit dem Winkel θ_2 und anschließend um eine andere Gelenkachse mit dem Winkel θ_1 gedreht. Daraus ergibt sich der Zielpunkt $\vec{q}$. Den Gelenkachsen sind die Twist–Koordinaten $\xi_1 = (-\vec{\omega}_1 \times \vec{r}_1, \vec{\omega}_1)$ mit $|\vec{\omega}_1| = 1$ und $\xi_2 = (-\vec{\omega}_2 \times \vec{r}_2, \vec{\omega}_2)$ mit $|\vec{\omega}_2| = 1$ zugeordnet. Im Gegensatz zu Teilproblem 2 müssen sich die Gelenkachsen nicht schneiden. Die Gleichungen werden zuerst umgeformt.

$$\exp(\hat{\xi}_1\theta_1) * \exp(\hat{\xi}_2\theta_2) * \vec{p} = \vec{q} \quad \Longrightarrow \quad \exp(\hat{\xi}_2\theta_2) * \vec{p} = \exp(-\hat{\xi}_1\theta_1) * \vec{q} \tag{4.41}$$

$$\Longrightarrow \quad (\exp(\hat{\xi}_2\theta_2) * \vec{p})^T \cdot \vec{\omega}_1 = \vec{q}^T \cdot \vec{\omega}_1 \tag{4.42}$$

Bei der Lösung dieses Teilproblems wird unterschieden, ob die Gelenkachsen parallel oder nicht parallel zueinander angeordnet sind. Sind die beiden Rotationsachsen nicht parallel angeordnet ($|\vec{\omega}_1 \times \vec{\omega}_2| \neq 0$), so wird Gleichung (4.42) ausmultipliziert und dabei die Vektoren $\vec{u} = \vec{p} - \vec{r}_2$ und $\vec{u}_2 = \vec{u} - \vec{\omega}_2 \cdot (\vec{\omega}_2 \cdot \vec{u})$ verwendet. Die resultierende Gleichung

(4.43) ist in Satz 4.5 in Abschnitt 4.3.1 bereits gelöst worden. Deshalb gibt es für θ_2 zwei Lösungen. Nachdem θ_2 bekannt ist, wird Teilproblem 1 verwendet, um θ_1 zu bestimmen.

$$\cos\theta_2 \cdot \vec{u}_2^T \cdot \vec{\omega}_1 + \sin\theta_2 \cdot (\vec{\omega}_2 \times \vec{u})^T \cdot \vec{\omega}_1 = (\vec{q} - \vec{p} + \vec{u}_2)^T \cdot \vec{\omega}_1 \tag{4.43}$$

Falls die beiden Rotationsachsen parallel angeordnet sind ($\vec{\omega}_1 \cdot \vec{\omega}_2 = 1$), so wird Gleichung (4.41) ausmultipliziert. Das Ergebnis ist Gleichung (4.44).

$$\begin{aligned} &\sin(\theta_1+\theta_2)\cdot(\vec{\omega}_2\times\vec{u}) + \cos(\theta_1+\theta_2)\cdot\vec{u}_2 + \sin\theta_1\cdot\vec{\omega}_2\times(\vec{r}_2-\vec{r}_1) \\ &+(1-\cos\theta_1)\cdot\vec{\omega}_2\cdot(\vec{\omega}_2\cdot(\vec{p}-\vec{r}_1)) + \cos\theta_1\cdot(\vec{r}_2-\vec{r}_1+\vec{\omega}_2\cdot(\vec{\omega}_2\cdot\vec{u})) = \vec{v} \end{aligned} \tag{4.44}$$

Dabei wird der Vektor $\vec{v} = \vec{q} - \vec{r}_1$ verwendet. Durch Multiplikation dieser Gleichung mit dem Vektor $\vec{u}_2$ beziehungsweise $\vec{\omega}_2 \times \vec{u}$ ergibt sich Gleichung (4.45) beziehungsweise (4.46).

$$\cos(\theta_1+\theta_2)\cdot|\vec{u}_2|^2 - \sin\theta_1\cdot(\vec{r}_2-\vec{r}_1)\cdot(\vec{\omega}_2\times\vec{u}) + \cos\theta_1\cdot(\vec{r}_2-\vec{r}_1)\cdot\vec{u}_2 = \vec{v}\cdot\vec{u}_2 \tag{4.45}$$

$$\sin(\theta_1+\theta_2)\cdot|\vec{u}_2|^2 + \sin\theta_1\cdot(\vec{r}_2-\vec{r}_1)\cdot\vec{u}_2 + \cos\theta_1\cdot(\vec{r}_2-\vec{r}_1)\cdot(\vec{\omega}_2\times\vec{u}) = \vec{v}\cdot(\vec{\omega}_2\times\vec{u}) \tag{4.46}$$

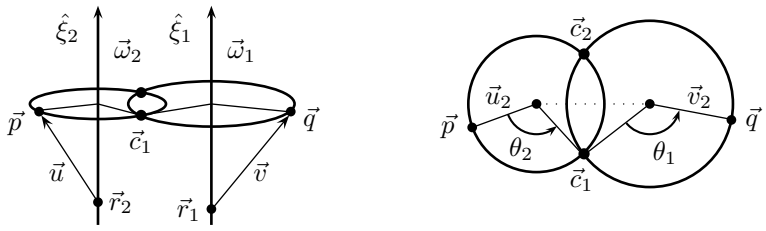
Gleichung (4.45) wird mit dem Term $a = (\vec{r}_2 - \vec{r}_1)\cdot\vec{u}_2$ und Gleichung (4.46) mit dem Term $b = (\vec{r}_2 - \vec{r}_1)\cdot(\vec{\omega}_2 \times \vec{u})$ multipliziert. Durch Addition der Gleichungen entsteht die Gleichung (4.47). Gleichung (4.45) wird mit dem Term $-b$ und Gleichung (4.46) mit dem Term a multipliziert. Durch Addition der Gleichungen entsteht die Gleichung (4.48). Bei den beiden Gleichungen (4.47) und (4.48) handelt es sich um zwei Gleichungstypen, die in Satz 4.9 in Abschnitt 4.3.1 bereits gelöst wurden. Somit ergeben sich zwei Lösungen für θ_2 und eine Lösung für θ_1 (Abbildung 4.11). Falls die beiden Rotationsachsen „zusammen fallen“, das heißt es gilt $|(\vec{r}_2 - \vec{r}_1) \times \vec{\omega}_2| = 0$, ist der Manipulator kinematisch redundant.

$$b\cdot|\vec{u}_2|^2\cdot\sin(\theta_1+\theta_2) + a\cdot|\vec{u}_2|^2\cdot\cos(\theta_1+\theta_2) + (a^2+b^2)\cdot\cos\theta_1 = d \tag{4.47}$$

$$a\cdot|\vec{u}_2|^2\cdot\sin(\theta_1+\theta_2) - b\cdot|\vec{u}_2|^2\cdot\cos(\theta_1+\theta_2) + (a^2+b^2)\cdot\sin\theta_1 = e \tag{4.48}$$

$$\text{mit} \quad d = a\cdot\vec{v}\cdot\vec{u}_2 + b\cdot\vec{v}\cdot(\vec{\omega}_2\times\vec{u}) \quad \text{und} \quad e = a\cdot\vec{v}\cdot(\vec{\omega}_2\times\vec{u}) - b\cdot\vec{v}\cdot\vec{u}_2$$

Abbildung 4.11 Graphische Lösung des Teilproblems 11 (Parallele Rotationsachsen)



Teilproblem 11: Translation / Rotation / Translation eines Punkt–Vektors

Zunächst wird ein Punkt $\vec{p}$ entlang einer Translationsachse mit dem Wert d_3 verschoben, dann um eine Rotationsachse mit dem Winkel θ gedreht und schließlich entlang einer anderen Translationsachse mit dem Wert d_1 verschoben. Daraus ergibt sich der Zielpunkt $\vec{q}$. Der ersten Translationsachse ist die Twist–Koordinate $\xi_3 = (\vec{\nu}_3, \vec{0}_{3\times1})$ mit $|\vec{\nu}_3| = 1$,

der Rotationsachse die Twist–Koordinate $\xi_2 = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ und der zweiten Translationsachse die Twist–Koordinate $\xi_1 = (\vec{\nu}_1, \vec{0}_{3\times1})$ mit $|\vec{\nu}_1| = 1$ zugeordnet. Die beiden Translationsachsen dürfen nicht parallel angeordnet sein, ansonsten ist der Manipulator kinematisch redundant. Die zu lösende Gleichung wird wie folgt umgeformt.

$$\exp(\hat{\xi}_1 d_1) * \exp(\hat{\xi}_2 \theta) * \exp(\hat{\xi}_3 d_3) * \vec{p} = \vec{q} \tag{4.49}$$

$$\Longrightarrow \quad \exp(\hat{\xi}_2 \theta) * (d_3 \cdot \vec{\nu}_3 + \vec{p}) = -d_1 \cdot \vec{\nu}_1 + \vec{q} \tag{4.50}$$

$$\Longrightarrow \quad d_1 \cdot (\vec{\nu}_1^T \cdot \vec{\omega}) + d_3 \cdot (\vec{\nu}_3^T \cdot \vec{\omega}) = (\vec{q} - \vec{p})^T \cdot \vec{\omega} \tag{4.51}$$

$$\Longrightarrow \quad |d_3 \cdot \vec{\nu}_3 + \vec{p} - \vec{r}| = |-d_1 \cdot \vec{\nu}_1 + \vec{q} - \vec{r}| \tag{4.52}$$

Je nachdem wie die Gelenkachsen angeordnet sind, werden bei der Lösung insgesamt vier Fälle unterschieden. Falls beide Translationsachsen zu der Rotationsachse rechtwinklig angeordnet sind, das heißt es gilt $\vec{\nu}_1^T \cdot \vec{\omega} = 0$ und $\vec{\nu}_3^T \cdot \vec{\omega} = 0$, so ist der Manipulator kinematisch redundant. Falls nur eine Translationsachse zu der Rotationsachse orthogonal steht, wird die Gelenkvariable der anderen Translationsachse direkt anhand Gleichung (4.51) bestimmt, das heißt $d_1 = ((\vec{q}-\vec{p})^T \cdot \vec{\omega})/(\vec{\nu}_1^T \cdot \vec{\omega})$ beziehungsweise $d_3 = ((\vec{q}-\vec{p})^T \cdot \vec{\omega})/(\vec{\nu}_3^T \cdot \vec{\omega})$. Die beiden anderen Gelenkvariablen werden dann mit den Teilproblemen 4 und 1 bestimmt.

Falls keine Translationsachse zu der Rotationsachse rechtwinklig angeordnet ist, wird Gleichung (4.51) nach d_3 aufgelöst. Die Variable θ wird dadurch aus Gleichung (4.50) eliminiert, dass bei einer Rotation der Abstand zwischen einem Punkt auf der Rotationsachse und dem zu drehenden Punkt erhalten bleibt. Durch Substitution von d_3 in der Abstandsgleichung (4.52) wird zunächst d_1 und anschließend d_3 und θ bestimmt.

Teilproblem 12: Translation / Translation / Rotation eines Punkt–Vektors

Dieses Problem ist ähnlich zu dem vorherigen Teilproblem. Es wird ein Punkt $\vec{p}$ entlang einer Translationsachse mit dem Wert d_3 und entlang einer weiteren Translationsachse mit dem Wert d_2 verschoben und dann um eine Rotationsachse mit dem Winkel θ gedreht. Daraus ergibt sich der Zielpunkt $\vec{q}$. Den beiden Translationsachsen sind die Twist–Koordinaten $\xi_3 = (\vec{\nu}_3, \vec{0}_{3\times1})$ mit $|\vec{\nu}_3| = 1$ und $\xi_2 = (\vec{\nu}_2, \vec{0}_{3\times1})$ mit $|\vec{\nu}_2| = 1$ und der Rotationsachse ist die Twist–Koordinate $\xi_1 = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ zugeordnet. Die Gleichung wird zuerst weiter umgeformt. Das Ergebnis sind die folgenden Gleichungen.

$$\exp(\hat{\xi}_1 \theta) * \exp(\hat{\xi}_2 d_2) * \exp(\hat{\xi}_3 d_3) * \vec{p} = \vec{q} \tag{4.53}$$

$$\Longrightarrow \quad d_2 \cdot \vec{\nu}_2 + d_3 \cdot \vec{\nu}_3 + \vec{p} = \exp(-\hat{\xi}_1 \theta) * \vec{q} \tag{4.54}$$

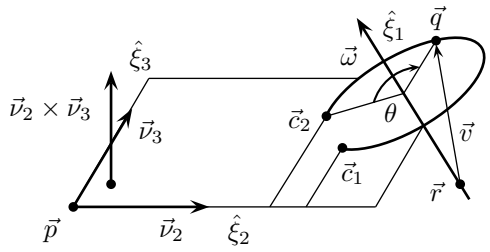
$$\Longrightarrow \quad \vec{p}^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3) = (\exp(-\hat{\xi}_1 \theta) * \vec{q})^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3) \tag{4.55}$$

Falls beide Translationsachsen zu der Rotationsachse senkrecht angeordnet sind, das heißt $\vec{\nu}_2^T \cdot \vec{\omega} = 0$ und $\vec{\nu}_3^T \cdot \vec{\omega} = 0$, ist der Manipulator kinematisch redundant. Anderfalls gibt es für dieses Teilproblem insgesamt zwei Lösungen (Abbildung 4.12). Gleichung (4.55) wird ausmultipliziert und dabei die Hilfsvektoren $\vec{v} = \vec{q} - \vec{r}$ und $\vec{v}_2 = \vec{v} - \vec{\omega} \cdot (\vec{\omega} \cdot \vec{v})$ verwendet. Das Ergebnis ist Gleichung (4.56) bei der es sich um eine Prototypgleichung handelt, deren zwei Lösungen bereits in Satz 4.5 in Abschnitt 4.3.1 bestimmt wurden.

$$\cos\theta \cdot \vec{v}_2^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3) - \sin\theta \cdot (\vec{\omega} \times \vec{v})^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3) = (\vec{p} - \vec{q} + \vec{v}_2)^T \cdot (\vec{\nu}_2 \times \vec{\nu}_3) \tag{4.56}$$

Geometrisch betrachtet wird durch die beiden Translationsachsen eine Ebene aufgespannt und deren Schnittpunkte mit der Kreisbahn des Rotationsgelenks ermittelt. Nachdem θ bekannt ist, werden die Lösungen von d_2 und d_3 mit Hilfe von Teilproblem 6 bestimmt.

Abbildung 4.12 Graphische Lösung des Teilproblems 12



Teilproblem 13: Rotation / Translation / Translation eines Punkt–Vektors

Dieses Problem ist dual zu dem vorherigen Problem. Es wird ein Punkt $\vec{p}$ um eine Rotationsachse mit dem Winkel θ gedreht und dann entlang einer Translationsachse mit dem Wert d_2 und entlang einer weiteren Translationsachse mit dem Wert d_1 verschoben. Daraus ergibt sich der Zielpunkt $\vec{q}$. Der Rotationsachse ist die Twist–Koordinate $\xi_3 = (-\vec{\omega} \times \vec{r}, \vec{\omega})$ mit $|\vec{\omega}| = 1$ und den Translationsachsen die Twist–Koordinaten $\xi_2 = (\vec{\nu}_2, \vec{0}_{3\times1})$ mit $|\vec{\nu}_2| = 1$ und $\xi_1 = (\vec{\nu}_1, \vec{0}_{3\times1})$ mit $|\vec{\nu}_1| = 1$ zugeordnet. Durch entsprechende Substitution wird die zu lösende Gleichung auf Teilproblem 12 zurückgeführt.

$$\exp(\hat{\xi}_1 d_1) * \exp(\hat{\xi}_2 d_2) * \exp(\hat{\xi}_3 \theta) * \vec{p} = \vec{q} \tag{4.57}$$

$$\Longrightarrow \quad \exp(-\hat{\xi}_3 \theta) * \exp(-\hat{\xi}_2 d_2) * \exp(-\hat{\xi}_1 d_1) * \vec{q} = \vec{p} \tag{4.58}$$

Teilproblem 14: Rotation / Translation / Rotation eines Punkt–Vektors

Zunächst wird ein Punkt $\vec{p}$ um eine Rotationsachse mit dem Winkel θ_3 gedreht, dann entlang einer Translationsachse mit dem Wert d verschoben und schließlich um eine andere Rotationsachse mit dem Winkel θ_1 gedreht. Daraus ergibt sich der Zielpunkt $\vec{q}$. Den beiden Rotationsachsen sind die Twist–Koordinaten $\xi_3 = (-\vec{\omega}_3 \times \vec{r}_3, \vec{\omega}_3)$ mit $|\vec{\omega}_3| = 1$ sowie $\xi_1 = (-\vec{\omega}_1 \times \vec{r}_1, \vec{\omega}_1)$ mit $|\vec{\omega}_1| = 1$ und der Translationsachse die Twist–Koordinate $\xi_2 = (\vec{\nu}, \vec{0}_{3\times1})$ mit $|\vec{\nu}| = 1$ zugeordnet. Die zu lösende Gleichung wird wie folgt umgeformt.

$$\exp(\hat{\xi}_1 \theta_1) * \exp(\hat{\xi}_2 d) * \exp(\hat{\xi}_3 \theta_3) * \vec{p} = \vec{q} \tag{4.59}$$

$$\Longrightarrow \quad d \cdot \vec{\nu} + \exp(\hat{\xi}_3 \theta_3) * \vec{p} = \exp(-\hat{\xi}_1 \theta_1) * \vec{q} \tag{4.60}$$

$$\Longrightarrow \quad d \cdot \vec{\nu}^T \cdot \vec{\omega}_1 + (\exp(\hat{\xi}_3 \theta_3) * \vec{p})^T \cdot \vec{\omega}_1 = \vec{q}^T \cdot \vec{\omega}_1 \tag{4.61}$$

$$\Longrightarrow \quad d \cdot \vec{\nu}^T \cdot \vec{\omega}_3 + \vec{p}^T \cdot \vec{\omega}_3 = (\exp(-\hat{\xi}_1 \theta_1) * \vec{q})^T \cdot \vec{\omega}_3 \tag{4.62}$$

Je nachdem wie die Gelenkachsen angeordnet sind, werden fünf Fälle unterschieden. Falls die beiden Rotationsachsen parallel zueinander stehen und die Translationsachse dazu senkrecht angeordnet ist ($\vec{\nu}^T \cdot \vec{\omega}_1 = 0$), ist der Manipulator kinematisch redundant. Falls die beiden Rotationsachsen parallel angeordnet sind und es gilt $\vec{\nu}^T \cdot \vec{\omega}_1 \neq 0$, so gibt es eine eindeutige Lösung für den Translationsweg $d = ((\vec{q} - \vec{p})^T \cdot \vec{\omega}_1)/(\vec{\nu}^T \cdot \vec{\omega}_1)$. Die Lösungen für θ_1 und θ_3 werden dann anhand Gleichung (4.60) mit dem Teilproblem 10 bestimmt.

Falls die beiden Rotationsachsen nicht parallel angeordnet sind, und es gilt $\vec{\nu}^T \cdot \vec{\omega}_1 = 0$, so werden zuerst die Lösungen für θ_3 bestimmt. Hierfür wird Gleichung (4.61) ausmultipliziert und dabei die Vektoren $\vec{u} = \vec{p} - \vec{r}_3$ und $\vec{u}_2 = \vec{u} - \vec{\omega}_3 \cdot (\vec{\omega}_3 \cdot \vec{u})$ verwendet. Bei der erhaltenen Gleichung (4.63) handelt es sich um einen Gleichungstyp, der in Satz 4.5

in Abschnitt 4.3.1 bereits gelöst wurde. Somit ergeben sich zwei Lösungen für θ_3. Die Lösungen von d und θ_1 werden anschließend mit Hilfe von Teilproblem 8 bestimmt.

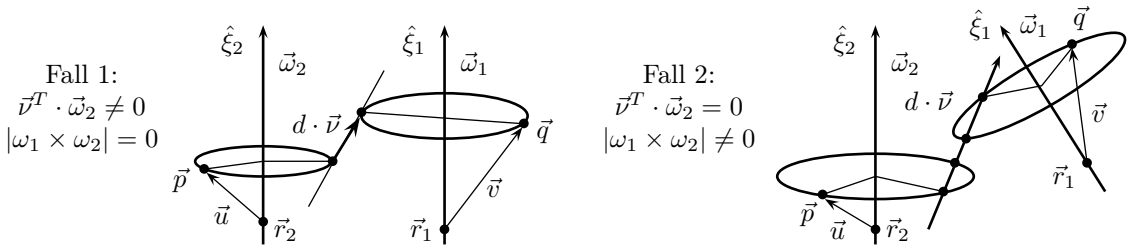
$$\cos\theta_3 \cdot \vec{u}_2^T \cdot \vec{\omega}_1 + \sin\theta_3 \cdot (\vec{\omega}_3 \times \vec{u})^T \cdot \vec{\omega}_1 = (\vec{q} - \vec{p} + \vec{u}_2)^T \cdot \vec{\omega}_1 \tag{4.63}$$

Falls die beiden Rotationsachsen nicht parallel angeordnet sind, und es gilt $\vec{\nu}^T \cdot \vec{\omega}_3 = 0$, so wird zunächst die Gleichung (4.62) ausmultipliziert. Dabei werden die Vektoren $\vec{v} = \vec{q} - \vec{r}_1$ und $\vec{v}_2 = \vec{v} - \vec{\omega}_1 \cdot (\vec{\omega}_1 \cdot \vec{v})$ verwendet. Das Ergebnis ist Gleichung (4.64), mit der die beiden Lösungen von θ_1 wiederum mit Satz 4.5 aus Abschnitt 4.3.1 ermittelt werden. Danach werden die Lösungen von d und θ_3 mit Hilfe von Teilproblem 9 bestimmt.

$$\cos\theta_1 \cdot \vec{v}_2^T \cdot \vec{\omega}_3 - \sin\theta_1 \cdot (\vec{\omega}_1 \times \vec{v})^T \cdot \vec{\omega}_3 = (\vec{p} - \vec{q} + \vec{v}_2)^T \cdot \vec{\omega}_3 \tag{4.64}$$

Geometrisch betrachtet handelt es sich bei den vorherigen beiden Fällen um die Bestimmung der beiden Schnittpunkte einer Ebene und eines Kreises (Abbildung 4.13). Die Ebene ist jeweils durch das Rotations– und Translationsgelenk, die senkrecht zueinander stehen, definiert. Der Spezialfall, dass die beiden Rotationsachsen nicht parallel angeordnet sind und weder $\vec{\nu}^T \cdot \vec{\omega}_1 = 0$ noch $\vec{\nu}^T \cdot \vec{\omega}_3 = 0$ gilt, führt zu sehr komplexen Lösungsgleichungen. Geometrisch gesehen werden hierbei die Schnittpunkte eines Kreises mit denen eines eventuell geneigten Zylinders bestimmt. Die Neigung des Zylinders hängt davon ab, ob die Translations– und eine der Rotationsachsen parallel zueinander sind oder nicht.

Abbildung 4.13 Graphische Lösung des Teilproblems 14



Teilproblem 15: Rotation / Rotation / Translation eines Punkt–Vektors

Zunächst wird ein Punkt $\vec{p}$ um eine Rotationsachse mit dem Winkel θ_3 gedreht, dann um eine weitere Rotationsachse mit dem Winkel θ_2 gedreht und schließlich entlang einer Translationsachse mit dem Wert d verschoben. Daraus ergibt sich der Zielpunkt $\vec{q}$. Den beiden Rotationsachsen sind die Twist–Koordinaten $\xi_3 = (-\vec{\omega}_3 \times \vec{r}_3, \vec{\omega}_3)$ mit $|\vec{\omega}_3| = 1$ sowie $\xi_2 = (-\vec{\omega}_2 \times \vec{r}_2, \vec{\omega}_2)$ mit $|\vec{\omega}_2| = 1$ und der Translationsachse die Twist–Koordinate $\xi_1 = (\vec{\nu}, \vec{0}_{3\times1})$ mit $|\vec{\nu}| = 1$ zugeordnet. Die zu lösende Gleichung wird wie folgt umgeformt.

$$\exp(\hat{\xi}_1 d) * \exp(\hat{\xi}_2 \theta_2) * \exp(\hat{\xi}_3 \theta_3) * \vec{p} = \vec{q} \tag{4.65}$$

$$\Longrightarrow \quad \exp(\hat{\xi}_3 \theta_3) * \vec{p} = \exp(-\hat{\xi}_2 \theta_2) * (-d \cdot \vec{\nu} + \vec{q}) \tag{4.66}$$

$$\Longrightarrow \quad d \cdot \vec{\nu}^T \cdot \vec{\omega}_2 + (\exp(\hat{\xi}_3 \theta_3) * \vec{p})^T \cdot \vec{\omega}_2 = \vec{q}^T \cdot \vec{\omega}_2 \tag{4.67}$$

$$\Longrightarrow \quad \vec{p}^T \cdot \vec{\omega}_3 = (\exp(-\hat{\xi}_2 \theta_2) * (-d \cdot \vec{\nu} + \vec{q}))^T \cdot \vec{\omega}_3 \tag{4.68}$$

Bei der Lösung dieses Teilproblems werden insgesamt fünf Fälle unterschieden. Falls die beiden Rotationsachsen parallel zueinander stehen und die Translationsachse dazu senkrecht angeordnet ist ($\vec{\nu}^T \cdot \vec{\omega}_2 = 0$), so ist der Manipulator kinematisch redundant. Falls

die beiden Rotationsachsen parallel angeordnet sind und es gilt $\vec{\nu}^T \cdot \vec{\omega}_2 \neq 0$, so gibt es eine eindeutige Lösung für den Translationsweg $d = ((\vec{q} - \vec{p})^T \cdot \vec{\omega}_2)/(\vec{\nu}^T \cdot \vec{\omega}_2)$. Die zwei Lösungen für θ_2 und θ_3 werden dann mit dem Teilproblem 10 bestimmt.

Falls die beiden Rotationsachsen nicht parallel angeordnet sind und es gilt $\vec{\nu}^T \cdot \vec{\omega}_2 = 0$, so werden zuerst die Lösungen für θ_3 bestimmt. Hierfür wird Gleichung (4.67) ausmultipliziert und dabei die Vektoren $\vec{u} = \vec{p} - \vec{r}_3$ und $\vec{u}_2 = \vec{u} - \vec{\omega}_3 \cdot (\vec{\omega}_3 \cdot \vec{u})$ verwendet. Die resultierende Gleichung (4.69) liefert zwei Lösungen für θ_3, die mit Satz 4.5 aus Abschnitt 4.3.1 ermittelt werden. Ein geometrische Betrachtung zeigt, dass es sich bei den beiden Lösungen um die Schnittpunkte einer Ebene und eines Kreises handelt. Die Ebene ist durch die Translations– und Rotationsachse, die rechtwinklig angeordnet sind, definiert. Die Lösungen von d und θ_2 werden anschließend mit Hilfe von Teilproblem 9 bestimmt.

$$\cos\theta_3 \cdot \vec{u}_2^T \cdot \vec{\omega}_2 + \sin\theta_3 \cdot (\vec{\omega}_3 \times \vec{u})^T \cdot \vec{\omega}_2 = (\vec{q} - \vec{p} + \vec{u}_2)^T \cdot \vec{\omega}_2 \tag{4.69}$$

Falls die beiden Rotationsachsen nicht parallel angeordnet sind und auch nicht $\vec{\nu}^T \cdot \vec{\omega}_2 = 0$ gilt, so wird unterschieden, ob die Translationsachse und die benachbarte Rotationsachse parallel zueinander sind oder es nicht sind. Beide Fälle führen zu sehr komplexen Lösungsgleichungen, wobei bei parallelen Gelenkachsen die Lösungen etwas einfacher sind.

Teilproblem 16: Translation / Rotation / Rotation eines Punkt–Vektors

Dieses Problem ist dual zu dem vorherigen Problem. Es wird ein Punkt $\vec{p}$ entlang einer Translationsachse mit dem Wert d verschoben, dann um eine Rotationsachse mit dem Winkel θ_2 gedreht und danach um eine weitere Rotationsachse mit dem Winkel θ_1 gedreht. Daraus ergibt sich der Zielpunkt $\vec{q}$. Der Translationsachse ist die Twist–Koordinate $\xi_3 = (\vec{\nu}, \vec{0}_{3\times1})$ mit $|\vec{\nu}| = 1$ und den Rotationsachsen die Twist–Koordinaten $\xi_1 = (-\vec{\omega}_1 \times \vec{r}_1, \vec{\omega}_1)$ mit $|\vec{\omega}_1| = 1$ und $\xi_2 = (-\vec{\omega}_2 \times \vec{r}_2, \vec{\omega}_2)$ mit $|\vec{\omega}_2| = 1$ zugeordnet. Durch entsprechende Substitution wird die zu lösende Gleichung auf Teilproblem 15 zurückgeführt.

$$\exp(\hat{\xi}_1\theta_1) * \exp(\hat{\xi}_2\theta_2) * \exp(\hat{\xi}_3 d) * \vec{p} = \vec{q} \tag{4.70}$$

$$\Longrightarrow \quad \exp(-\hat{\xi}_3 d) * \exp(-\hat{\xi}_2\theta_2) * \exp(-\hat{\xi}_1\theta_1) * \vec{q} = \vec{p} \tag{4.71}$$

Teilproblem 17: Rotation / Rotation / Rotation eines Punkt–Vektors

Zuerst wird ein Punkt $\vec{p}$ um eine Rotationsachse mit dem Winkel θ_3, dann um eine zweite Rotationsachse mit dem Winkel θ_2 und schließlich um eine dritte Rotationsachse mit dem Winkel θ_1 gedreht. Daraus ergibt sich der Zielpunkt $\vec{q}$. Den Rotationsachsen sind die Twist–Koordinaten $\xi_3 = (-\vec{\omega}_3 \times \vec{r}_3, \vec{\omega}_3)$ mit $|\vec{\omega}_3| = 1$, $\xi_2 = (-\vec{\omega}_2 \times \vec{r}_2, \vec{\omega}_2)$ mit $|\vec{\omega}_2| = 1$ und $\xi_1 = (-\vec{\omega}_1 \times \vec{r}_1, \vec{\omega}_1)$ mit $|\vec{\omega}_1| = 1$ zugeordnet. Die zu lösende Gleichung wird umgeformt.

$$\exp(\hat{\xi}_1\theta_1) * \exp(\hat{\xi}_2\theta_2) * \exp(\hat{\xi}_3\theta_3) * \vec{p} = \vec{q} \tag{4.72}$$

$$\Longrightarrow \quad \exp(\hat{\xi}_2\theta_2) * \exp(\hat{\xi}_3\theta_3) * \vec{p} = \exp(-\hat{\xi}_1\theta_1) * \vec{q} \tag{4.73}$$

$$\Longrightarrow \quad (\exp(\hat{\xi}_3\theta_3) * \vec{p})^T \cdot \vec{\omega}_2 = (\exp(-\hat{\xi}_1\theta_1) * \vec{q})^T \cdot \vec{\omega}_2 \tag{4.74}$$

Bei der Lösung werden die folgenden vier Fälle bezüglich der Anordnung der Gelenkachsen unterschieden. Falls die drei Rotationsachsen parallel zueinander angeordnet sind und weder $\vec{p}$ auf der ersten noch $\vec{q}$ auf der dritten Rotationsachse liegt, so ist der Manipulator kinematisch redundant. Als nächstes werden die beiden Fälle betrachtet, dass genau zwei Rotationsachsen parallel zueinander sind. Falls $|\vec{\omega}_1 \times \vec{\omega}_2| = 0$ gilt, so gibt es zwei Lösungen

für θ_3. Zur Bestimmung dieser Lösungen wird Gleichung (4.74) mit den Vektoren $\vec{u} = \vec{p} - \vec{r}_3$ und $\vec{u}_2 = \vec{u} - \vec{\omega}_3 \cdot (\vec{\omega}_3 \cdot \vec{u})$ umgeformt. Das Ergebnis ist Gleichung (4.75), bei der es sich um einen Gleichungstyp handelt, der in Satz 4.5 in Abschnitt 4.3.1 bereits gelöst wurde. Die Lösungen von θ_1 und θ_2 werden anschließend mit Hilfe von Teilproblem 10 bestimmt.

$$\cos\theta_3 \cdot \vec{u}_2^T \cdot \vec{\omega}_2 + \sin\theta_3 \cdot (\vec{\omega}_3 \times \vec{u})^T \cdot \vec{\omega}_2 = (\vec{q} - \vec{p} + \vec{u}_2)^T \cdot \vec{\omega}_2 \tag{4.75}$$

Falls $|\vec{\omega}_2 \times \vec{\omega}_3| = 0$ gilt, so gibt es zwei Lösungen für θ_1. Diese Lösungen werden durch Ausmultiplikation von Gleichung (4.74) bestimmt. Das Ergebnis ist die Gleichung (4.76), wobei die Vektoren $\vec{v} = \vec{q} - \vec{r}_1$ und $\vec{v}_2 = \vec{v} - \vec{\omega}_1 \cdot (\vec{\omega}_1 \cdot \vec{v})$ verwendet werden. Die zwei Lösungen für θ_1 sind geometrisch betrachtet die Schnittpunkte einer Ebene und eines Kreises. Die Ebene ist durch die zwei parallelen Rotationsachsen festgelegt. Danach werden die beiden Lösungen von θ_2 und θ_3 mit Hilfe von Teilproblem 10 bestimmt.

$$\cos\theta_1 \cdot \vec{v}_2^T \cdot \vec{\omega}_2 - \sin\theta_1 \cdot (\vec{\omega}_1 \times \vec{v})^T \cdot \vec{\omega}_2 = (\vec{p} - \vec{q} + \vec{v}_2)^T \cdot \vec{\omega}_2 \tag{4.76}$$

Der Spezialfall, dass überhaupt keine Gelenkachsen parallel angeordnet sind, führt zu sehr komplexen Lösungsgleichungen.

4.5.4 Dekomposition der direkten Kinematik

Die Bestimmung der inversen Kinematik mit Twist–Koordinaten erfolgt durch Dekomposition der direkten Kinematik in kleinere Teilprobleme. Schneiden sich aufeinanderfolgende Rotationsachsen eines Manipulators in einem Punkt, so existiert ein Fixpunkt, mit dem sich die Berechnungen vereinfachen lassen. Zur Bestimmung einer geschlossenen Lösung der Rückwärtstransformation werden deswegen zuerst die Schnittpunkte der Rotationsachsen ermittelt. Für zwei aufeinanderfolgende Twist–Koordinaten $\xi_i = (-\vec{\omega}_i \times \vec{r}_i, \vec{\omega}_i)$ mit $|\vec{\omega}_i| = 1$ und $\xi_{i+1} = (-\vec{\omega}_{i+1} \times \vec{r}_{i+1}, \vec{\omega}_{i+1})$ mit $|\vec{\omega}_{i+1}| = 1$ gibt es einen eindeutigen Schnittpunkt $\vec{p}$, falls die beiden Rotationsachsen nicht parallel zueinander sind, das heißt es gilt $|\vec{\omega}_{i+1} \times \vec{\omega}_i| \neq 0$, und falls die beiden Rotationsachsen in einer Ebene liegen, das heißt es gilt $(\vec{r}_i - \vec{r}_{i+1}) \cdot (\vec{\omega}_{i+1} \times \vec{\omega}_i) = 0$. Der Schnittpunkt $\vec{p}$ wird dann wie folgt berechnet:

$$\vec{p} = \vec{r}_{i+1} + \frac{((\vec{r}_i - \vec{r}_{i+1}) \times \vec{\omega}_i)^T \cdot (\vec{\omega}_{i+1} \times \vec{\omega}_i)}{|\vec{\omega}_{i+1} \times \vec{\omega}_i|^2} \cdot \vec{\omega}_{i+1} \tag{4.77}$$

Der Ausgangspunkt für die Berechnung der inversen Kinematik ist Gleichung (4.8), wobei die Transformationsmatrix $T_{0,n}(\vec{0})$, die die Nullstellung des Roboters angibt, auf die andere Seite gebracht wird. Liegt der zu rotierende Punkt $\vec{p}$ auf der Rotationsachse, so ändert eine Drehung um die gegebene Achse diesen Punkt-Vektor nicht. Es gilt $\exp(\hat{\xi}_i q_i) * \vec{p} = \vec{p}$ für jeden beliebigen Punkt $\vec{p}$ auf dieser Rotationsachse. Schneiden sich drei aufeinanderfolgende Rotationsachsen eines sechsachsigen beziehungsweise zwei aufeinanderfolgende Rotationsachsen eines fünfachsigen Roboters in dem Punkt $\vec{p}$, oder liegt der Punkt $\vec{p}$ auf genau einer Rotationsachse eines vierachsigen Roboters, so vereinfacht sich die Gleichung der direkten Kinematik stark. Liegt $\vec{p}$ auf den Achsen 4 bis n eines n–achsigen Roboters ($n = 4, \ldots, 6$), so lässt sich die direkte Kinematik zur Gleichung (4.79) vereinfachen, die mit den Teilproblemen 1 bis 17 analytisch gelöst wird.

$$\left(\prod_{i=1}^{3} \exp(\hat{\xi}_i q_i)\right) * \left(\prod_{i=4}^{n} \exp(\hat{\xi}_i \theta_i)\right) = T_{0,n}(\vec{q}) * T_{0,n}^{-1}(\vec{0}) \tag{4.78}$$

$$\Longrightarrow \quad \left(\prod_{i=1}^{3} \exp(\hat{\xi}_i q_i)\right) * \vec{p} = T_{0,n}(\vec{q}) * T_{0,n}^{-1}(\vec{0}) * \vec{p} \tag{4.79}$$

Nachdem q_1, q_2 und q_3 gelöst sind, wird Gleichung (4.78) nach den verbleibenden Unbekannten q_4 bis q_n aufgelöst. Diese Gelenkvariablen werden wiederum durch Anwendung der Teilprobleme 1 bis 17 analytisch gelöst. Die inverse Kinematik wird mit diesem Verfahren vollautomatisch bestimmt, wenn sich maximal drei Rotationsachsen in einem Punkt schneiden. Schneiden sich vier Rotationsachsen in einem Punkt, so ist der Manipulator kinematisch redundant und kann nur numerisch gelöst werden.

4.5.5 Experimentelle Ergebnisse

Für dieselben Roboter für die in Abschnitt 4.3.3 das kinematische Modell mit Hilfe von homogenen Transformationsmatrizen bestimmt wurde, wurde es auch mit Hilfe von Twist–Koordinaten bestimmt. Das direkte kinematische Modell, das bei beiden Verfahren generiert wird, ist jedesmal vollkommen identisch und darf deswegen als korrekt angenommen werden. Beim inversen kinematischen Modell unterscheiden sich teilweise die generierten analytischen Ausdrücke. Bei der numerischen Ausführung werden allerdings dieselben Werte berechnet. Die erforderlichen Operationen der generierten kinematischen Modelle auf Basis von Twist–Koordinaten sind in Tabelle 4.7 dargestellt.

Tabelle 4.7 Evaluation des kinematischen Robotermodells für ausgewählte Roboter

	Anzahl arithmetische Operatoren der direkten Kinematik								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	0	0	4	0	2	2	0	0	0
Scara I	8	5	4	0	4	4	0	0	0
Scara II	12	1	4	0	4	4	0	0	0
Zylindrischer Roboter	8	10	55	0	4	4	0	0	0
Stanford Arm	19	16	123	0	5	5	0	0	0
Gelenkarmroboter	54	18	128	0	21	22	0	0	0

	Anzahl arithmetische Operatoren der inversen Kinematik								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	0	0	0	0	0	0	2	0	0
Scara I	11	9	31	1	3	3	4	1	9
Scara II	12	7	35	1	2	2	4	1	9
Zylindrischer Roboter	12	15	56	0	3	3	5	2	3
Stanford Arm	27	17	129	0	4	4	7	3	6
Gelenkarmroboter	29	26	170	1	5	5	8	2	12

4.6 Jacobi–Matrix / Geschwindigkeitstransformation

Die inverse Kinematik kann weder mit der Denavit–Hartenberg Konvention noch mit Twist–Koordinaten für jeden beliebigen Robotertyp analytisch bestimmt werden. In diesen Fällen werden näherungsweise Verfahren, die auf der Berechnung der Jacobi–Matrix basieren, angewendet. Die *Jacobi–Matrix* beziehungsweise Funktional– oder Ableitungsmatrix ist nach dem Mathematiker *Carl Gustav Jacob Jacobi* (1804–1851) benannt. Sie beschreibt

den Zusammenhang zwischen den Gelenkgeschwindigkeiten und der Geschwindigkeit des Endeffektors und wird in dieser Arbeit für die Ermittlung singulärer Konfigurationen (Abschnitt 4.6.2), für die näherungsweise Rückwärtstransformation (Abschnitt 4.6.3) und für die Bestimmung des dynamischen Robotermodells nach Lagrange (Abschnitt 5.1) verwendet. Die Dimension der Jacobi–Matrix beträgt $6 \times n$, wobei n die Anzahl der Gelenkachsen des Roboters ist, das heißt jede Spalte entspricht einer Gelenkachse.

4.6.1 Bestimmung der Jacobi–Matrix

Die Jacobi–Matrix wird auch als Matrix der partiellen Ableitungen bezeichnet und kann auf unterschiedliche Arten bestimmt werden [126]. Sie kann beispielsweise durch Differentiation der Gleichungen des direkten kinematischen Modells bestimmt werden. Die drei Funktionen f_1, f_2 und f_3 berechnen die Position und die drei Funktionen f_4, f_5 und f_6 die Orientierung des Endeffektors in Abhängigkeit der Gelenkvariablen $q_1, \ldots, q_n$.

$$\mathcal{J} = \frac{d\mathcal{F}}{d\vec{q}} = \begin{pmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial q_1} & \frac{\partial f_6}{\partial q_2} & \cdots & \frac{\partial f_6}{\partial q_n} \end{pmatrix} \tag{4.80}$$

Die ersten drei Zeilen der Jacobi–Matrix können durch partielle Ableitung des Positionsvektors des direkten kinematischen Modells (Gleichung 4.2) bestimmt werden. Die Bestimmung der Zeilen vier bis sechs durch partielle Ableitung der Orientierung ist hingegen kompliziert, da die Orientierung des Endeffektors als Rotationsmatrix angegegeben ist und zunächst in Euler Winkel (Abschnitt A.1.1) umgewandelt werden muß. Dabei entstehen meistens Terme, die die Arcustangensfunktion enthalten. Da diese Funktion sehr aufwendig abzuleiten ist, wird die Jacobi–Matrix unter Verwendung der in Abschnitt 4.1 festgelegten Koordinatensysteme iterativ berechnet. Die i–te Spalte der Jacobi–Matrix hat für Translations– beziehungsweise Rotationsgelenke die folgende Gestalt [162]:

$$\mathcal{J}_i^T = \begin{pmatrix} \vec{z}_{i-1} \\ 0_{3\times 1} \end{pmatrix} \qquad \mathcal{J}_i^R = \begin{pmatrix} \vec{z}_{i-1} \times (\vec{p}_n - \vec{p}_{i-1}) \\ \vec{z}_{i-1} \end{pmatrix} \tag{4.81}$$

Der Vektor $\vec{p}_{i-1}$ entspricht dem Ursprung und der Vektor $\vec{z}_{i-1}$ der z–Achse des $(i-1)$–ten Koordinatensystems im Basiskoordinatensystem des Roboters. Der Vektor $\vec{p}_n$ gibt die Position des Endeffektors im Basiskoordinatensystem wieder.

Darüber hinaus wird die Jacobi–Matrix auch mit Hilfe von Twist–Koordinaten (Abschnitt 4.5.1) und deren Adjungierten (Abschnitt 4.5.2) bestimmt. Die Jacobi–Matrix kann entweder beginnend vom Basiskoordinatensystem oder beginnend vom Endeffektorkoordinatensystem hergeleitet werden. Im ersten Fall wird zunächst die Matrix $\mathcal{JS}$ (engl.: spatial manipulator Jacobian) und im zweiten Fall zuerst die Matrix $\mathcal{JB}$ (engl.: body manipulator Jacobian) bestimmt. Die i–te Spalte von $\mathcal{JS}$ hängt von den vorangehenden Gelenkkoordinaten $q_1, \ldots, q_{i-1}$ ab, während die i–te Spalte der Matrix $\mathcal{JB}$ von den nachfolgenden Gelenkkoordinaten $q_{i+1}, \ldots, q_n$ abhängt.

$$\mathcal{JS}_i = Ad_{\exp(\hat{\xi}_1 q_1)*\ldots*\exp(\hat{\xi}_{i-1} q_{i-1})} * \xi_i \qquad \mathcal{JB}_i = Ad^{-1}_{\exp(\hat{\xi}_i q_i)*\ldots*\exp(\hat{\xi}_n q_n)*T_{0,n}(\vec{0})} * \xi_i \tag{4.82}$$

Durch Multiplikation der inversen Adjungierten von der Transformatiosmatrix T_s und der Matrix $\mathcal{JS}$ beziehungsweise durch Multiplikation der Adjungierten von der Transformationsmatrix T_b und der Matrix $\mathcal{JB}$ wird die Jacobi–Matrix $\mathcal{J}$ bestimmt. Die Matrix T_s

berücksichtigt nur den Translationsanteil $\vec{p}_{dk}$ und die Matrix T_b nur den Rotationsanteil R_{dk} des direkten kinematischen Modells (Gleichung 4.8).

$$\mathcal{J} = Ad_{T_s}^{-1} * \mathcal{JS} \qquad \text{mit} \quad T_s = \begin{pmatrix} 1_{3\times3} & \vec{p}_{dk} \\ 0_{1\times3} & 1_{1\times1} \end{pmatrix} \tag{4.83}$$

$$\mathcal{J} = Ad_{T_b} * \mathcal{JB} \qquad \text{mit} \quad T_b = \begin{pmatrix} R_{dk} & 0_{3\times1} \\ 0_{1\times3} & 1_{1\times1} \end{pmatrix} \tag{4.84}$$

4.6.2 Durchführung einer Singularitätsanalyse

Singuläre Konfigurationen sind Gelenkstellungen bei denen der Roboter einen oder mehrere Freiheitsgrade im kartesischen Raum verliert. Die Erkennung und Vermeidung singulärer Konfigurationen ist eine weitere Hauptaufgabe bei der Steuerung von Robotern, da in singulären Konfigurationen die Beweglichkeit eines Manipulators eingeschränkt ist. Es werden zwei Arten von Singularitäten unterschieden.

- *Äußere Singularitäten* liegen am Rand des Arbeitsraums eines Roboters. Hier ist zum Beispiel der Arm voll ausgestreckt oder zusammmengefaltet und kann in einer Raumrichtung nicht bewegt werden.
- *Innere Singularitäten* treten im Inneren des Arbeitsraums auf. Hier liegen zum Beispiel zwei Gelenkachsen auf einer Linie und es gibt unendlich viele Gelenkeinstellungen, die zur gleichen Effektorlage führen. Die Drehung der einen Gelenkachse kann durch entsprechende Gegendrehung der anderen Gelenkachse kompensiert werden.

Werden die kartesischen Geschwindigkeiten eines Roboters über die inverse Jacobi–Matrix in Gelenkgeschwindigkeiten überführt, können sich in der Nähe von Singularitäten unendlich hohe Gelenkgeschwindigkeiten ergeben.

Mit Hilfe der Jacobi–Matrix wird automatisch eine Singularitätsanalyse durchgeführt. Bei sechsachsigen Robotern ist die Jacobi–Matrix in singulären Konfigurationen singulär, das heißt sie ist dann nicht mehr invertierbar. Zur Ermittlungen singulärer Konfigurationen werden bei sechsachsigen Robotern die Nullstellen der Jacobi–Determinante bestimmt. Hierbei wird die Jacobi–Matrix nach Möglichkeit in die folgende Blockgestalt gebracht, da es dann möglich ist, die Jacobi–Determinante durch zwei Unterdeterminaten auszudrücken, die nach der Sarrus–Regel bestimmt werden.

$$\det \begin{bmatrix} A_{3\times3} & 0_{3\times3} \\ B_{3\times3} & C_{3\times3} \end{bmatrix} = \det \left[\begin{pmatrix} A_{3\times3} & 0_{3\times3} \\ B_{3\times3} & 1_{3\times3} \end{pmatrix} * \begin{pmatrix} 1_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & C_{3\times3} \end{pmatrix} \right] = \det(A_{3\times3}) \cdot \det(C_{3\times3}) \tag{4.85}$$

4.6.3 Näherungsweise Rückwärtstransformation

Falls es nicht möglich ist, eine geschlossene Lösung der inversen Kinematik zu bestimmen, so werden numerische (iterative) Näherungsverfahren eingesetzt, die prinzipiell für alle Robotertypen mit beliebig vielen Freiheitsgraden verwendbar sind. Das Newton–Raphson Verfahren, benannt nach *Isaac Newton* (1643–1727) und *Joseph Raphson* (1648–1715), ist die am weitesten verbreitete numerische Methode, um die inverse Kinematik zu berechnen. Die Jacobi–Matrix beziehungsweise inverse Jacobi–Matrix ist Bestandteil dieses Verfahrens. Der größte Vorteil dieser Methode ist ihre quadratische Konvergenz, sofern der Anfangswert der Iteration nahe genug bei der Lösung liegt [127]. Zu den Nachteilen zählen der geringe Konvergenzradius und der hohe Rechenaufwand. Für jeden Bahnpunkt ist

eine erneute Iteration erforderlich und in jedem Iterationsschritt muß die inverse Jacobi–Matrix neu berechnet werden. Die Matrix kann nur dann invertiert werden, wenn sie quadratisch ist, das heißt wenn es sich um einen sechsachsigen Roboter handelt. Andernfalls wird die nach einem amerikanischen und einem englischen Mathematiker benannte *Moore–Penrose–Inverse* beziehungsweise *Pseudoinverse* $\mathcal{J}^+$ bestimmt [116, 136]. Für Roboter mit weniger als sechs Gelenkachsen wird die linke und für Roboter mit mehr als sechs Gelenkachsen die rechte Pseudoinverse bestimmt. Die Dimension der Matrix beträgt $n \times 6$.

$$\mathcal{J}^+ = \begin{cases} \mathcal{J}^T * (\mathcal{J} * \mathcal{J}^T)^{-1} & , n > 6 \\ (\mathcal{J}^T * \mathcal{J})^{-1} * \mathcal{J}^T & , n < 6 \end{cases} \tag{4.86}$$

Beim Newton–Raphson Verfahren müssen von dem Startpunkt P_0 sowohl die kartesischen $\vec{x}_0$ als auch die Gelenkkoordinaten $\vec{q}_0$ bekannt sein. Mit Hilfe der inversen Jacobi–Matrix werden dann die kartesischen Koordinaten $\vec{x}$ eines Punkts P in Gelenkkoordinaten $\vec{q}_k$ umgerechnet. Es wird die folgende Rekursionsformel verwendet:

$$\vec{q}_{k+1} = \vec{q}_k + \mathcal{J}^{-1}(\vec{q}_k) \cdot (\vec{x} - \vec{x}_k) \qquad \text{mit} \quad \vec{x}_k = f(\vec{q}_k) \tag{4.87}$$

Damit sich die einzelnen Fehler nicht addieren, wird nach jedem Iterationsschritt mit der Funktion f eine Vorwärtstransformation durchgeführt. Mit dem Vektor $\vec{x}_k$ wird dann weiter gerechnet. Die Iteration wird beendet, falls die Differenz zwischen der gegebenen kartesischen Lage und der iterierten Lösung unter eine Schwelle sinkt: $|\vec{x} - \vec{x}_k| < \varepsilon$. Um „endlose“ Iterationen zu verhindern, wird die Anzahl der Iterationsschritte gezählt. Ein Abbruch erfolgt auch dann, wenn die geforderte Genauigkeit nach einer bestimmten Anzahl von Iterationszyklen noch nicht erreicht wurde. Die Iteration darf jedoch nicht zu früh abgebrochen werden.

4.6.4 Experimentelle Ergebnisse

Für dieselben Robotertypen für die in Abschnitt 4.3.3 das kinematische Modell aufgestellt wurde, wurde ebenfalls die Jacobi–Matrix in analytischer Form ermittelt (Tabelle 4.8). Die Jacobi–Matrix wurde sowohl mit Hilfe von homogenen Koordinaten (Gleichung 4.81) als auch mit Hilfe von Twist–Koordinaten (Gleichung 4.84) bestimmt. Die beiden generierten Jacobi–Matrizen für den jeweiligen Robotertyp wurden miteinander verglichen. Da sie jedes Mal völlig übereinstimmen, dürfen diese komplizierten Matrizen als korrekt angenommen werden. Dieses diversitäre Vorgehen garantiert eine hohe Softwarequalität.

Tabelle 4.8 Evaluation der Jacobi–Matrix für verschiedene Roboter

	Anzahl arithmetische Operatoren der Jacobi–Matrix								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	0	0	0	0	1	1	0	0	0
Scara I	5	1	6	0	3	3	0	0	0
Scara II	5	1	6	0	3	3	0	0	0
Zylindrischer Roboter	4	4	41	0	3	3	0	0	0
Stanford Arm	9	19	120	0	4	4	0	0	0
Gelenkarmroboter	55	29	151	0	26	27	0	0	0

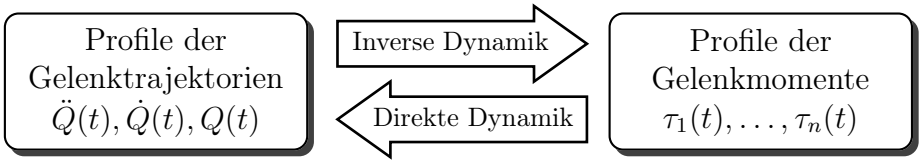
Kapitel 5

Bestimmung des dynamischen Robotermodells

In diesem Kapitel wird beschrieben, wie die Herleitung des dynamischen Robotermodells anhand einer Beschreibung der mechanischen Struktur des Roboters vollautomatisch durchgeführt wird. Hierbei handelt es sich um ein sehr komplexes Problem, da das dynamische Modell aus partiellen Differentialgleichungen mit großen Nichtlinearitäten besteht. Für die meisten Robotertypen läßt sich diese Herleitung nicht mehr fehlerfrei von Hand bewerkstelligen, da sie sehr komplex und mit hohem Aufwand verbunden ist. Dynamik (griech.: dynamis = Kraft) ist die mathematische Beschreibung der Bewegung von Körpern unter der Einwirkung von Kräften. Zur Berechnung des dynamischen Robotermodells werden verschiedene Berechnungsverfahren angewendet, und zwar das *Lagrange Verfahren* (Abschnitt 5.1) und das *rekursive Newton–Euler Verfahren* (Abschnitt 5.2).

Das dynamische Modell berechnet den Zusammenhang zwischen Position, Geschwindigkeit und Beschleunigung auf der einen Seite und den Antriebskräften und Momenten, die zur Durchführung der Bewegung erforderlich sind, auf der anderen Seite [161, 170, 179]. Es werden dabei sowohl externe Kräfte wie zum Beispiel Gravitation als auch interne Parameter wie zum Beispiel konzentrierte Massen der einzelnen Manipulatorglieder berücksichtigt. Ähnlich wie beim kinematischen Robotermodell gibt es beim dynamischen Robotermodell auch ein direktes und ein inverses Problem (Abbildung 5.1).

Abbildung 5.1 Direkte und inverse Dynamik



Die *inverse Dynamik* beziehungsweise *Rückwärtsdynamik* berechnet die in den Gelenkachsen auftretenden Kräfte und Momente in Abhängigkeit der Lage, der Geschwindigkeit und der Beschleunigung der einzelnen Achsen. Die *direkte Dynamik* beziehungsweise *Vorwärtsdynamik* berechnet die Bewegung eines Manipulators in Abhängigkeit von intern oder extern wirkenden Kräften und Momenten. Das direkte Modell erhält als Eingabe die aktuellen Gelenkwerte und Gelenkgeschwindigkeiten sowie die momentan wirkenden Kräfte und Momente und liefert als Ausgabe die resultierenden Gelenkbeschleunigungen.

Durch Integration der berechneten Beschleunigungen nach der Zeit werden die Geschwindigkeiten und durch Integration der Geschwindigkeiten nach der Zeit werden die Positionen der Gelenkachsen bestimmt. Die kartesischen Koordinaten des Endeffektors können dann mit der direkten Kinematik bestimmt werden. Die inverse Dynamik ist wichtig für den modellbasierten Reglerentwurf. Durch die Einbeziehung der inversen Dynamik in die Berechnung der Sollwerte für die Antriebe kann die Positioniergenauigkeit eines Manipulators verbessert werden. Die direkte Dynamik ist wichtig für die Simulation und für die Neuentwicklung von Robotern sowie für die Auswahl von geeigneten Antrieben.

Die Roboterdynamik kann numerisch oder analytisch bestimmt werden. Bei der Ermittlung einer Matrixdarstellung werden die Massen–, Coriolis– und Zentrifugalmatrix sowie der Gravitationsvektor explizit berechnet. Das analytische Aufstellen der Bewegungsgleichungen in Form eines Differentialgleichungssystems hat den Vorteil, dass diese Gleichungen sehr leicht nach den Beschleunigungsgrößen aufgelöst werden können. Somit kann die direkte Dynamik durch Invertierung der Massenmatrix und Umstellung der Gleichungen einfach berechnet werden. Die Komplexität der Gleichungen hängt stark von der Zahl der Freiheitsgrade ab. Deswegen eignet sich eine analytische Lösung nicht für jeden Robotertyp, insbesondere wenn er sehr viele Freiheitsgrade hat.

5.1 Lagrange Verfahren

Joseph Louis Lagrange (1736–1813) war ein französischer Mathematiker italienischer Herkunft, der allgemeine Gleichungen für die Bewegungen dynamischer Systeme unter Betrachtung von Energiebilanzen einführte. Ausgangspunkt ist die Lagrange–Funktion $\mathcal{L}$, die sich bei Starrkörpersystemen aus der Differenz zwischen der kinetischen Energie $\mathcal{K}$ (Bewegungsenergie) und der potentiellen Energie $\mathcal{P}$ (Lageenergie) ergibt. Die Lagrange–Gleichungen (Gleichung 5.1) ergeben sich durch Differentiation der Lagrange–Funktion nach der Zeit t sowie nach sogenannten *verallgemeinerten Koordinaten* q_i und deren zeitlichen Ableitungen $\dot{q}_i$. Bei Rotationsgelenken handelt es sich bei den Größen q_i um Drehwinkel und bei Translationsgelenken um Verfahrwege. Die Größen τ_i stellen bei Rotationsgelenken Momente und bei Translationsgelenken Kräfte dar.

$$\tau_i = \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \frac{d}{dt}\left(\frac{\partial \mathcal{K}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{K}}{\partial q_i} + \frac{\partial \mathcal{P}}{\partial q_i} \qquad i = 1, \ldots, n \tag{5.1}$$

Beim Lagrange–Verfahren werden die Bewegungsgleichungen eines Manipulators in Form von n gekoppelten nichtlinearen Differentialgleichungen 2. Ordnung aufgestellt. Jeder Freiheitsgrad ergibt eine Gleichung. Die mathematische Herleitung erfolgt durch die analytische Umformung der Lagrange Funktion und deren partiellen Differenzierung.

5.1.1 Automatische Erzeugung der Bewegungsgleichungen

Für das Aufstellen der Bewegungsgleichungen werden die Massen m_i, Massenverteilungen $I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}$ und Schwerpunktslagen $\vec{r}_i$ der Manipulatorglieder in den Konfigurator (Abschnitt 6.2) eingegeben, der in dieser Arbeit entwickelt wurde. Dabei wird für jedes Glied angenommen, dass die Masse jeweils im Schwerpunkt des Glieds konzentriert ist. Anhand der Benutzerbeschreibung wird zunächst die kinetische und potentielle Energie der einzelnen Glieder bestimmt. Dazu wird die *Trägheitstensormatrix* aufgestellt, die beschreibt wie die Masse eines Gliedes hinsichtlich eines Referenzkoordinatensystems (hier

A) verteilt ist. Sie ist eine symmetrische 3×3 Matrix, die durch drei Trägheitsmomente in der Hauptdiagonalen und drei Deviationsmomente außerhalb der Hauptdiagonalen charakterisiert ist. Diese sechs Elemente sind voneinander unabhängig.

$$ {}^{A}\mathcal{I} = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{pmatrix} \tag{5.2} $$

In der Praxis wird die Trägheitstensormatrix üblicherweise im Hinblick auf den Massenschwerpunkt bestimmt. Hierauf basierend wird die *verallgemeinerte Massenmatrix* M_i für das i–te Glied definiert, wobei m_i die Masse des Glieds und $I_{3\times3}$ die 3×3 Einheitsmatrix ist. Diese zeitinvariante Matrix wird zur Berechnung der kinetischen Energien verwendet.

$$ M_i = \begin{pmatrix} m_i \cdot I_{3\times3} & 0 \\ 0 & {}^{A}\mathcal{I} \end{pmatrix}_{6\times6} \tag{5.3} $$

Durch Aufsummierung der kinetischen Energien der einzelnen Glieder wird die kinetische Energie des gesamten Manipulators bestimmt. Die kinetische Energie eines Manipulators setzt sich aus der kinetischen Translations– und der kinetischen Rotationsenergie zusammen. Sie wird durch Gleichung (5.4) bestimmt. Die kinetische Energie eines Systems ist ein quadratischer Term und deshalb positiv, soforn das System sich nicht im Ruhezustand befindet. Die Matrix $\mathcal{J}_i$ ist die i–te partielle Jacobi–Matrix (Dimension $6 \times n$), die die Gelenkvariablen $q_1, \ldots, q_i$ auf die Trägheitsgeschwindigkeit des Schwerpunkts des i–ten Gliedes abbilden. Die Transformationsmatrix $T_{0,i}$ beschreibt das i–te Koordinatensystem relativ zum Basiskoordinatensystem des Roboters. Der Vektor $\vec{r}_i = (rx_i, ry_i, rz_i)^T$ gibt die Position des Schwerpunkts des i–ten Gliedes im i–ten Koordinatensystem an.

$$ \mathcal{K} = \frac{1}{2}\dot{Q} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathcal{J}_i \left[T_{0,i} * \begin{pmatrix} I_{3\times3} & \vec{r}_i \\ 0 & 1 \end{pmatrix} \right] \right)^T * M_i * \left(\mathcal{J}_i \left[T_{0,i} * \begin{pmatrix} I_{3\times3} & \vec{r}_i \\ 0 & 1 \end{pmatrix} \right] \right)}_{\mathcal{M}(Q)} \right\} \dot{Q} \tag{5.4} $$

Die potentielle Energie des Manipulators ergibt sich als Summe der potentiellen Energien der einzelnen Glieder. Die potentielle Energie jedes Glieds hängt von der jeweiligen Masse und den jeweiligen Schwerpunktkoordinaten ab. Durch Aufsummierung ergibt sich die potentielle Gesamtenergie als Funktion der Gelenkvariablen. Der Gravitätsvektor $\vec{g} = (g_x, g_y, g_z, 0)^T$ wird im Basiskoordinatensystem des Roboters angegeben. Für die meisten Roboter, die ortsfest am Boden verankert sind, ist $\vec{g} = (0, 0, -g, 0)^T$. Die Gravitationsbeschleunigung g ist ortsabhängig. In Meereshöhe, auf 45° geographischer Breite beträgt die Normalfallbeschleunigung im Mittel $g = 9,80665\ m/s^2$ (DIN 1305).

$$ \mathcal{P} = \sum_{i=1}^{n} -m_i \cdot (g_x, g_y, g_z, 0) * T_{0,i} * \begin{pmatrix} \vec{r}_i \\ 1 \end{pmatrix} \tag{5.5} $$

Das Zusammenfügen dieser Gleichungen führt zu einer vollständigen Darstellung der dynamischen Gleichungen eines Manipulators.

$$ \begin{pmatrix} \tau_1 \\ \vdots \\ \tau_n \end{pmatrix} = \mathcal{M}(Q) * \underbrace{\ddot{Q}}_{\frac{\partial \dot{Q}}{\partial t}} + \underbrace{\underbrace{\left(\sum_{i=1}^{n} \frac{\partial \mathcal{M}(Q)}{\partial q_i} \dot{q}_i \right)}_{\frac{\mathcal{M}(Q)}{\partial t}} * \dot{Q} - \frac{1}{2} \begin{pmatrix} \dot{Q}^T \frac{\mathcal{M}(Q)}{\partial q_1} \dot{Q} \\ \vdots \\ \dot{Q}^T \frac{\mathcal{M}(Q)}{\partial q_n} \dot{Q} \end{pmatrix}}_{\mathcal{C}(Q,\dot{Q}) * \dot{Q}} + \underbrace{\begin{pmatrix} \frac{\mathcal{P}}{\partial q_1} \\ \vdots \\ \frac{\mathcal{P}}{\partial q_n} \end{pmatrix}}_{\mathcal{G}(Q)} \tag{5.6} $$

Die einzelnen Matrizen und Vektoren haben folgende Bedeutungen und Eigenschaften.

- Die Matrix $\mathcal{M}(Q)$ wird als *Massenmatrix* beziehungsweise *Trägheitsmatrix* eines Manipulators bezeichnet. Diese Matrix ist symmetrisch und positiv definit und deshalb immer invertierbar. Die Dimension dieser quadratischen Matrix ist gleich $n \times n$. Die Massenmatrix beschreibt die Massenverteilung und somit das Trägheitsverhalten des Roboters sowohl für translatorische als auch für rotatorische Bewegungen.

- Die Matrix $\mathcal{C}(Q, \dot{Q})$ wird als *Matrix der Zentrifugal– und Corioliskräfte* bezeichnet. Die Dimension dieser quadratischen Matrix beträgt $n \times n$.

- Der Vektor $\mathcal{G}(Q)$ wird als *Gravitationsvektor* bezeichnet. Er berechnet die aus der Erdanziehung resultierende Kraft, die senkrecht in Richtung zur Erdoberfläche an den Manipulatorgliedern wirkt. Die Dimension des Vektors ist $n \times 1$.

Die Matrix $\mathcal{M}(Q)$ und der Vektor $\mathcal{G}(Q)$ hängen nur von Gelenkvariablen ab, während die Matrix $\mathcal{C}(Q, \dot{Q})$ sowohl von den Gelenkvariablen als auch von den Gelenkgeschwindigkeiten abhängt. Die Matrix $\mathcal{C}(Q, \dot{Q})$ kann in die zwei Matrizen $\mathcal{CO}(Q)$ und $\mathcal{CF}(Q)$ umgeformt werden, die dann nur noch von den Gelenkvariablen abhängen. Die einzelnen Elemente dieser beiden Matrizen werden durch Differentiation der Massenmatrixelemente bestimmt. Für die Bestimmung des (i, j)–ten Elements der Coriolismatrix werden zwei Hilfsvariablen k und l eingeführt. Die Hilfsvariable k wird von 1 bis n und die Variable l von $k+1$ bis n iteriert. Der Wert von j ergibt sich dann als $l + (k-1) \cdot n - k \cdot (k+1)/2$.

$$\mathcal{CO}[i,j] = \frac{1}{2} \cdot \left(\frac{\partial}{\partial q_l} \mathcal{M}[i,k] + \frac{\partial}{\partial q_k} \mathcal{M}[i,l] - \frac{\partial}{\partial q_i} \mathcal{M}[k,l] \right) \tag{5.7}$$

$$\mathcal{CF}[i,j] = \frac{\partial}{\partial q_j} \mathcal{M}[i,j] - \frac{1}{2} \cdot \frac{\partial}{\partial q_i} \mathcal{M}[j,j] \tag{5.8}$$

Mit diesen Matrizen wird das inverse und direkte dynamische Modell berechnet. Bei Gleichung (5.6) handelt es sich um die inverse Dynamik, das bei gegebenen Gelenkvariablen, Geschwindigkeiten und Beschleunigungen die Kräfte und Momente berechnet. Die Bewegungsgleichungen werden wie folgt in Matrizenschreibweise kompakt dargestellt.

$$\vec{\tau} = \mathcal{M}(Q) * \ddot{Q} + \mathcal{C}(Q, \dot{Q}) * \dot{Q} + \mathcal{G}(Q) \tag{5.9a}$$

$$= \mathcal{M}(Q) * \ddot{Q} + \mathcal{CO}(Q) * [\dot{Q}\dot{Q}] + \mathcal{CF}(Q) * [\dot{Q}^2] + \mathcal{G}(Q) \tag{5.9b}$$

- Die Matrix $\mathcal{CO}(Q)$ enthält die Coriolis–Koeffizienten. Die Coriolis–Kräfte bewirken in rotierenden Systemen eine seitliche Ablenkung und sind zu den Gelenkgeschwindigkeiten proportional. Die Dimension der Matrix ist $n \times n \cdot (n-1)/2$.

- Die Matrix $\mathcal{CF}(Q)$ enthält die Zentrifugal–Koeffizienten. Die Zentrifugalmatrix gibt die radial nach außen weisenden Reaktionskräfte bei Rotationsbewegungen an. Diese Kräfte sind umso größer, je größer die quadrierten Gelenkgeschwindigkeiten und die Massen der Manipulatorglieder sind. Die Dimension der Matrix ist $n \times n$.

- Der Vektor $[\dot{Q}\dot{Q}] = [\dot{q}_1\dot{q}_2, \dot{q}_1\dot{q}_3, \ldots, \dot{q}_1\dot{q}_n, \dot{q}_2\dot{q}_3, \dot{q}_2\dot{q}_4, \ldots, \dot{q}_{n-2}\dot{q}_n, \dot{q}_{n-1}\dot{q}_n]^T$ enthält alle möglichen Kombinationen von unterschiedlichen Paaren von Gelenkgeschwindigkeiten. Die Dimension des Vektors ist $n \cdot (n-1)/2 \times 1$.

- Der Vektor $[\dot{Q}^2] = [\dot{q}_1^2, \dot{q}_2^2, \ldots, \dot{q}_n^2]^T$ enthält die quadrierten Gelenkgeschwindigkeiten. Die Dimension des Vektors ist $n \times 1$.

Für die Bestimmung des direkten dynamischen Modells werden bei gegebenem Vektor $\vec{\tau}$ die Gleichungen (5.9a) und (5.9b) nach dem Beschleunigungsvektor $\ddot{Q}$ aufgelöst.

$$\ddot{Q} = \mathcal{M}(Q)^{-1} * \Big(\vec{\tau} - \mathcal{C}(Q,\dot{Q}) * \dot{Q} - \mathcal{G}(Q)\Big) \tag{5.10a}$$

$$= \mathcal{M}(Q)^{-1} * \Big(\vec{\tau} - \mathcal{CO}(Q) * [\dot{Q}\dot{Q}] - \mathcal{CF}(Q) * [\dot{Q}^2] - \mathcal{G}(Q)\Big) \tag{5.10b}$$

Das Vorgehen zur Aufstellung der Bewegungsgleichungen für einen Manipulator nach dem Lagrange Verfahren besteht im Wesentlichen aus vier Schritten. Es erfolgt eine kurze Zusammenfassung dieser Schritte, die vollautomatisch durchgeführt werden.

1. Berechnung der potentiellen und kinetischen Energie für jedes Gelenk.
2. Aufsummierung der potentiellen Energien. Hieraus ergibt sich der Gravitätsvektor.
3. Aufsummierung der kinetischen Energien. Hieraus ergibt sich die Massenmatrix.
4. Differentiation und Umformung der Massenmatrix gemäß den Lagrange Gleichungen. Hieraus ergeben sich die Zentrifugal– und die Coriolismatrix.

5.1.2 Experimentelle Ergebnisse

Es wurde ein sehr effektiver C/C++ Codegenerator entwickelt, welcher anhand einer Beschreibung des Roboters die Bewegungsgleichungen automatisch nach Lagrange erzeugt. Die einzelnen Schritte bei der Herleitung sowie das Ergebnis werden zudem in dem Satzsystem LaTeX dokumentiert. Der Zeitaufwand zum Aufstellen des dynamischen Modells für dieselben Robotertypen für die in Abschnitt 4.3.3 das kinematische Modell aufgestellt wurde, wurde experimentell mit einer mit 2.0 GHz getakteten Pentium–CPU unter dem Betriebssystem Linux ermittelt (Tabelle 5.1). Der Vorteil des Lagrange Verfahrens ist, dass man eine vollständige und geschlossene Darstellung der Roboterdynamik erhält. Da die Komplexität $\mathcal{O}(n^4)$ beträgt, wobei n die Anzahl der Gelenkachsen ist, entstehen teilweise auch viele Terme mit komplexen Ausdrücken (Tabelle 5.2). Werden konkrete Zahlen für die dynamischen Parameter, wie Massen, Schwerpunktslagen und Trägheitstensoren, eingesetzt, so verkürzen sich die erzeugten Gleichungen erheblich. Die numerische Berechnung ist dadurch unter Echtzeitbedingungen möglich und dauert bei einem Gelenkarmroboter 0.2 Millisekunden. Darüber hinaus ist die Auswertung der Gleichungen hochgradig parallelisierbar und kann zum Beispiel mit einer FPGA Architektur erfolgen.

Tabelle 5.1 Zeitbedarf zum Aufstellen der Bewegungsgleichungen (min:sec)

Roboter	Gravitäts–vektor	Massen–matrix	Zentrifugal–matrix	Coriolis–matrix
Kartesischer Roboter	00:01	00:02	00:01	00:01
Scara I	00:01	00:01	00:01	00:01
Scara II	00:01	00:01	00:01	00:01
Zylindrischer Roboter	00:01	00:05	00:01	00:01
Stanford Arm	00:01	00:34	00:08	00:02
Gelenkarmroboter	00:02	01:11	00:14	00:05

Tabelle 5.2 Evaluation des dynamischen Robotermodells für verschiedene Roboter

	Anzahl arithmetische Operatoren des Gravitationsvektors								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	2	5	28	0	2	2	0	0	0
Scara I	0	1	2	0	0	0	0	0	0
Scara II	3	0	4	0	0	0	0	0	0
Zylindrischer Roboter	5	4	32	0	2	2	0	0	0
Stanford Arm	8	15	103	0	3	3	0	0	0
Gelenkarmroboter	46	24	156	0	26	14	0	0	0

	Anzahl arithmetische Operatoren der Massenmatrix								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	24	10	78	0	0	0	0	0	0
Scara I	37	0	74	0	1	0	0	0	0
Scara II	30	0	59	0	1	0	0	0	0
Zylindrischer Roboter	66	20	311	0	1	1	0	0	0
Stanford Arm	178	62	1130	0	3	1	0	0	0
Gelenkarmroboter	520	96	1774	0	137	101	0	0	0

	Anzahl arithmetische Operatoren der Zentrifugalmatrix								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	8	11	66	0	2	0	0	0	0
Scara I	3	3	24	0	0	1	0	0	0
Scara II	2	2	18	0	0	1	0	0	0
Zylindrischer Roboter	38	38	350	0	4	0	0	0	0
Stanford Arm	135	161	1627	0	35	0	0	0	0
Gelenkarmroboter	739	232	2654	0	271	234	0	0	0

	Anzahl arithmetische Operatoren der Coriolismatrix								
Roboter	$+$	$-$	$*$	$/$	cos	sin	$atan_2$	$sqrt$	pow
Kartesischer Roboter	2	4	40	0	2	0	0	0	0
Scara I	0	3	16	0	0	0	0	0	0
Scara II	0	2	12	0	0	0	0	0	0
Zylindrischer Roboter	71	57	685	0	46	0	0	0	0
Stanford Arm	329	290	3750	0	204	6	0	0	0
Gelenkarmroboter	1531	572	6515	0	801	473	0	0	0

5.2 Rekursive Newton–Euler Verfahren

Außer dem Lagrange Verfahren wird das dynamische Modell auch nach dem rekursiven Verfahren von *Isaac Newton* (1643–1727) und *Leonhard Euler* (1707–1783) bestimmt. Rekursiv bedeutet in diesem Zusammenhang, dass die Absolutgrößen jedes Robotergelenks als Funktionen der Absolutgrößen des vorherigen Gelenks und den Relativgrößen zwischen ihnen beschrieben werden. Als Grundlage für dieses Verfahren dienen die drei

Newton'sche Axiome *(Trägheitsprinzip, Aktionsprinzip, Actio–Reactio–Prinzip)* [125], die Aussagen über die Bewegung von Körpern unter der Einwirkung von Kräften machen. Insbesondere das zweite Axiom beziehungsweise die *Newton'sche Gravitationsgleichung* ist wichtig für das Aufstellen von Bewegungsgleichungen. Der Berechnungsvorgang nach Newton–Euler erfolgt in zwei Teilen, und zwar einer Vorwärtsrechnung von der Basis des Roboters zum Endeffektor und einer Rückwärtsrechnung vom Endeffektor zur Basis.

5.2.1 Vorwärtsrechnung von der Basis zum Endeffektor

In der Vorwärtsrechnung werden ausgehend von der Basis die vektorielle Winkelgeschwindigkeit $\vec{\omega}_i$, Winkelbeschleunigung $\dot{\vec{\omega}}_i$ und Linearbeschleunigung $\dot{\vec{v}}_i$ bezüglich des Koordinatensystems jeder Gelenkachse iterativ bestimmt. Außerdem werden die auftretende Linearbeschleunigung $\dot{\vec{v}}_{ci}$ und Kraft $\mathcal{F}_i$ sowie das auftretende Drehmoment $\mathcal{N}_i$ im Schwerpunkt jedes Manipulatorglieds bestimmt. Im folgenden wird durch einen Kennungsparamter σ_i unterschieden, ob die i–te Achse translatorisch ($\sigma_i = 1$) oder rotatorisch ($\sigma_i = 0$) ist. Im ersten Schritt der Vorwärtsrechnung wird die Winkelgeschwindigkeit der i–ten Gelenkachse $\vec{\omega}_i$ berechnet, indem die Winkelgeschwindigkeit der vorangegangen Achse $\vec{\omega}_{i-1}$ und die Winkelgeschwindigkeit der aktuellen Achse zuerst addiert und dann mit der Rotationsmatrix $R_{i,i-1}$ in das aktuelle Koordinatensystem transformiert werden ($\vec{\omega}_0 = \vec{0}$).

$$\vec{\omega}_i = R_{i,i-1} * \left(\vec{\omega}_{i-1} + (1 - \sigma_i) \cdot \begin{pmatrix} 0 \\ 0 \\ \dot{\theta}_i \end{pmatrix} \right) \tag{5.11}$$

Durch Ableitung des Geschwindigkeitsvektors $\vec{\omega}_i$ nach der Zeit wird der Beschleunigungsvektor $\dot{\vec{\omega}}_i$ im Koordinatensystem der i–ten Gelenkachse bestimmt ($\dot{\vec{\omega}}_0 = \vec{0}$).

$$\dot{\vec{\omega}}_i = R_{i,i-1} * \left(\dot{\vec{\omega}}_{i-1} + (1 - \sigma_i) \cdot \left[\begin{pmatrix} 0 \\ 0 \\ \ddot{\theta}_i \end{pmatrix} + \vec{\omega}_{i-1} \times \begin{pmatrix} 0 \\ 0 \\ \dot{\theta}_i \end{pmatrix} \right] \right) \tag{5.12}$$

Als nächstes wird der Geschwindigkeitsvektor $\dot{\vec{v}}_i$ bezüglich des Ursprungs des i–ten Koordinatensystems ermittelt. Hierbei wird der Vektor $\vec{p}_i = (a_i, d_i \cdot \sin\alpha_i, d_i \cdot \cos\alpha_i)^T$ verwendet, der die Position des i–ten Koordinatensystems bezüglich des Vorgängerkoordinatensystems angibt. Die Erdanziehungskraft wird in der Initialisierung berücksichtigt ($\dot{\vec{v}}_0 = -\vec{g}$).

$$\dot{\vec{v}}_i = R_{i,i-1} * \left(\dot{\vec{v}}_{i-1} + \sigma_i \cdot \left[\begin{pmatrix} 0 \\ 0 \\ \ddot{d}_i \end{pmatrix} + 2 \cdot \vec{\omega}_{i-1} \times \begin{pmatrix} 0 \\ 0 \\ \dot{d}_i \end{pmatrix} \right] \right) + \dot{\vec{\omega}}_i \times \vec{p}_i + \vec{\omega}_i \times (\vec{\omega}_i \times \vec{p}_i) \tag{5.13}$$

Zur Berechnung des Linearbeschleunigungsvektors $\dot{\vec{v}}_{ci}$ des Massenmittelpunkts des i–ten Glieds wird der Vektor $\vec{r}_i$ verwendet. Der Vektor $\vec{r}_i$ ist der Vektor zum Massenmittelpunkt des i–ten Manipulatorglieds und wird im i–ten Koordinatensystem angegeben.

$$\dot{\vec{v}}_{ci} = \dot{\vec{v}}_i + \dot{\vec{\omega}}_i \times \vec{r}_i + \vec{\omega}_i \times (\vec{\omega}_i \times \vec{r}_i) \tag{5.14}$$

Die Kraft $\mathcal{F}_i$, die im Schwerpunkt des i–ten Glieds wirkt, entspricht der Änderung des Impulses nach der Zeit und wird durch Multiplikation der Masse m_i des Glieds mit der Linearbeschleunigung des Schwerpunkts des Glieds $\dot{\vec{v}}_{ci}$ berechnet. Diese Gleichung wird auch als *Impulssatz* bezeichnet und ist gleichwertig zu dem zweiten Newton'schen Axiom.

$$\mathcal{F}_i = m_i \cdot \dot{\vec{v}}_{ci} \tag{5.15}$$

Für die Berechnung des Moments, das im Schwerpunkt des i–ten Glieds wirkt, wird zuerst der Trägheitstensor ${}^{A}\mathcal{I}$ des Glieds in das Basiskoordinatensystem transformiert.

$$\mathcal{I}_{ci} = R_{i,0} *^{A} \mathcal{I} * R_{0,i} \tag{5.16}$$

Der Drehimpuls beziehungsweise Drall bezüglich des Schwerpunkts des i–ten Glieds wird durch Multiplikation der Matrix des Trägheitstensors $\mathcal{I}_{ci}$ mit dem Geschwindigkeitsvektor der Winkelrotation $\vec{\omega}_i$ berechnet. Durch Ableitung des Drehimpulses nach der Zeit wird das Drehmoment $\mathcal{N}_i$, das im Schwerpunkt des i–ten Glieds wirkt, bestimmt. Dieser Zusammenhang ist im *Drehimpulssatz* beziehungsweise *Drallsatz* festgehalten.

$$\mathcal{N}_i = \mathcal{I}_{ci} * \dot{\vec{\omega}}_i - \vec{\omega}_i \times (\mathcal{I}_{ci} * \vec{\omega}_i) \tag{5.17}$$

5.2.2 Rückwärtsrechnung vom Endeffektor zur Basis

In der Rückwärtsrechnung, die am Endeffektor beginnt und an der Roboterbasis endet, werden der Kraftvektor $\vec{f}_i$ und der Momentenvektor $\vec{n}_i$ den das Manipulatorglied i auf das Glied $i-1$ ausübt sowie die Kraft beziehungsweise das Moment τ_i, das im Gelenk i wirkt, iterativ bestimmt. Die Lasten am Endeffektor werden voreingestellt mit Null initialisiert ($\vec{f}_{i+1} = \vec{0}, \vec{n}_{i+1} = \vec{0}$). Der Kraftvektor $\vec{f}_i$ wird berechnet, indem der Kraftvektor der vorangegangenen Iteration mit der Rotationsmatrix $R_{i,i+1}$ in das aktuelle Koordinatensystem transformiert wird und die Kraft $\mathcal{F}_i$, die im Schwerpunkt des i–ten Glieds wirkt, addiert wird. Der Momentenvektor $\vec{n}_i$ wird analog hierzu berechnet. Zusätzlich werden hier allerdings noch zwei weitere Terme berechnet, und zwar ist ein Drehmoment als Vektorprodukt von Positionsvektor $\vec{p}_i$ und Kraftvektor $\vec{f}_i$ und ein weiteres als Vektorprodukt von Massenmittelpunktvektor $\vec{r}_i$ und Vektor $\mathcal{F}_i$ definiert. Schließlich wird in Abhängigkeit des Gelenktyps die Kraft beziehungsweise das Moment des i–ten Gelenks bestimmt.

$$\vec{f}_i = R_{i,i+1} * \vec{f}_{i+1} + \mathcal{F}_i \tag{5.18}$$

$$\vec{n}_i = R_{i,i+1} * \vec{n}_{i+1} + \mathcal{N}_i + \vec{p}_i \times \vec{f}_i + \vec{r}_i \times \mathcal{F}_i \tag{5.19}$$

$$\tau_i = (1-\sigma_i) \cdot \vec{n}_i^T * \left[R_{i,i-1} * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right] + \sigma_i \cdot \vec{f}_i^T * \left[R_{i,i-1} * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right] \tag{5.20}$$

5.2.3 Modifikation der Vorwärtsrechnung

Die Berechnungen der Vorwärts– und Rückwärtsiteration können sowohl numerisch als auch symbolisch durchgeführt werden. Bei einem symbolischen Vorgehen können die ermittelten Bewegungsgleichungen in Matrizenschreibweise mit Gravitationsvektor sowie Massen–, Zentrifugal– und Coriolismatrix (Gleichung 5.9b) gebracht werden, um eine übersichtliche Struktur der Gleichungen zu erhalten. Außerdem ergeben sich dadurch dieselben Differentialgleichungen 2. Ordnung wie beim Lagrange Verfahren. Dieses Vorgehen ist bei einfacheren Robotertypen mit bis zu fünf Gelenkachsen möglich. Da die Anzahl und die Komplexität der Gleichungen mit der Anzahl der Gelenkachsen eines Roboters stark wächst, werden bei komplexeren Robotertypen sehr umfangreiche und unüberschaubare Gleichungen generiert. Diese Gleichungen lassen sich nur noch sehr schlecht vereinfachen und sind letztlich nicht mehr umformbar.

Um die Gleichungen beherrschbar zu machen, wurde die Vorwärtsiteration im Rahmen

dieser Arbeit dahingehend abgeändert, dass in der ersten Iteration nur die Terme bezüglich des Gravitationsvektors, in der zweiten Iteration nur die Terme bezüglich der Massenmatrix, in der dritten Iteration nur die Terme bezüglich der Zentrifugalmatrix und in der vierten Iteration nur die Terme bezüglich der Coriolismatrix berechnet werden. Die Berechnungen der Rückwärtsiteration bleiben unverändert. Zur Berechnung des Gravitationsvektors wird die Vorwärtsiteration folgendermaßen abgeändert. Hierbei werden lediglich die Massen m_i, die Rotationsmatrizen $R_{i,0}$ und der Gravitätsvektor $\vec{g}$ verwendet.

$$\mathcal{F}_i = -m_i \cdot R_{i,0} * \vec{g} \tag{5.21a}$$

$$\mathcal{N}_i = \vec{0} \tag{5.21b}$$

Die Modifikation der Vorwärtsiteration zur Berechnung der Terme der Massenmatrix sieht wie folgt aus. In der Initialisierung wird die Erdanziehungskraft nicht mehr berücksichtigt ($\vec{\omega}_0 = 0, \vec{v}_0 = 0$) und die Berechnungen bezüglich Winkelgeschwindigkeiten $\vec{\omega}_i$ entfallen.

$$\dot{\vec{\omega}}_i = R_{i,i-1} * \left(\dot{\vec{\omega}}_{i-1} + (1 - \sigma_i) \cdot \begin{pmatrix} 0 \\ 0 \\ \ddot{\theta}_i \end{pmatrix} \right) \tag{5.22a}$$

$$\dot{\vec{v}}_i = R_{i,i-1} * \left(\dot{\vec{v}}_{i-1} + \sigma_i \cdot \begin{pmatrix} 0 \\ 0 \\ \ddot{d}_i \end{pmatrix} \right) + \dot{\vec{\omega}}_i \times \vec{p}_i \tag{5.22b}$$

$$\mathcal{F}_i = m_i \cdot (\dot{\vec{v}}_i + \dot{\vec{\omega}}_i \times \vec{r}_i) \tag{5.22c}$$

$$\mathcal{N}_i = (R_{i,0} *^A \mathcal{I} * R_{0,i}) * \dot{\vec{\omega}}_i \tag{5.22d}$$

5.2.4 Experimentelle Ergebnisse

Es wurde ein System entwickelt, welches anhand einer Manipulatorbeschreibung die Bewegungsgleichungen analytisch nach dem Newton–Euler Verfahren herleitet und daraus eine hocheffiziente Softwarekomponente in C/C++ generiert. Für dieselben Robotertypen für die in Abschnitt 5.1.2 das Dynamikmodell in symbolischer Matrizendarstellung nach Lagrange bestimmt wurde, wurde es auch nach dem Newton–Euler Verfahren bestimmt. Der Gravitationsvektor sowie die Massen–, Zentrifugal– und Coriolismatrix wurden für die einzelnen Robotertypen, wie in Abschnitt 5.2.3 beschrieben, nacheinander berechnet. Durch dieses Vorgehen wurden beim Newton–Euler Verfahren äquivalente Ergebnisse bezüglich der generierten Differentialgleichungen 2. Ordnung wie beim Lagrange Verfahren erzielt (Tabelle 5.2). Eine sehr hohe Softwarequalität ist hierduch sichergestellt.

5.3 Kane's dynamische Gleichungen

Kane's Methode [84, 190] ist eine ebenfalls wichtige Methode zur Bestimmung der Roboterdynamik. Im Gegensatz zu den Verfahren von Lagrange und Newton–Euler werden keine Differentialgleichungen 2. Ordnung, sondern Differentialgleichungen 1. Ordnung (engl.: ordinary differential equation) bestimmt, das heißt die Bewegungsgleichungen enthalten nur erste Ableitungen nach der Zeit. Der Vorteil ist, dass die Gleichungen leichter zu integrieren sind. Kane's Grundgleichung besagt, dass die Summe aller Kräfte, die auf einen Körper einwirken, gleich Null sein muss. Dabei sind die von außen an einen Körper angreifenden Kräfte betragsmäßig so groß wie die Trägheitskraft eines Körpers. In den Termen für die Trägheitskraft sind konstante Massen und Gelenkbeschleunigungen enthalten.

5.4 Vergleich der Verfahren

Das dynamische Robotermodell ist sehr stark von der Bauart des Roboters abhängig und wird beispielsweise zur Vorsteuerung in Kaskadenreglern verwendet (Abschnitt 6.1). Zur Aufstellung des vollständigen Dynamikmodells werden sowohl das Lagrange Verfahren (Abschnitt 5.1) als auch das rekursive Newton–Euler Verfahren (Abschnitt 5.2) angewendet. Das dynamische Modell wird nicht nur in Form von Zahlenwerten berechnet, sondern es werden die analytischen Gleichungen automatisch aufgestellt. Hierfür werden die Koordinatensysteme gemäß der Denavit–Hartenberg Konvention (Abschnitt 4.1) festgelegt.

Die Äquivalenz des Lagrange und des Newton–Euler Verfahrens für Manipulatoren wurde von Silver [158] bewiesen. Beide Verfahren sind sowohl für Manipulatoren mit rotatorischen als auch mit translatorischen Gelenkachsen geeignet. Während das erste Verfahren ein analytisches Verfahren ist, handelt es sich beim zweiten um ein synthetisches Verfahren. Das Newton–Euler Verfahren hat den Nachteil, dass Zwangskräfte und –momente in den Gelenken explizit berechnet werden müssen, die in den Differentialgleichungen gar nicht explizit auftauchen. Um eine geschlossene Darstellung der Roboterdynamik zu erhalten, müssen diese Zwangskräfte und –momente zunächst ermittelt und in einem weiteren Schritt eliminiert werden. Beim Lagrange Verfahren brauchen hingegen keine Zwangskräfte und –momente in den Gelenken berücksichtigt werden. Deswegen ist der analytische Aufwand zum Aufstellen der Bewegungsgleichungen beim Lagrange Verfahren geringer als beim Newton–Euler Verfahren. Bei einer ausschließlich numerischen Bestimmung der Dynamik ist hingegen der Aufwand bei Newton–Euler linear proportional zu der Anzahl der Gelenkachsen und damit, insbesondere bei Robotern mit vielen Freiheitsgraden, niedriger als bei Lagrange.

Kapitel 6

Softwarearchitektur und Konfigurationswerkzeug für die Bewegungssteuerung von Robotern

In diesem Kapitel wird die eigentliche Konfiguration der Bewegungssteuerung unter Verwendung der automatisch generierten kinematischen und dynamischen Robotermodelle (Kapitel 4 und 5) beschrieben. Zunächst wird ein Referenzmodell mit den Komponenten einer Bewegungssteuerung aufgestellt (Abschnitt 6.1). Als nächstes wird die graphische Benutzeroberfläche beschrieben, mit der der Benutzer die Bewegungssteuerung eines Roboters konfigurieren kann (Abschnitt 6.2). In Abschnitt 6.3 werden verschiedene Trajektoriengenerierungskomponenten entwickelt, die für die unterschiedlichen Betriebsarten einer Bewegungssteuerung benötigt werden. Für die Vorverarbeitung und Ausführung von Roboterprogrammen wurde ein IRL–Interpreter (Industrial Robot Language) entwickelt (Abschnitt 6.4). Außerdem kann der Benutzer eigene Komponenten über eine Anwendungsprogrammierschnittstelle in das System integrieren (Abschnitt 6.5).

Die Bewegungssteuerung eines Roboters stellt eine Echtzeitanwendung dar, das heißt es bestehen harte Anforderungen hinsichtlich der Einhaltung zeitkritischer Berechnungen [40, 193]. Diese Berechnungen müssen in jeder Betriebssituation innerhalb einer vorgegebenen, kurzen Zeitspanne durchgeführt werden. Zu den Hauptaufgaben einer Bewegungssteuerung gehören die Berechnung von geeigneten zeitlichen Zwischenwerten zwischen den programmierten Zielstellungen des Roboters (Interpolation) und die Transformation der Effektorlage, die in kartesischen Koordinaten angegeben wird, in entsprechende Sollwerte für die Gelenkregelkreise (inverse Kinematik). Außerdem steuert die Bewegungssteuerung die einzelnen Gelenkachsen zeitlich parallel an, um die gewünschte Bewegung zu erzeugen.

6.1 Referenzmodell einer Bewegungssteuerung

Die in dieser Arbeit betrachtete Bewegungssteuerung ist in mehrere Softwarekomponenten aufgeteilt und wird in Abbildung 6.1 als Blockschaltbild dargestellt. Die Pfeile in der Abbildung stellen die Kommunikationswege der verschiedenen Komponenten untereinander dar. Durch die Strukturierung der Bewegungssteuerung in Teilsysteme wird ihre Komplexität reduziert und ihre Wartbarkeit verbessert. Die einzelnen Komponenten sind mit einheitlichen Schnittstellen ausgestattet, um mit den anderen Komponenten zusammenzuarbeiten. Die einzelnen Komponenten erfüllen die folgenden Aufgaben:

Roboter Interpreter: Zur Abarbeitung der Bewegungsprogramme wurde ein herstel-

lerübergreifender Interpreter nach DIN 66312 [38] entwickelt (Abschnitt 6.4). Der Interpreter sendet die vorverarbeiteten Befehle an die Interpolationsvorbereitung.

Kartesische Interpolation: Ist im Roboterprogramm eine Bewegung des TCP im kartesischen Raum angegeben, so berechnen kartesische Interpolationskomponenten (Abschnitt 6.3.2) eine entsprechende lineare oder zirkulare Bahn. Vor Interpolationsbeginn berechnet die Interpolationsvorbereitung die Bahnlänge beziehungsweise den Kurvenradius und den Kreismittelpunkt sowie die Orientierungsänderung.

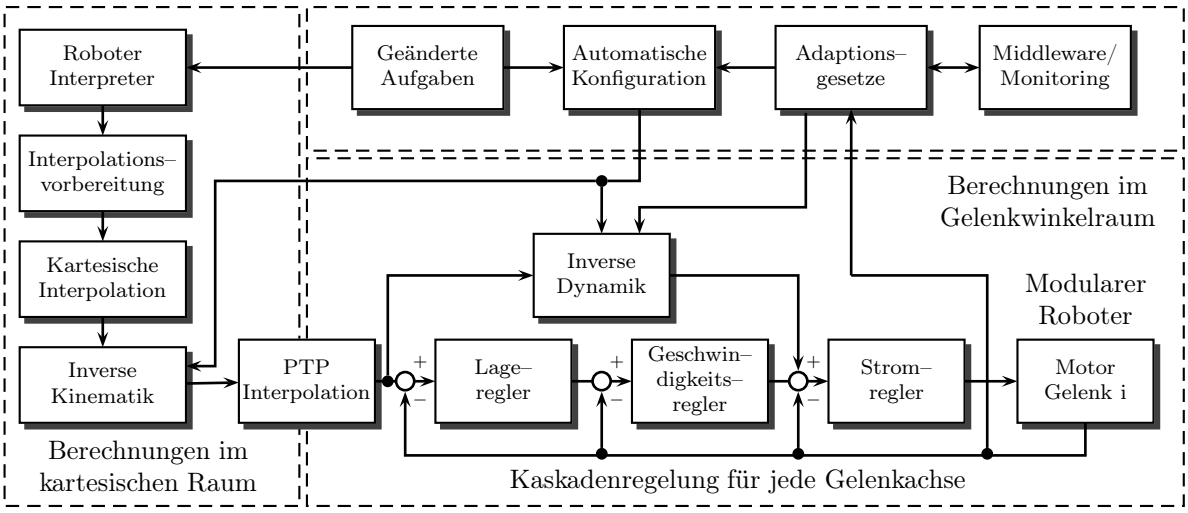
Inverse Kinematik: Die Umwandlung von Koordinaten im kartesischen Raum in entsprechende Koordinaten im Gelenkwinkelraum erfolgt analytisch. Falls für einen Manipulator keine geschlossene Lösung bestimmt werden kann, so wird eine numerische Approximation angewendet (Kapitel 4).

PTP–Interpolation: Diese Komponente (Abschnitt 6.3.1) berechnet die Bewegung jedes Gelenks zwischen dem Startpunkt und dem Endpunkt einer Bewegung entsprechend eines ausgewählten Beschleunigungsprofils, ohne dass die kartesische Bahn des Endeffektors hierbei definiert wurde. Die Achsen werden so verfahren, dass sich ein zeitoptimaler Bewegungsablauf ergibt und die Maximalwerte für Geschwindigkeit, Beschleunigung und Ruck der einzelnen Antriebe nicht überschritten werden.

Kaskadenregelungen: Weitere Komponenten sind eine Kaskadenreglung pro Roboterachse, bestehend aus einem Lage–, Geschwindigkeits– und Stromregler, wobei der übergeordnete Regler den Sollwert für den untergeordneten Regler bestimmt.

Inverse Dynamik: Zur Verbesserung des Führungsverhaltens wird mit Hilfe des inversen dynamischen Modells eine Stellgröße aufgeschaltet, um die auf die Roboterbewegungen einwirkenden Massenträgheits–, Coriolis–, Zentrifugal– und Gravitationskräfte beziehungsweise Momente zu kompensieren (Kapitel 5). Außerdem werden die einzelnen Kaskadenregler dadurch entlastet. Durch entsprechende Adaptionsgesetze können nicht exakt bekannte Größen, wie zum Beispiel Modellparameter oder zu bewegende Traglasten, an die realen Werte angepasst werden.

Abbildung 6.1 Softwarearchitektur der Bewegungssteuerung eines Roboters



6.2 Beschreibung des Aufbaus des zu konfigurierenden Roboters

Zur Beschreibung der mechanischen Struktur eines Manipulators wurde ein Konfigurationswerkzeug entwickelt. Die graphische Benutzeroberfläche (Abbildung 6.2) entstand im Rahmen einer Studienarbeit [139], die in dieser Arbeit betreut wurde. Bei der Gestaltung der Oberfläche wurde auf Benutzerfreundlichkeit und einfache Bedienbarkeit geachtet. Um plattformunabhängig und leicht portierbar zu sein, wurde die Oberfläche in der Programmiersprache JAVA implementiert. Mit diesem Werkzeug kann der Benutzer zwischen sehr vielen Konfigurations– und Kombinationsmöglichkeiten auswählen:

- Anzahl und Typen der Gelenkachsen, zum Beispiel passive oder angetriebene (translatorische beziehungsweise rotatorische) Gelenkachsen. In Abhängigkeit der angegebenen Gelenkanzahl werden die Eingabefelder für die weiteren Roboterparameter angepasst. Je mehr Gelenkachsen, insbesondere rotatorische Gelenkachsen der Roboter hat, desto komplizierter werden die zu bestimmenden mathematischen Modelle des Roboters (erhebliches Skalierbarkeitsproblem).

- Mechanische Bewegungsbeschränkungen der Achsen, das heißt Stellbereiche (Anschlag bei Rotationsgelenken und Längenbegrenzung bei Translationsgelenken), maximal mögliche Geschwindigkeiten und Beschleunigungen, Maximalruck sowie maximal erlaubte Momente und Kräfte der einzelnen Achsen. Diese Werte müssen so gewählt werden, damit das Leistungsvermögen der Antriebe nicht überschritten wird. Der Stellbereich von Rotationsgelenken kann heute auch größer als 360° sein.

- Anordnung und Bewegungsmöglichkeiten der Haupt– und Nebenachsen, das heißt Lage der Drehrichtung von rotatorischen Achsen und Lage der Verschiebungsrichtung von translatorischen Achsen. Zwei aufeinanderfolgende Gelenkachsen können beispielsweise parallel, rechtwinklig oder windschief zueinander angeordnet werden. Winkelmaße können numerisch im Bogenmaß oder symbolisch angegeben werden.

- Geometrische Parameter, wie Hebellängen und Hebelverschiebungen werden entweder direkt vom Hersteller geliefert oder am Roboter gemessen und durch Kalibrierung angepasst.

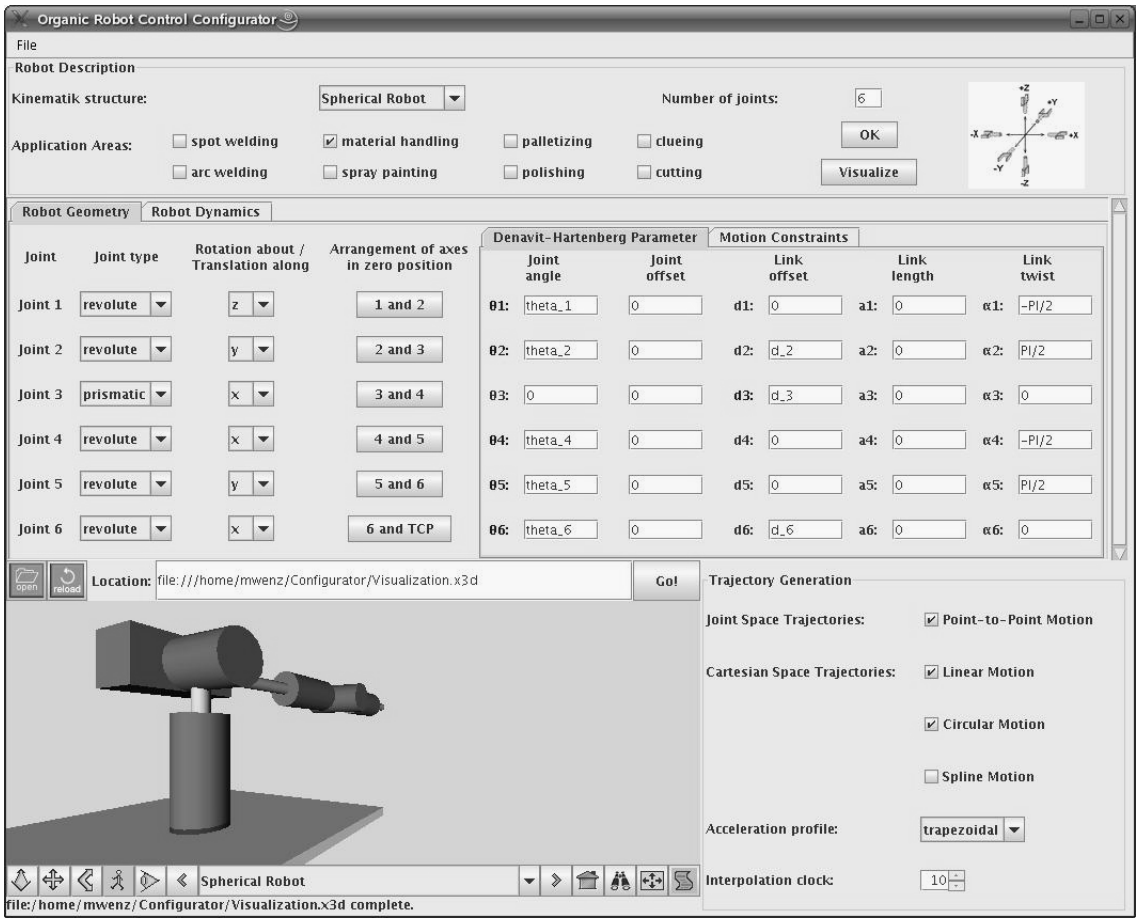
- Konzentrierte Massen der Manipulatorglieder, Schwerpunkte der Glieder in Koordinaten der lokalen Koordinatensysteme sowie Trägheitstensoren, die die Trägheit der Gelenkachsen bezüglich Bewegungen angeben. Für jedes Glied wird die Lage des Schwerpunkts durch einen dreidimensionalen und die Trägheitstensoren durch einen sechsdimensionalen Vektor angegeben. Für die Massen der Manipulatorglieder gilt die Einschränkung, dass keine negativen Werte eingegeben werden dürfen.

- Kinematische Bauart, zum Beispiel kartesischer, zylindrischer, kugelförmiger Roboter, horizontaler oder vertikaler Knickarmroboter sowie Sonderbauformen.

- Zu unterstützende Interpolationsarten, Interpolationstakt, maximale Geschwindigkeit bei der Interpolation und Beschleunigungsprofilart, zum Beispiel Trapezprofil (beziehungsweise Dreieckprofil bei kurzer Weglänge), Sinoidenprofil oder Polynomfunktionprofil. Außerdem kann zwischen asynchronen, synchronen und vollsynchronen Punkt–zu–Punkt Bewegungen ausgewählt werden.

Die Benutzereingaben werden zunächst auf Plausibilität überprüft und dann in einem XML–Dokument (Listing C.1) gespeichert. Anhand der ausgewählten Kriterien und den spezifizierten geometrischen Abmessungen werden die mathematischen Modelle des Roboters „auf Knopfdruck“ generiert und die Bewegungssteuerung (Abbildung 6.1) konfiguriert. Die Beschreibung des Benutzers kann auch in einer Bibliothek abgelegt legen, um sie später wiederzuverwenden beziehungsweise anzupassen, beispielsweise bei einem Roboter mit der gleichen Gelenkanordnung, aber anderen geometrischen Abmessungen. Außerdem wird zur Kontrolle der Eingaben der beschriebene Manipulator im unteren Fenster des Konfigurationswerkzeugs dreidimensional dargestellt. Diese Darstellung kann vom Benutzer gedreht und verschoben beziehungsweise vergrößert oder verkleinert werden.

Abbildung 6.2 Konfigurator für die Bewegungssteuerung eines Roboters



6.3 Betriebsarten einer Bewegungssteuerung

Damit ein Roboter eine bestimmte Bewegung dürchführen kann, muß die Bewegungssteuerung entsprechende Sollwerte für die Gelenkregelkreise erzeugen. Die zeitliche Aufeinanderfolge der Roboterkonfiguration zwischen Anfangs– und Endstellung des Endeffektors wird auch als *Trajektorie* bezeichnet. In Bewegungssteuerungen werden grundsätzlich zwei verschiedene Betriebsarten unterschieden. *Punkt–zu–Punkt Bewegungen* (engl.: Point–To–Point, PTP) werden genutzt, wenn der Endeffektor gewünschte Lagen anfahren soll und

der Weg dorthin nicht genau festgelegt sein muss („unkontrollierter Verlauf“). Außer dem Verlauf der kartesischen Bahn sind auch die auftretenden kartesischen Geschwindigkeiten und Beschleunigungen dabei nicht vorhersagbar. Diese Bewegungsart wird typischerweise für reine Positionierbewegungen, zum Beispiel beim Punktschweißen und bei Handhabungsaufgaben eingesetzt. Für die Werkstückbearbeitung und in der Nähe von Hindernissen ist diese Bewegungsart nicht geeignet, da die Gefahr von Kollisionen besteht.

Im Gegensatz hierzu wird bei bahnbezogenen Aufgabenstellungen, wie zum Beispiel beim Entgraten, Schleifen, Polieren, Lackieren, Kleben, Falzen, Montieren oder Bahnschweißen, die Werkzeugspitze exakt entlang von vorgeschriebenen Bahnen bewegt. Um den Rechenaufwand gering zu halten werden diese Bahnen mathematisch meistens durch Geraden oder Kreise beziehungsweise Kreissegmente beschrieben. Bei *kartesischer Bahnsteuerung* (engl.: Continuous Path, CP) können sich im Gegensatz zu PTP–Bewegungen Probleme mit Singularitäten und Arbeitsraumgrenzen ergeben. Auch wenn der Anfangs– und der Endpunkt der Bewegung sich innerhalb des Arbeitsraums befindet, kann es Zwischenpunkte geben, die außerhalb des Arbeitsraums liegen.

6.3.1 Trajektoriengenerierung im Gelenkwinkelraum

PTP–Bewegungen sind theoretisch die schnellste Möglichkeit, um einen Roboter von einer Anfangspose zu einer Endpose zu verfahren. Nur für die Zielpose, die in kartesischen Koordinaten angegeben wird, werden mit Hilfe der inversen Kinematik entsprechende Gelenkkoordinaten im Gelenkwinkelraum ermittelt. Die Zwischenkoordinaten werden im Gelenkwinkelraum generiert. Bei PTP–Bewegungen wird zwischen asynchronen, synchronen und vollsynchronen Bewegungen unterschieden.

Asynchrone PTP–Bewegung: Die Gelenkachsen werden so schnell wie möglich in ihre Endstellung verfahren. Die Sollwerte für die einzelnen Achsregler werden unabhängig voneinander berechnet. In der Regel erreichen die Achsen nicht zur gleichen Zeit die Endposition.

Synchrone PTP–Bewegung: Alle Achsen beginnen und beenden ihre Bewegung zur gleichen Zeit. Hierfür wird für jedes Gelenk anhand der Weg– beziehungsweise Winkeldistanz, maximalen Geschwindigkeit und Beschleunigung die Bewegungsdauer berechnet. Die Achse mit der größten Bewegungsdauer wird zur *Leitachse*. Die Maximalgeschwindigkeiten der übrigen Achsen werden entsprechend verringert, so dass ihre Bewegungsdauer so lange wird, wie die der Leitachse.

Vollsynchrone PTP–Bewegung: Zusätzlich werden auch die Beschleunigungs– und Bremsphasen angepasst, so dass sie bei allen Achsen gleich lange dauern.

Eine Bewegung unterteilt sich in die Phasen Beschleunigung, gleichförmige Bewegung und Bremsvorgang. Zur Trajektoriengenerierung werden für alle Gelenke Beschleunigungsprofile berechnet mit denen ruckfreie Bewegungen sowie die Einhaltung der maximal erlaubten Gelenkgeschwindigkeiten und –beschleunigungen gewährleistet werden. Durch die Begrenzung des Rucks werden die Gelenke sanft beschleunigt und abgebremst und es verringert sich die Beanspruchung der Mechanik und der Maschinenverschleiß. Werden die Grenzwerte nicht eingehalten, so treten gravierende Bahnabweichungen auf. Für die Berechnung eines Beschleunigungsprofils gibt es mehrere Möglichkeiten, die im folgenden vorgestellt werden. Die Integration des Beschleunigungsprofils ergibt das Geschwindigkeitsprofil. Integriert man das Geschwindigkeitsprofil, so erhält man das Wegprofil.

Trapezförmiges Beschleunigungsprofil (Dreieckprofil bei kurzer Weglänge)

Bei einem trapezförmigen Beschleunigungsprofil existieren maximal sieben Zeitintervalle, in denen eine Anfangsposition $q(0)$ in eine Endposition $q(t_7)$ unter Berücksichtigung der Maximalgeschwindigkeit v_{max}, der Maximalbeschleunigung a_{max} und des Maximalrucks j_{max} überführt wird. Die Bewegungszeit t_7 soll dabei möglichst kurz sein. Die resultierende Bahn $q(t)$, das Geschwindigkeitsprofil $\dot{q}(t)$ und das Beschleunigungsprofil $\ddot{q}(t)$ werden durch die Gleichungen (6.1)–(6.3) bestimmt. Durch Ableitung des Beschleunigungsprofils entsteht ein blockförmiges Ruckprofil. Die Gelenkgeschwindigkeiten beginnen mit $\dot{q}(0)$ und enden mit $\dot{q}(t_7)$, die Gelenkbeschleunigungen beginnen und enden bei Null.

$$q(t) = \begin{cases} q(0) + \dot{q}(0) \cdot t + \frac{1}{6} \cdot j_{max} \cdot t^3 & , 0 \le t \le t_1 \\ q(0) + \dot{q}(0) \cdot t + \frac{1}{2} \cdot a_{max} \cdot t^2 - \frac{1}{2} \cdot \frac{a_{max}^2}{j_{max}} \cdot t + \frac{1}{6} \cdot \frac{a_{max}^3}{j_{max}^2} & , t_1 < t \le t_2 \\ q(0) + \dot{q}(0) \cdot t + a_{max} \cdot t_2 \cdot t + \frac{j_{max}}{6} \cdot (t_3 - t)^3 - \frac{1}{2} \cdot a_{max} \cdot t_2 \cdot t_3 & , t_2 < t \le t_3 \\ q(0) + v_{max} \cdot t + \frac{t_3}{2} \cdot (\dot{q}(0) - v_{max}) & , t_3 < t \le t_4 \\ q(0) + v_{max} \cdot t + \frac{j_{max}}{6} \cdot (t_4 - t)^3 + \frac{t_3}{2} \cdot (\dot{q}(0) - v_{max}) & , t_4 < t \le t_5 \\ q(0) + v_{max} \cdot t - \frac{a_{max}}{2} \cdot (t_5 - t)^2 + \frac{a_{max}^2}{j_{max}} \cdot (\frac{t_5}{3} + \frac{t_4}{6} - \frac{t}{2}) & \\ \quad + \frac{t_3}{2} \cdot (\dot{q}(0) - v_{max}) & , t_5 < t \le t_6 \\ q(t_7) + \dot{q}(t_7) \cdot (t - t_7) + \frac{j_{max}}{6} \cdot (t - t_6)^3 - \frac{a_{max}}{2} \cdot (t_7 - t)^2 & \\ \quad + \frac{a_{max}^2}{j_{max}} \cdot (\frac{t_7}{3} + \frac{t_6}{6} - \frac{t}{2}) & , t_6 < t \le t_7 \end{cases} \tag{6.1}$$

$$\dot{q}(t) = \begin{cases} \dot{q}(0) + \frac{1}{2} \cdot j_{max} \cdot t^2 & , 0 \le t \le t_1 \\ \dot{q}(0) + a_{max} \cdot t - \frac{1}{2} \cdot \frac{a_{max}^2}{j_{max}} & , t_1 < t \le t_2 \\ \dot{q}(0) + a_{max} \cdot t_2 - \frac{j_{max}}{2} \cdot (t_3 - t)^2 & , t_2 < t \le t_3 \\ v_{max} & , t_3 < t \le t_4 \\ \dot{q}(t_7) + a_{max} \cdot (t_7 - t_5) - \frac{j_{max}}{2} \cdot (t_4 - t)^2 & , t_4 < t \le t_5 \\ \dot{q}(t_7) + a_{max} \cdot (t_7 - t) - \frac{1}{2} \cdot \frac{a_{max}^2}{j_{max}} & , t_5 < t \le t_6 \\ \dot{q}(t_7) + \frac{j_{max}}{2} \cdot (t - t_6)^2 + a_{max} \cdot (t_7 - t) - \frac{1}{2} \cdot \frac{a_{max}^2}{j_{max}} & , t_6 < t \le t_7 \end{cases} \tag{6.2}$$

$$\ddot{q}(t) = \begin{cases} j_{max} \cdot t & , 0 \le t \le t_1 \\ a_{max} & , t_1 < t \le t_2 \\ a_{max} - j_{max} \cdot (t - t_2) & , t_2 < t \le t_3 \\ 0 & , t_3 < t \le t_4 \\ -j_{max} \cdot (t - t_4) & , t_4 < t \le t_5 \\ -a_{max} & , t_5 < t \le t_6 \\ j_{max} \cdot (t - t_6) - a_{max} & , t_6 < t \le t_7 \end{cases} \tag{6.3}$$

Der zu fahrende Weg ist $\Delta q = |q(0) - q(t_7)|$. In den Phasen $i = 1, 3, 5, 7$ wird jeweils mit $t_i - t_{i-1} = |a_{max}/j_{max}|$ maximal beschleunigt beziehungsweise abgebremst. Die gesamte Beschleunigungszeit t_3 und die gesamte Abbremszeit $t_{74} = t_7 - t_4$ ergeben sich zu:

$$t_3 = \frac{v_{max} - \dot{q}(0)}{a_{max}} + \frac{a_{max}}{j_{max}} \qquad t_{74} = \frac{v_{max} - \dot{q}(t_7)}{a_{max}} + \frac{a_{max}}{j_{max}} \tag{6.4}$$

Die Zeitspanne $t_{43} = t_4 - t_3$, in der mit konstanter maximaler Geschwindigkeit verfahren wird, beträgt:

$$t_{43} = \frac{1}{v_{max}} \cdot \left(\Delta q - \frac{v_{max} + \dot{q}(0)}{2} \cdot t_3 - \frac{v_{max} + \dot{q}(t_7)}{2} \cdot t_{74} \right) \tag{6.5}$$

Falls $t_{43} < 0$ gilt, ist die Maximalgeschwindigkeit hinsichtlich der Bahnlänge und der Beschleunigung zu groß und wird entsprechend Gleichung (6.6) verringert. Das Zeitintervall in dem die Geschwindigkeit konstant ist, fällt weg und das Gesamtprofil teilt sich nur noch in sechs Zeitintervalle auf.

$$t_{43} = 0 \Longrightarrow \bar{v}_{max} = \frac{2 \cdot \Delta q - \dot{q}(0) \cdot t_3 - \dot{q}(t_7) \cdot t_{74}}{t_3 + t_{74}} \tag{6.6}$$

Bei synchronen Bewegungen bestimmt die Achse mit der größten Verfahrzeit die Dauer der Bewegung. Es wird für jede Achse die Fahrzeit bestimmt und die Achse mit der maximalen Fahrzeit t_{max} zur Leitachse gemacht. Die Gleichung $t_{max} = t_3 + t_{43} + t_{74}$ wird nach der neuen maximalen Geschwindigkeit $\tilde{v}_{max}$ für jedes Gelenk aufgelöst. Durch die Reduzierung der Maximalgeschwindigkeit von jeder Achse außer der Leitachse, ergibt sich für alle Achsen dieselbe Bewegungsdauer.

$$\tilde{v}_{max} = \frac{\dot{q}(0) + \dot{q}(t_7) + a_{max} \cdot (t_{max} - t_1)}{2} - \sqrt{\frac{(\dot{q}(0)+\dot{q}(t_7)+a_{max}\cdot(t_{max}-t_1))^2 - 2\cdot(\dot{q}^2(0)+\dot{q}^2(t_7)+a_{max}\cdot(2\cdot\Delta q - t_1\cdot(\dot{q}(0)+\dot{q}(t_7))))}{4}} \tag{6.7}$$

Für vollsynchrone Bewegungen wird die maximale Beschleunigungszeit $t_{3,max}$, die maximale Abbremszeit $t_{74,max}$ und die maximale Zeit $t_{43,max}$, in der mit konstanter Geschwindigkeit verfahren wird, bestimmt. Mit diesen Werten wird die maximale Gesamtverfahrzeit $t_{7,max} = t_{3,max} + t_{43,max} + t_{74,max}$ ermittelt. Durch Auflösung der Gleichungen (6.4) und (6.5) wird die neue Maximalgeschwindigkeit $\hat{v}_{max}$ sowie die Maximalbeschleunigung während der Beschleunigungsphase $\hat{a}_{1,max}$ und während der Abbremsphase $\hat{a}_{2,max}$ von jeder Achse bestimmt. Die Beschleunigungs– und Abbremszeit ist nun für alle Achsen gleich.

$$\hat{v}_{max} = \frac{2 \cdot \Delta q - t_{3,max} \cdot \dot{q}(0) - t_{74,max} \cdot \dot{q}(t_7)}{2 \cdot t_{7,max} - t_{3,max} - t_{74,max}} \tag{6.8}$$

$$\hat{a}_{1,max} = \frac{t_{3,max} \cdot j_{max} - \sqrt{(t_{3,max} \cdot j_{max})^2 - 4 \cdot j_{max} \cdot (\hat{v}_{max} - \dot{q}(0))}}{2} \tag{6.9}$$

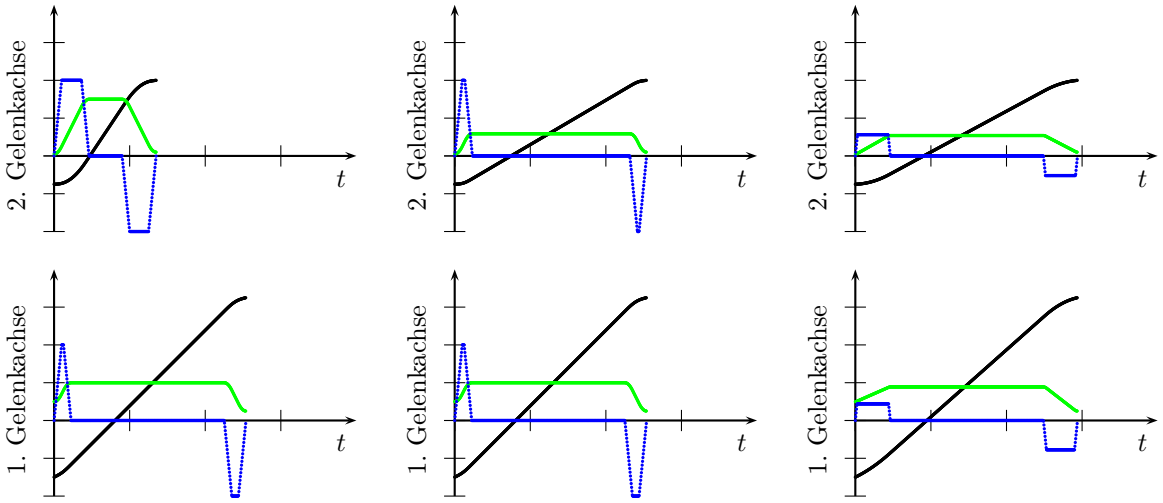
$$\hat{a}_{2,max} = \frac{t_{74,max} \cdot j_{max} - \sqrt{(t_{74,max} \cdot j_{max})^2 - 4 \cdot j_{max} \cdot (\hat{v}_{max} - \dot{q}(t_7))}}{2} \tag{6.10}$$

Die drei verschiedenen Möglichkeiten für PTP–Bewegungen mit trapezförmigen Beschleunigungsprofil sind in Abbildung 6.3 dargestellt. Zwei Gelenkachsen werden asynchron (links), synchron (Mitte) und vollsynchron (rechts) von einem Startpunkt zu einem Endpunkt unter Berücksichtigung der maximal zulässigen Gelenkgeschwindigkeiten und Beschleunigungen sowie des maximal zulässigen Rucks bewegt.

Sinusförmiges Beschleunigungsprofil

Das Sinoidenprofil hat den Vorteil, dass im Gegensatz zum Trapezprofil der Beschleunigungsverlauf unendlich oft stetig differenzierbar ist. Allerdings steigt durch die Berechnung von trigonometrischen Funktionen auch der Rechenaufwand. Das Gesamtprofil ist

Abbildung 6.3 PTP–Bewegungen mit trapezförmigen Beschleunigungsprofil



nur noch in drei und nicht mehr in sieben Segmente aufgeteilt. Gleichung (6.11) gibt den Verlauf der Gelenkkoordinaten mit Anfangswert $q(0)$ und Endwert $q(t_3)$, Gleichung (6.12) den Verlauf der Gelenkgeschwindigkeiten mit Anfangsgeschwindigkeit $\dot{q}(0)$ und Endgeschwindigkeit $\dot{q}(t_3)$ und Gleichung (6.13) den Verlauf der Gelenkbeschleunigungen an. Anfangsbeschleunigung $\ddot{q}(0)$ und Endbeschleunigung $\ddot{q}(t_3)$ sind Null.

$$q(t) = \begin{cases} q(0) + \dot{q}(0) \cdot t + a_{max} \cdot \left[\frac{1}{4} \cdot t^2 + \frac{t_1^2}{8 \cdot \pi^2} \cdot \left(\cos(\frac{2\pi}{t_1} \cdot t) - 1\right)\right] & , 0 \leq t \leq t_1 \\ q(0) + v_{max} \cdot t - \left(\frac{v_{max} - \dot{q}(0)}{2}\right) \cdot t_1 & , t_1 < t \leq t_2 \\ q(t_3) + v_{max} \cdot (t - t_3) + \frac{(v_{max} - \dot{q}(t_3))^2}{a_{max}} + \frac{a_{max}}{2} \cdot \\ \quad \left[\frac{(t_3 - t_2)^2}{4\pi^2} \cdot \left(1 - \cos(\frac{2\pi}{t_3 - t_2} \cdot (t - t_2))\right) - \frac{1}{2} \cdot (t - t_2)^2\right] & , t_2 < t \leq t_3 \end{cases} \tag{6.11}$$

$$\dot{q}(t) = \begin{cases} \dot{q}(0) + a_{max} \cdot \left[\frac{1}{2} \cdot t - \frac{t_1}{4\pi} \cdot \sin\left(\frac{2\pi}{t_1} \cdot t\right)\right] & , 0 \leq t \leq t_1 \\ v_{max} & , t_1 < t \leq t_2 \\ v_{max} - a_{max} \cdot \left[\frac{1}{2} \cdot (t - t_2) - \frac{t_3 - t_2}{4\pi} \cdot \sin\left(\frac{2\pi}{t_3 - t_2} \cdot (t - t_2)\right)\right] & , t_2 < t \leq t_3 \end{cases} \tag{6.12}$$

$$\ddot{q}(t) = \begin{cases} a_{max} \cdot \sin^2\left(\frac{\pi}{t_1} \cdot t\right) & , 0 \leq t \leq t_1 \\ 0 & , t_1 < t \leq t_2 \\ -a_{max} \cdot \sin^2\left(\frac{\pi}{t_3 - t_2} \cdot (t - t_2)\right) & , t_2 < t \leq t_3 \end{cases} \tag{6.13}$$

Der zu durchfahrende Weg ist $\Delta q = |q(0) - q(t_3)|$. Für $t = t_1$ ergibt sich in Gleichung (6.12) der Wert $\dot{q}(t) = v_{max}$. Es gibt drei Zeitintervalle in denen der Ruck die konstanten Werte $j_{max}, 0$ und $-j_{max}$ annimmt. Die Bewegungsdauer für die Beschleunigungszeit t_1 und der Abbremszeit $t_{32} = t_3 - t_2$ ergeben sich zu:

$$t_1 = \frac{2 \cdot (v_{max} - \dot{q}(0))}{a_{max}} \qquad t_{32} = \frac{2 \cdot (v_{max} - \dot{q}(t_3))}{a_{max}} \tag{6.14}$$

Die gesamte Verfahrzeit soll möglichst klein sein und wird gemäß Gleichung (6.15) berechnet.

$$t_3 = \frac{\Delta q}{v_{max}} + \frac{a_{max}}{4 \cdot v_{max}} \cdot \left(t_1^2 + t_{32}^2\right) \tag{6.15}$$

Bei kurzen Bewegungen, bei denen die maximale Geschwindigkeit nicht erreicht werden kann, reduziert sich die Anzahl der Zeitintervalle auf zwei. Sofern $t_2 - t_1 < 0$ gilt, gibt es keinen Abschnitt in dem mit konstanter Geschwindigkeit verfahren wird. Die reduzierte, maximale Geschwindigkeit $\bar{v}_{max}$ bei kurzen Bahnlängen berechnet sich aus den Gleichungen (6.14) und (6.15) folgendermaßen:

$$t_1 = t_2 \Longrightarrow t_1 + t_{32} = t_3 \Longrightarrow \bar{v}_{max} = \sqrt{\frac{\dot{q}^2(0) + \dot{q}^2(t_3) + \Delta q \cdot a_{max}}{2}} \tag{6.16}$$

Für synchrone Bewegungen wird zunächst die Fahrzeit für jede Achse und hieraus die Verfahrzeit t_{max} der Leitachse bestimmt. Durch Umformung der Gleichungen (6.14) und (6.15) wird die neue maximale Geschwindigkeit $\tilde{v}_{max}$ für jedes Gelenk bestimmt.

$$\tilde{v}_{max} = \frac{2 \cdot (\dot{q}(0) + \dot{q}(t_3)) + a_{max} \cdot t_{max}}{4} - \sqrt{\frac{(2 \cdot (\dot{q}(0) + \dot{q}(t_3)) + a_{max} \cdot t_{max})^2 - 8 \cdot (\dot{q}^2(0) + \dot{q}^2(t_3) + \Delta q \cdot a_{max})}{16}} \tag{6.17}$$

Damit ein vollsynchroner Bewegungsablauf entsteht, wird die maximale Geschwindigkeit und Beschleunigung der Achsen an die maximalen Verfahrzeiten der einzelnen Abschnitte angepasst. Es wird zunächst die maximale Beschleunigungszeit $t_{1,max}$, die maximale Abbremszeit $t_{32,max}$ und die maximale Zeit mit konstanter Geschwindigkeit $t_{21,max} = t_2 - t_1$ bestimmt. Hieraus wird die maximale Gesamtverfahrzeit $t_{3,max} = t_{1,max} + t_{21,max} + t_{32,max}$ ermittelt. Anhand dieser Zeitspannen wird die maximale Geschwindigkeit $\hat{v}_{max}$, die maximale Beschleunigung während der Beschleunigungsphase $\hat{a}_{1,max}$ und die maximale Beschleunigung während der Abbremsphase $\hat{a}_{32,max}$ für jede Achse bestimmt. Ausgangspunkt hierfür sind die Gleichungen (6.14) und (6.15), die entsprechend umgeformt werden.

$$\hat{v}_{max} = \frac{2 \cdot \Delta q - t_{1,max} \cdot \dot{q}(0) - t_{32,max} \cdot \dot{q}(t_3)}{2 \cdot t_{3,max} - t_{1,max} - t_{32,max}} \tag{6.18}$$

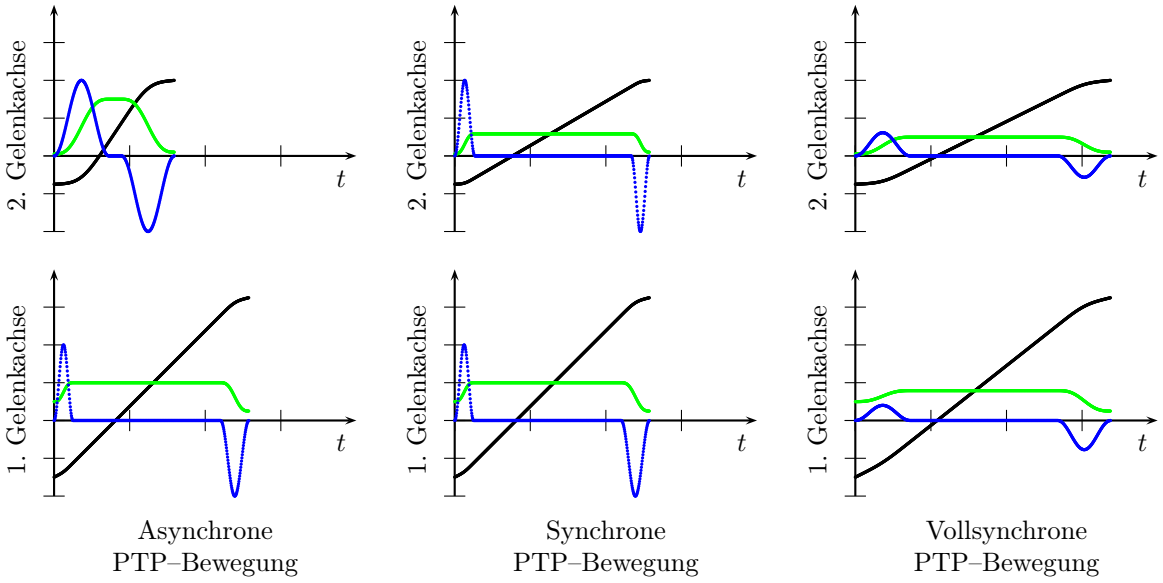
$$\hat{a}_{1,max} = \frac{2 \cdot (\hat{v}_{max} - \dot{q}(0))}{t_{1,max}} \qquad \hat{a}_{32,max} = \frac{2 \cdot (\hat{v}_{max} - \dot{q}(t_3))}{t_{32,max}} \tag{6.19}$$

In Abbildung 6.4 sind die drei Möglichkeiten für PTP–Bewegungen mit sinusförmigen Beschleunigungsprofil bei zwei Gelenkachsen einander gegenübergestellt. Dabei werden die Sollwerteverläufe von Lage, Geschwindigkeit und Beschleunigung für eine asynchrone (links), synchrone (Mitte) und vollsynchrone (rechts) PTP–Bewegung dargestellt. Wenn die Anfangsgeschwindigkeit $\dot{q}(0)$ oder Endgeschwindigkeit $\dot{q}(t_3)$ ungleich Null sind, so werden Zwischenpunkte ohne Stillstand der Gelenkachsen durchfahren und es ergibt sich ein sanfterer Übergang zwischen den einzelnen Bewegungsabschnitten. Außerdem verringert sich die Verfahrzeit, da weniger abgebremst und beschleunigt werden muss. Diese Bewegungsart wird auch als *Überschleifen* bezeichnet.

Polynomfunktionen

Für Beschleunigungsprofile können als weitere Möglichkeit auch Polynomfunktionen, zum Beispiel Polynome fünften Grades, verwendet werden.

Abbildung 6.4 PTP–Bewegungen mit sinusförmigen Beschleunigungsprofil



6.3.2 Trajektoriengenerierung im kartesischen Raum

Bei der Trajektoriengenerierung im kartesischen Raum werden die Achsen des Roboters so angesteuert, dass der Endeffektor sich exakt entlang einer vorgeschriebenen Bahn bewegt. Die beiden Hauptvarianten sind Linear– und Zirkularinterpolation. Linearinterpolation kann auch zur Umorientierung des Endeffektors bei gleichbleibender Position verwendet werden. Diese Bewegungsart wird als *Quirlen* bezeichnet. Wird zirkular interpoliert, so liegen die Interpolationspunkte in einer Ebene, die eventuell geneigt ist. Die Trajektoriengenerierung im kartesischen Raum erfordert mehr Rechenaufwand als im Gelenkwinkelraum, da in jedem Interpolationstakt mit Hilfe der inversen Kinematik die berechnete Zwischenkoordinate in den Gelenkwinkelraum transformiert werden muß.

Eine Orientierung kann als dreidimensionaler Vektor, als Rotationsmatrix oder als Quaternion repräsentiert werden. Bei der Interpolation von Orientierungen mit Euler Winkeln können Mehrdeutigkeiten entstehen. Außerdem kann bei dieser Darstellung ein sogenannter *Gimbal Lock* [66] auftreten, das heißt dass bei bestimmten Winkelkonstellationen nicht mehr um drei, sondern nur noch um zwei Achsen gedreht werden kann. Mit einer Rotationsmatrix können kontinuierliche Orientierungsänderungen nur schwierig ausgedrückt werden und es besteht die Gefahr, dass die Rotationsmatrix während der Interpolation ihre Orthonormalität verliert. Deswegen werden in dieser Arbeit für die Interpolation von Orientierungen Quaternionen eingesetzt. Quaternionen können einfach normiert werden, benötigen weniger Rechenoperationen und damit weniger Rechenzeit als Rotationsmatrizen und es treten keine Singularitäten auf [156]. Außerdem können Euler Winkel, Quaternionen und Rotationsmatrizen leicht ineinander umgewandelt werden (Abschnitt A.1).

Lineare Interpolation in kartesischen Koordinaten

Für geradlinige Bewegungen müssen der Anfangspunkt $\vec{p}_a = (p_{a,x}, p_{a,y}, p_{a,z})^T$ und der Endpunkt $\vec{p}_e = (p_{e,x}, p_{e,y}, p_{e,z})^T$ der Geraden angegeben werden (Abbildung 6.5). Die Anfangsorientierung $\hat{q}_a = (q_{a,w}, q_{a,x}, q_{a,y}, q_{a,z})^T$ und die Endorientierung $\hat{q}_e = (q_{e,w}, q_{e,x}, q_{e,y}, q_{e,z})^T$

werden als Quaternionen dargestellt. Sofern die Bahnanfangs– und der Bahnendlage die gleiche Orientierung haben, wird die Orientierung während der Interpolation konstant gehalten. Unterscheiden sich die Orientierungen, so wird die Anfangsorientierung während der Interpolation kontinuierlich in die Endorientierung überführt. Zunächst wird die Länge der abzufahrenden Translationsstrecke Δp im kartesischen Raum und die Orientierungsänderung Δq im Quaternionenraum berechnet.

$$\Delta p = \sqrt{(p_{e,x} - p_{a,x})^2 + (p_{e,y} - p_{a,y})^2 + (p_{e,z} - p_{a,z})^2} \tag{6.20}$$

$$\Delta q = \sqrt{(q_{e,w} - q_{a,w})^2 + (q_{e,x} - q_{a,x})^2 + (q_{e,y} - q_{a,y})^2 + (q_{e,z} - q_{a,z})^2} \tag{6.21}$$

Zur Berechnung der Interpolationspunkte $\vec{p}(t)$ und der Quaternionen $\hat{q}(t)$ wird der Bahnparameter $s_l(t)$ eingeführt. Um ruckfreies Beschleunigen und Abbremsen zu erreichen, wird der Bahnparameter wie bei PTP–Bewegungen interpoliert mit Startwert 0 und Endwert $\max\{\Delta p, \Delta q\}$. Die Interpolationsgleichungen zur Überführung des Anfangspunkts in den Endpunkt und zur Überführung der Anfangs– in die Endorientierung lauten:

$$\vec{p}(t) = \vec{p}_a + \frac{s_l(t)}{\max\{\Delta p, \Delta q\}} \cdot (\vec{p}_e - \vec{p}_a) \tag{6.22}$$

$$\hat{q}(t) = \hat{q}_a + \frac{s_l(t)}{\max\{\Delta p, \Delta q\}} \cdot (\hat{q}_e - \hat{q}_a) \tag{6.23}$$

Nach dieser Berechnung wird das Quaternion normiert $\hat{q}(t) = \hat{q}(t)/|\hat{q}(t)|$. Dieser Schritt ist in jedem Interpolationstakt erforderlich, damit aus diesem Einheitsquaternion die korrespondierende Rotationsmatrix bestimmt werden kann (Gleichung A.6). Aus Rotationsmatrix und Positionsvektor wird anschließend eine homogene Transformationsmatrix gebildet, die mit der inversen Kinematik in den Gelenkwinkelraum transformiert wird.

Zirkuläre Interpolation in kartesischen Koordinaten

Für kreisförmige Bewegungen ist die Angabe des Anfangspunkts $\vec{p}_a = (p_{a,x}, p_{a,y}, p_{a,z})^T$ und des Endpunkts $\vec{p}_e = (p_{e,x}, p_{e,y}, p_{e,z})^T$ der Bewegung sowie die Angabe eines zusätzlichen Hilfspunktes $\vec{p}_h = (p_{h,x}, p_{h,y}, p_{h,z})^T$ auf der Bahn erforderlich (Abbildung 6.5). Diese drei Punkte dürfen nicht identisch sein und nicht auf einer Geraden liegen. Außerdem sollte der Hilfspunkt nicht zu nahe am Anfangs– oder Endpunkt liegen. Der Hilfspunkt dient dazu, den Kreismittelpunkt $\vec{p}_m = (p_{m,x}, p_{m,y}, p_{m,z})^T$, Kurvenradius r und Normalenvektor $\vec{n}$ der Kreisebene zu definieren, in der interpoliert wird. Ein geeigneter Normalenvektor wird durch das Kreuzprodukt $\vec{n} = (\vec{p}_h - \vec{p}_a) \times (\vec{p}_e - \vec{p}_h)$ definiert. Der Kurvenradius r ergibt sich aus den Abständen $a = |\vec{p}_h - \vec{p}_a|$, $b = |\vec{p}_e - \vec{p}_h|$ und $c = |\vec{p}_e - \vec{p}_a|$ gemäß dem Satz des Heron.

$$r = \frac{a \cdot b \cdot c}{\sqrt{(a+b+c) \cdot (a+b-c) \cdot (a-b+c) \cdot (-a+b+c)}} \tag{6.24}$$

Der Kreismittelpunkt liegt im gemeinsamen Schnittpunkt der Mittelsenkrechten des von $\vec{p}_a$, $\vec{p}_h$, und $\vec{p}_e$ aufgespannten Dreiecks. Die gemeinsame Lösung der Mittelsenkrechtsgleichungen und der Kreisebenengleichung, die hier in Normalvektorform angegeben sind, bestimmen die Koordinaten des Kreismittelpunkts $\vec{p}_m$.

$$\begin{aligned} (\vec{p}_h - \vec{p}_a) \cdot \vec{p}_m &= \tfrac{1}{2} \cdot (\vec{p}_a + \vec{p}_h) \cdot (\vec{p}_h - \vec{p}_a) \\ (\vec{p}_e - \vec{p}_h) \cdot \vec{p}_m &= \tfrac{1}{2} \cdot (\vec{p}_h + \vec{p}_e) \cdot (\vec{p}_e - \vec{p}_h) \\ \vec{n} \cdot \vec{p}_m &= \vec{n} \cdot \vec{p}_a \end{aligned} \tag{6.25}$$

Zur numerischen Lösung dieses linearen Gleichungssystems wird die Cramersche Regel beziehungsweise das Determinantenverfahren [12] angewendet. Anschließend wird in den Kreismittelpunkt ein rechtshändiges orthonormiertes Koordinatensystem KS gelegt. Die x–Achse zeigt dabei vom Kreismittelpunkt $\vec{p}_m$ auf den Anfangspunkt $\vec{p}_a$ der zu interpolierenden Kreisbahn und die z–Achse entlang des Normalenvektors $\vec{n}$ der Kreisebene.

$$KS = (\vec{e}_1, \vec{e}_2, \vec{e}_3) = \left(\frac{\vec{p}_a - \vec{p}_m}{|\vec{p}_a - \vec{p}_m|}, \vec{e}_3 \times \vec{e}_1, \frac{\vec{n}}{|\vec{n}|} \right) \tag{6.26}$$

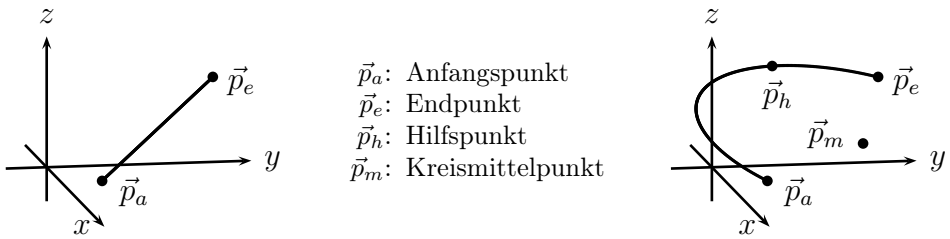
Der Mittelpunktswinkel eines Kreisbogens ist doppelt so groß wie einer der zugehörigen Umfangswinkel. Mit Hilfe des Kosinussatzes wird der zu durchfahrende Gesamtdrehwinkel $\Delta\alpha$ berechnet.

$$\Delta\alpha = 2\pi - 2 \cdot \arccos\left(\frac{a^2 + b^2 - c^2}{2 \cdot a \cdot b} \right) \tag{6.27}$$

Die zu durchfahrende Bogenlänge beträgt $\Delta s = r \cdot \Delta\alpha$. Die Berechnung der Interpolationspunkte $\vec{p}(t)$ erfolgt über den Bahnparameter $s_c(t)$, der die zurückgelegte Bogenlänge angibt. Der Bahnparameter wird wie bei PTP–Bewegungen im Gelenkwinkelraum mit Startwert 0 und Endwert Δs interpoliert. In der folgenden Interpolationsgleichung wird zunächst in den Kreismittelpunkt transformiert und anschließend in der Kreisebene die kartesischen Koordinaten berechnet.

$$\vec{p}(t) = \vec{p}_m + KS * \begin{pmatrix} r \cdot \cos\left(\frac{s_c(t)}{r}\right) \\ r \cdot \sin\left(\frac{s_c(t)}{r}\right) \\ 0 \end{pmatrix} \tag{6.28}$$

Abbildung 6.5 Lineare und zirkuläre Trajektoriengenerierung



Bei Kreisinterpolation kann die Orientierung des Endeffektors während der gesamten Interpolation entweder konstant bleiben oder variabel sein. Außerdem kann entweder raumbezogen oder bahnbezogen interpoliert werden. Der raumbezogene Fall ist analog zur Orientierungsinterpolation bei Linearbewegungen definiert.

Bei *konstanter, raumbezogener Interpolation* bleibt die Orientierung des Endeffektors bezüglich des Basiskoordinatensystems während der gesamten Bewegung konstant.

Bei *variabler, raumbezogener Interpolation* wird eine Anfangsorientierung kontinuierlich in eine Endorientierung überführt. Die Anfangs– und die Endorientierung werden als Quaternionen $\hat{q}_a = (q_{a,w}, q_{a,x}, q_{a,y}, q_{a,z})^T$ beziehungsweise $\hat{q}_e = (q_{e,w}, q_{e,x}, q_{e,y}, q_{e,z})^T$ dargestellt. Die Orientierung des Hilfspunkts, der die Kreisbahn festlegt, wird nicht benötigt. Zunächst wird die Orientierungsänderung Δq im Quaternionenraum wie in Gleichung (6.21) berechnet. Es wird der Bahnparameter $s_{c2}(t)$ mit Startwert 0 und Endwert Δq eingeführt. Die

beiden Paramter $s_c(t)$ und $s_{c2}(t)$ werden wie bei synchronen PTP–Bewegungen interpoliert, damit die Interpolation von Positionen und Orientierungen gleichzeitig beginnt und endet. Die Interpolationsgleichung zur Überführung der Anfangs– in die Endorientierung lautet:

$$\hat{q}(t) = \hat{q}_a + \frac{s_{c2}(t)}{\Delta q} \cdot (\hat{q}_e - \hat{q}_a) \tag{6.29}$$

Bei *konstanter, bahnbezogener Orientierung* wird die Orientierung bezüglich der sich ändernden Orientierung der Bahntangente interpoliert. Dadurch wird eine Bahnbewegung mit konstanter Orientierung des Endeffektors zur Bahntangente bewirkt. Das resultierende Orientierung ergibt sich durch Quaternionenmultiplikation der Anfangsorientierung $\hat{q}_a = (q_{a,w}, q_{a,x}, q_{a,y}, q_{a,z})^T$ mit dem Quaternion, das sich anhand der Kreisebene und dem über die zurückgelegte Bogenlänge definierten Drehwinkel ergibt.

$$\hat{q}(t) = \hat{q}_a * \begin{pmatrix} \cos\left(\frac{s_c(t)}{r}\right) \\ \sin\left(\frac{s_c(t)}{r}\right) \cdot \vec{n}/|\vec{n}| \end{pmatrix} \tag{6.30}$$

Bei *variabler, bahnbezogener Orientierung* wird sowohl eine Anfangs– in eine Endorientierung überführt als auch die sich ändernde Orientierung der Bahntangente berücksichtigt. Dieser Fall ist eine Kombination der beiden vorherigen Fälle.

$$\hat{q}(t) = (\hat{q}_a + \frac{s_{c2}(t)}{\Delta q} \cdot (\hat{q}_e - \hat{q}_a)) * \begin{pmatrix} \cos\left(\frac{s_c(t)}{r}\right) \\ \sin\left(\frac{s_c(t)}{r}\right) \cdot \vec{n}/|\vec{n}| \end{pmatrix} \tag{6.31}$$

6.4 Entwicklung eines Interpreters

Die Aufgabe des Interpreters ist die Entgegennahme und Abarbeitung von Roboterbefehlen und Programmen. Der Interpreter ist im Gegensatz zu den anderen Komponenten der Bewegungssteuerung völlig unabhängig von der kinematischen Struktur eines bestimmten Roboters. Ändert sich die Aufgabe eines Roboters, so wird das zu interpretierende Roboterprogramm ausgetauscht und gegebenenfalls die Steuerungssoftware rekonfiguriert.

6.4.1 Herstellerspezifische Roboterprogrammiersprachen

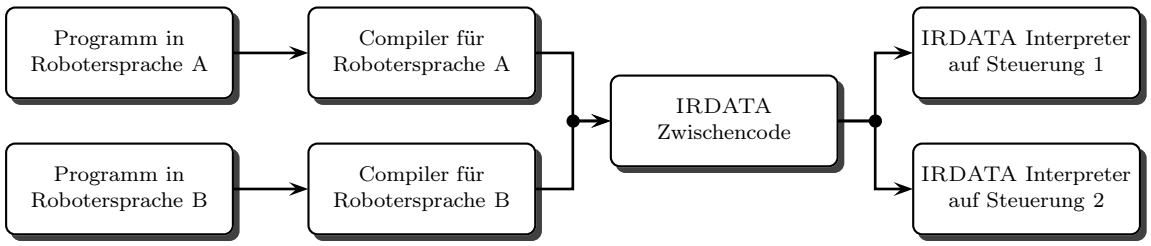
Roboterprogrammiersprachen enthalten neben den in imperativen Programmiersprachen üblichen Konstrukten zusätzlich auch Bewegungsanweisungen, Geschwindigkeits– und Beschleunigungsangaben, Anweisungen für den Betrieb von Werkzeugen und Greifern sowie zur Ansteuerung eventuell externer Gelenkachsen und Anweisungen für die Signal– und Sensordatenverarbeitung. Außerdem gibt es robotikspezifische Datentypen, wie zum Beispiel Vektoren und Transformationsmatrizen und spezielle Operationen auf diesen Datentypen, wie beispielsweise die Umrechnung von Euler–Winkeln in Rotationsmatrizen.

Während im Bereich der Werkzeugmaschinenprogrammierung die DIN 66025 [36] Verbreitung fand, konnte sich im Bereich der Roboterprogrammiersprachen bislang kein einheitlicher Standard durchsetzen. Jeder Hersteller verwendet seine eigene spezifische Sprache, wie zum Beispiel KUKA ROBOT LANGUAGE (KRL), VARIABLE ASSEMBLY LANGUAGE (VAL) oder die ABB–Roboterprogrammiersprache RAPID. Deswegen existieren heutzutage viele Roboterprogrammiersprachen. Die Sprachen der verschiedenen Hersteller unterscheiden sich sowohl in ihrem Sprachumfang als auch in der Syntax und Semantik.

6.4.2 DIN 66313: Industrial Robot Data (IRDATA)

Herstellerspezifische Sprachen können nicht für beliebige Robotertypen, sondern nur für die speziellen Roboter eines Herstellers verwendet werden. Da Programme in diesen Sprachen nur zwischen Robotern desselben Herstellers, nicht aber herstellerübergreifend ausgetauscht werden können, wurde die Schnittstellensprache IRDATA [37] vorgeschlagen. Roboterprogramme, die in verschiedenen Programmiersprachen implementiert sind, werden zuerst mit einem Compiler in diese Zwischensprache übersetzt. Anschließend wird dieser einheitliche Zwischencode von dem Interpreter auf der Robotersteuerung ausgeführt. Der Zwischencode ist ein Zahlencode, bei dem die Anweisungen aus *numerisch codierten Sätzen* bestehen. Jeder Satz besteht aus einer Folge von Wörtern, die durch Kommata voneinander getrennt werden. Das erste Wort eines Satzes ist die *Satzfolgenummer* mit der die Sätze in aufsteigender Folge numeriert werden. Das zweite Wort eines Satzes ist das *Hauptwort*, das den Satztyp beinhaltet, zum Beispiel Bewegungsbefehl, arithmetische Anweisung oder Sensorfunktion. Die restlichen Wörter eines Satzes sind *Nebenwörter*, die die erforderlichen Daten zu einem Hauptwort enthalten. Nebenwörter sind beispielsweise bei einem Bewegungsbefehl Geschwindigkeits– und Beschleunigungswerte und bei einer arithmetischen Operation sind es Operanden und deren Typen. Da es sich bei den meisten Wörtern eines Satzes um Zahlen handelt, sollten Roboterprogramme nicht von Hand in IRDATA implementiert werden, sondern in einer Hochsprache und dann mit einem Compiler nach IRDATA übersetzt werden (Abbildung 6.6). Ein IRDATA–Programm besteht aus Programmbeginn, Vereinbarungsteil, Blöcke und Programmende. In dem Vereinbarungsteil werden Prozeduren und Daten deklariert und in Blöcken, die geschachtelt sein können, sind die eigentlichen Anweisungen enthalten. Es gibt eine Reihe vordefinierter Datentypen, zum Beispiel den Vektordatentyp, der aus drei reelen Zahlen besteht. Außerdem kann der Benutzer neue Datentypen wie Felder und Verbunde einführen.

Abbildung 6.6 Programmierung über IRDATA Schnittstelle



Der Vorteil von IRDATA ist, dass Roboterprogramme und Roboter von verschiedenen Herstellern miteinander kombiniert werden können. Diese Flexibilität wird durch die zweistufige Programmverarbeitung, das heißt Compilierung eines Programms in roboterunabhängigen Zwischencode und anschließende Interpretation, erreicht. Der Nachteil von IRDATA ist, dass diese Zwischensprache sehr umfangreich und schwer zu erlernen ist. Da bisher für diese Norm keine Referenzimplementierungen erhältlich sind, ist für jede Roboterprogrammiersprache, die unterstützt werden soll, ein Compiler zu entwickeln. Zur Neuentwicklung mindestens eines Compilers und eines Interpreters muß ein hoher Aufwand betrieben werden. Deswegen wird in dieser Arbeit für die einheitliche Roboterprogrammierung eine herstellerübergreifende Sprache verwendet, da der Einarbeitungsaufwand geringer als bei IRDATA ist und kein Compiler entwickelt zu werden braucht.

6.4.3 DIN 66312: Industrial Robot Language (IRL)

Die einzige speziell für die Roboterprogrammierung normierte Programmiersprache ist die INDUSTRIAL ROBOT LANGUAGE (IRL), die in DIN 66312 [38] spezifiziert ist. Die Sprache dient der Vereinheitlichung der bisher herstellerabhängigen Roboterprogrammiersprachen. Die Ausführung von IRL–Programmen durch die Bewegungssteuerung erfordert die Entwicklung eines entsprechenden Interpreters. Darüber hinaus besteht die Möglichkeit, einen Compiler zu entwickeln, um IRL–Programme zuerst nach IRDATA (Abschnitt 6.4.2) zu übersetzen. In dieser Arbeit wurde ein IRL–Interpreter entwickelt, der die Befehle im Roboterprogramm interruptgesteuert abarbeitet und dabei Funktionsaufrufe für die Trajektoriengenerierungskomponenten erzeugt. Teile des Interpreters wurden im Rahmen einer Studienarbeit implementiert, die innerhalb dieser Arbeit betreut wurde. Mit Hilfe des Parsergenerators ANTLR (ANother Tool for Language Recognition) [132] wird die Syntaxanalyse durchgeführt, das Roboterprogramm in einen abstrakten Syntaxbaum als Zwischenform umgewandelt und dieser direkt durchlaufen und ausgewertet. Die Grammatik von IRL wurde als LL(k) beziehungsweise als linksabsteigende Grammatik formalisiert, das heißt jeder Ableitungsschritt zum Aufbau des Syntaxbaums ist eindeutig durch die nächsten k Eingabesymbole festgelegt. Es wurde eine Vorschau von $k = 2$ benutzt, da größere Werte zu längeren Programmlaufzeiten führen würden. Der für diese Grammatik erzeugte Parser verwendet die Methode des rekursiven Abstiegs, das heißt jedem Nichtterminalsymbol wird eine Funktion zugeordnet, die alle aus dem Nichtterminalsymbol ableitbaren Zeichenketten analysiert. Bei der Entwicklung des Parsers wurde berücksichtigt, dass IRL grundsätzlich *nicht* zwischen Groß– und Kleinschreibung unterscheidet.

IRL ist eine PASCAL–ähnlich Sprache und enthält zudem Konstrukte für Bewegungsanweisungen. IRL–Programme bestehen aus einem *Programmkopf*, einem *Programmvereinbarungsteil*, einer *Unterprogrammliste* und einem *Anweisungsteil*. Der Programmkopf enthält im Wesentlichen einen Programmbezeichner. Im folgenden werden die wichtigsten Konstrukte dieser Sprache, die auch der entwickelte Interpreter beherrscht, vorgestellt.

Kommentare: Die Lesbarkeit von Programmen wird durch Kommentare erhöht. Kommentare werden durch geschweifte Klammern `{ }` eingeleitet und beendet. Sie können sich über eine oder mehrere Zeilen erstrecken und dürfen auch geschachtelt werden.

Bezeichner: Vom Benutzer deklarierte Variablen und Funktionen werden Bezeichner zugeordnet. Bezeichner müssen mit einem Buchstaben oder einem Unterstrich beginnen. Daran dürfen beliebig viele Buchstaben oder Ziffern sowie Unterstriche folgen.

Schlüsselwörter: In IRL ist eine Vielzahl von Schlüsselwörtern oder reservierten Wörtern definiert (135). Diese Zeichenketten dürfen nicht als Bezeichner verwendet werden.

Grunddatentypen: IRL enthält eine Reihe vordefinierter Datentypen und erlaubt es, neue Datentypen zu definieren. Der Typ **bool** umfaßt die Wahrheitswerte *wahr* ($\neq 0$) und *falsch* ($= 0$). Der Typ **char** dient dazu, ein einzelnes ASCII–Zeichen darzustellen. Für ganze Zahlen ist der Typ **int** mit einem minimalen Wertebereich von $-2^{31} + 1$ bis $2^{31} - 1$ verfügbar. Der Typ **real** wird für Gleitpunktzahlen verwendet. Als minimaler Wertebereich wird $-1.7 \cdot 10^{38}$ bis $-1.7 \cdot 10^{-38}$ und $1.7 \cdot 10^{-38}$ bis $1.7 \cdot 10^{38}$ sowie einschließlich der Null gefordert. Mit dem Typ **string** können Zeichenketten dargestellt werden.

Geometrische Datentypen: Außer diesen Grundtypen, die auch in anderen Programmiersprachen anzutreffen sind, sind geometrische Datentypen vordefiniert mit denen die Position und Orientierung des Endeffektors beschrieben werden kann. Der Typ **position**

besteht aus drei Gleikommazahlen, um die x–, y– und z–Richtung von kartesischen Koordinaten anzugeben. Der Typ **orientation** beschreibt die Orientierung eines Koordinatensystems hinsichtlich eines Referenzkoordinatensystems. Die interne Implementierung dieses Datentyps ist nicht Teil der Norm, sondern wird dem Übersetzerbauer überlassen. Allerdings sind eine Reihe von Funktionen vordefiniert, um eine Variable dieses Typs beispielsweise anhand von Euler Winkeln zu bestimmen. Der Typ **pose** ist ein zusammengesetzter Datentyp, der aus den beiden Grundtypen **position** und **orientation** besteht.

Benutzerdefinierte Datentypen: Als benutzerdefinierte Typen sind Felder und zusammengesetzte Typen erlaubt. Felder sind Datenstrukturen, die aus einer Anzahl von Elementen desselben Typs bestehen und auf deren Elemente über einen ganzzahligen Index zugegriffen werden kann. Zusammengesetzte Typen sind Verbunde, die aus mehreren Typen bestehen. Benutzerdefinierte Typen können auch ineinander geschachtelt werden.

Programmvereinbarungsteil: Im Programmvereinbarungsteil werden Konstanten, Variablen und Typen vereinbart. Die Reihenfolge dieser Vereinbarungen ist beliebig. Durch Konstantendeklarationen wird die Lesbarkeit von Programmen erhöht. Mit Typdeklarationen werden neue Typen, das heißt zusammengesetzte Typen oder Felder, eingeführt.

Unterprogrammliste: In der Unterprogrammliste werden Prozeduren und Funktionen vereinbart. Im Gegensatz zu Prozeduren liefern Funktionen am Ende ihrer Ausführung einen Rückgabewert. Drei verschiedene Parameterübergabemechanismen werden unterstützt, und zwar Wertaufruf **in**, Ergebnisaufruf **out** und die Kombination **inout**. Beim Wertaufruf wird der Parameter mit dem Wert des Arguments initialisiert. Beim Ergebnisaufruf wird nach Beendigung des Unterprogramms der Wert des Parameters an das Argument zugewiesen. Das Argument selbst ist im Unterprogramm nicht lesbar.

Anweisungsteil: Der Anweisungsteil enthält die eigentlichen Anweisungen und Anweisungsblöcke, die geschachtelt sein können. Ein Anweisungsblock besteht aus einer einzelnen oder mehreren Anweisungen. Eine Anweisung kann eine Zuweisung, ein Unterprogrammaufruf, eine Programmflußanweisung, Ein/Ausgabeanweisung oder Bewegungsanweisung sein. Durch eine Zuweisung erhält eine Variable einen neuen Wert, sofern die Typen kompatibel sind. Mit einem Unterprogrammaufruf wird eine Prozedur oder eine Funktion gestartet. Zur Steuerung des Programmablaufs gibt es Sprunganweisungen, bedingte Verzweigungen, Schleifenkonstrukte und Rückkehranweisungen. Programmsprünge sind nur innerhalb desselben Anweisungsblocks erlaubt. Aufgrund der besonderen Klammerung tritt bei geschachtelten Verzweigungen das *Dangling–Else–Problem* nicht auf. Es gibt verschiedene Schleifenkonstrukte bei denen die Bedingung entweder am Schleifenanfang oder am Schleifenende geprüft wird und es gibt Zählschleifen bei denen eine Laufvariable die Anzahl der Schleifendurchläufe bestimmt. Funktionen müssen explizit eine Rückkehranweisung enthalten, die auch durchlaufen wird und einen Wert zurückgibt. Mit Ausgabeanweisungen können Daten auf die Konsole oder in Dateien geschrieben werden.

Bewegungsanweisungen: Als Bewegungsarten können Punkt–zu–Punkt, lineare und zirkulare Bewegungen angegeben werden. Je nachdem, ob eine Positionierung bezüglich des Basiskoordinatensystems gewünscht ist oder der Manipulator relativ zu seiner aktuellen Istposition verfahren soll, werden absolute und relative Bewegungen unterschieden. Bei Punkt–zu–Punkt und linearen Bewegungen wird die Zielposition mit einem geometrischen Ausdruck angegeben. Bei zirkularen Bewegungen werden der Hilfspunkt und der Endpunkt der Kreisbahn durch zwei geometrische Ausdrücke festgelegt. Darüber hinaus kann auch ein Bewegungspfad angegeben werden. Pfade werden durch eindimensionale

Felder oder Listen festgelegt. Bei Kreisbewegungen muß der Pfad eine gerade Anzahl von geometrischen Ausdrücken enthalten. Die ungeraden Elemente des Pfades definieren Hilfspunkte und die gerade Elemente die Endpunkte der Kreisbahnen. Die Anfangsposition wird in Bewegungsanweisungen nicht angegeben, da sie die momentane Stellung ist. Für Bewegungsanweisungen wurden folgende grammatikalische Produktionsregeln definiert.

```
MovementStatement : PTPMovement | LinearMovement | CircleMovement ';'
PTPMovement       : ('move' | 'move_inc') 'ptp'^ ToPoint PTPParameter ';'
LinearMovement    : ('move' | 'move_inc') 'lin'^ ToPoint CPParameter ';'
CircleMovement    : ('move' | 'move_inc') 'circle'^ Circle CPParameter ';'
ToPoint           : Expression | Path ';'
Circle            : Expression ',' Expression | Path ';'
```

PTP–Bewegungsparameter: Bei einer Punkt–zu–Punkt Bewegung können optional mehrere Parameter spezifiziert werden, zum Beispiel kann der aktuelle Roboter angegeben werden. Es gibt die Möglichkeiten, einen Überschleiffaktor zu definieren und Überschleifen einzuschalten, die Geschwindigkeit und Beschleunigung für jede Gelenkachse vorzugeben sowie die Bewegungsdauer oder eine Bedingung für das Ende der Bewegung anzugeben. Außerdem kann eine externe Prozedur aufgerufen werden, mit der die Bewegung weiter angepasst werden kann, beispielsweise indem Sensorsignale verarbeitet werden.

```
PTPParameter : ('act_rob' ':=' Expression)?       { aktueller Roboter }
               ( 'c_ptp' (':=' Expression)*       { Überschleiffaktor }
               | 'c_pass'                         { Überschleifen }
               | 'speed_ptp' (':=' Expression)*   { Geschwindigkeit }
               | 'acc_ptp' (':=' Expression)*     { Beschleunigung }
               | 'time' ':=' Expression           { Bewegungsdauer }
               | 'until' Expression               { Stopbedingung }
               | 'alter_by' ProcedureCall )*      { Prozeduraufruf }
```

CP–Bewegungsparameter: Bei kartesischer Bahnsteuerung können sehr viele Parameter angegeben werden, die die Art der Bewegung festlegen. Außer den Parametern, die auch bei Punkt–zu–Punkt Verbindungen zur Verfügung stehen, gibt es noch weitere Parameter. Es ist möglich, koordinierte Bewegungen hinsichtlich zusätzlicher Achsen durchführen zu lassen. Bezüglich einer Überschleifbewegung kann der Radius der Überschleifkugel sowie die Überschleifgeschwindigkeit spezifiziert werden. Die Orientierungsgeschwindigkeit bei einer Quirlbewegung kann vorgegeben werden. Darüber hinaus kann angegeben werden, dass eine Bewegung mit einer Pendelbewegung überlagert werden soll.

```
CPParameter  : ('act_rob' ':=' Expression)?       { aktueller Roboter }
               ('adax_control' ':=' Expression)?  { Zusatzachsen }
               ( 'c_cp' (':=' Expression)*        { Überschleifabstand }
               | 'c_speed' (':=' Expression)*     { Überschleifgeschw. }
               | 'speed_ori' (':=' Expression)*   { Orientierungsgeschw. }
               | 'c_pass'                         { Überschleifen }
               | 'speed' (':=' Expression)*       { Geschwindigkeit }
               | 'acc' (':=' Expression)*         { Beschleunigung }
               | 'time' ':=' Expression           { Bewegungsdauer }
               | 'until' Expression               { Stopbedingung }
               | 'wobble'                         { Pendelbewegung }
               | 'alter_by' ProcedureCall )*      { Prozeduraufruf }
```

6.4.4 Experimentelle Ergebnisse

In diesem Abschnitt werden eine Reihe von Beispielprogrammen aufgeführt, die von dem entwickelten Roboterinterpreter direkt ausgeführt werden können. Bei dem Programm 6.1 handelt es sich um ein Roboterprogramm, wie es tatsächlich bereits auf einer realen Robotersteuerung ausgeführt wurde. Das Programm besteht aus noch sehr viel mehr Bewegungsbefehlen, die aus Platzgründen hier allerdings weggelassen wurden.

Programm 6.1 Roboterprogramm zur Inspektion komplexer Teile

```
program Inspection;
begin
    move ptp pose (position(1289.57, 71.98, −595.04),  orizyx(−1.60, 87.96, 89.52));
    move ptp pose (position(1126.06, 71.53, −767.46),  orizyx(−29.11, 89.71, 62.01));
    move lin  pose (position(1126.71, 104.75, −767.31), orizyx(−28.90, 89.71, 62.22));
    writeln('Position reached.');
endprogram;
```

Programm 6.2 ist ein weiteres Roboterprogramm, um Teile von einem Transportband durch einen Roboter entnehmen und palettieren zu lassen. Auf– und Abbewegungen zum Hochheben und Absetzen eines Teils werden linear interpoliert. Sonstige Bewegungen werden PTP interpoliert. Die Initialisierung wurde zur leichteren Lesbarkeit abgekürzt.

Programm 6.2 Roboterprogramm zur Palettierung von Objekten

```
program Palette;
var  pose: TakePose, LayPose, UpPose, DownPose, SavePose;
     int: p, x, xn, y, yn;
     bool: IsConveyer, IsPalette;
     position: xdiff, ydiff;
begin { Beginn des Programms }
     Initialization();
     move ptp SavePose;
     Gripper := GripperOpen;
     while IsConveyer and IsPalette
          for y := 1 to yn { Palettenpositionen in y–Richtung }
               for x := 1 to xn { Palettenpositionen in x–Richtung }
                    move ptp UpPose;
                    move lin TakePose;
                    Gripper := GripperClose; { Entnahme vom Transportband }
                    move lin UpPose speed := 100.0;
                    move ptp DownPose;
                    move lin LayPose;
                    Gripper := GripperOpen; { Ablage auf Palette }
                    move lin DownPose speed := 100.0;
                    DownPose.pos := DownPose.pos + xdiff;
                    LayPose.pos := LayPose.pos + xdiff;
               endfor; { x–Richtung }
               DownPose.pos := DownPose.pos - (xn - 1) * xdiff;
               DownPose.pos := DownPose.pos + ydiff;
               LayPose.pos := LayPose.pos - (xn - 1) * xdiff;
               LayPose.pos := LayPose.pos + ydiff;
          endfor; { y–Richtung }
          move ptp SavePose;
     endwhile;
endprogram; { Ende des Programms }
```

6.5 Anwendungsprogrammierschnittstelle

In dieser Arbeit wird ein objektorientiertes Komponentenmodell verwendet. Die Verschaltung der einzelnen Softwarekomponenten zu der Bewegungssteuerung erfolgt gemäß eines Referenzmodells (Abschnitt 6.1). Die Kommunikation zwischen den Komponenten wird durch die Middleware unterstützt. Von allen Komponenten der Bewegungssteuerung wurden einheitliche Schnittstellen definiert hinter denen die Komponenten ihre Funktionalität kapseln. Dadurch kann eine existierende Komponente sehr leicht durch eine neue Komponente mit denselben Schnittstellen ausgetauscht werden. Der Benutzer kann auf Basis dieser Schnittstellen eigene Komponenten entwickeln und in die Bewegungssteuerung integrieren. Um eine möglichst allgemeine Softwarearchitektur zu erreichen, sind die Schnittstellen unabhängig von Betriebssystemen und Robotertypen. Für jeden Robotertyp gibt es eine andere Kinematikkomponente, die allerdings dieselben öffentlichen Schnittstellen implementiert. Der Konstruktor der Kinematikkomponente initialisiert die kinematischen Parameter mit den Werten in dem angegebenen XML–Dokument, das durch den Konfigurator (Abschnitt 6.2) erzeugt wurde. Beispielsweise werden Denavit–Hartenberg Parameter, die bisher nur symbolisch angegeben wurden, mit numerischen Werten belegt.

```
KinematicsComponent (char *fileName)
```

Diese Komponente stellt im Wesentlichen drei Methoden zur Verfügung, die in die Bewegungssteuerung eingebunden werden. Die Methode, die die direkte Kinematik berechnet, erhält als Eingabe die Gelenkkoordinaten und berechnet daraus die Lage des Endeffektors im Basiskoordinatensystem des Roboters in Form einer Transformationsmatrix.

```
virtual int directKinematics
            (const Vector &positions, Frame &tcpBase) = 0
```

Die inverse Kinematik berechnet zu einer kartesischen Lage des Endeffektors die entsprechenden Gelenkkoordinaten als Sollwerte für die Achsregler. Falls es mehrere Lösungen gibt, so wird die der aktuellen Gelenkkonfiguration am dichtesten liegende Konfiguration, die zulässig ist, ausgewählt. Die Methode gibt eine Variable zurück, mit der überprüft werden kann, ob die Berechnung erfolgreich war oder ob eine unerreichbare Pose vorliegt.

```
virtual int inverseKinematics
            (Vector &positions, const Frame &tcpBase) = 0
```

Die Methode, die die Jacobi–Matrix berechnet, erhält als Eingabe die Gelenkkoordinaten.

```
virtual int computeJacobian
            (const Vector &positions, Matrix &jacobian) = 0
```

Für jeden Robotertyp gibt es auch eine andere Dynamikkomponente. Der Konstruktor der Dynamikkomponente initialisiert die dynamischen Parameter, beispielsweise die Massen der einzelnen Manipulatorglieder mit den Werten in dem angegebenen XML–Dokument.

```
DynamicsComponent (char *fileName)
```

Die Methode, die die direkte Dynamik berechnet, erhält als Eingabe die momentanen Gelenkkoordinaten, Gelenkgeschwindigkeiten und die in den Gelenken wirkenden Käfte und Momente. Hieraus berechnet die Methode die resultierenden Gelenkbeschleunigungen.

```
virtual int directDynamics
            (const Vector &positions, const Vector &velocities,
             Vector &accelerations, const Vector &torques) = 0
```

Die Methode, die die inverse Dynamik berechnet, erhält als Eingabe die Gelenkkoordinaten, Gelenkgeschwindigkeiten und Gelenkbeschleunigungen. Sie berechnet die Gelenkkräfte und –momente, die bei Ausführung der Trajektorie auftreten.

```
virtual int inverseDynamics
            (const Vector &positions, const Vector &velocities,
             const Vector &accelerations, Vector &torques) = 0
```

Für die PTP– und CP–Trajektoriengenerierungskomponenten gibt es eine Basisklasse, die mit den Werten in dem XML–Dokument initialisiert wird. Für die Anpassung dieser Komponenten an unterschiedliche Robotertypen ist keine Änderung der Implementierung und somit auch keine Rekompilierung notwendig. Die Anpassung wird über das XML–Dokument vorgenommen, das beispielsweise die Gelenkgeschwindigkeiten enthält.

```
TrajGen (char *fileName)
```

Die PTP–Trajektoriengenerierungskomponenten generieren Trajektorien im Gelenkwinkelraum anhand eines ausgewählten Beschleunigungsprofils. Die Methoden, die asynchrone, synchrone und vollsynchrone Bewegungen generieren, erhalten als Argumente die Koordinaten und Geschwindigkeiten der einzelnen Gelenkachsen zu Beginn und am Ende der Bewegung. Die Anfangs– und Endbeschleunigungen der Gelenkachsen sind Null.

```
virtual void generateAsynchronousTrajectory
             (const Vector &initialPos, const Vector &finalVel,
              const Vector &initialVel, const Vector &finalVel) = 0

virtual void generateSynchronousTrajectory
             (const Vector &initialPos, const Vector &finalVel,
              const Vector &initialVel, const Vector &finalVel) = 0

virtual void generateFullySynchronousTrajectory
             (const Vector &initialPos, const Vector &finalVel,
              const Vector &initialVel, const Vector &finalVel) = 0
```

Die CP–Trajektoriengenerierungskomponenten generieren lineare und zirkulare Trajektorien im kartesischen Raum. Die Methode, die lineare Trajektorien generiert, erhält als Argumente, die Position und Orientierung zu Beginn und am Ende der Bewegung sowie die kartesische Geschwindigkeit und Beschleunigung. Die Orientierung wird als Quaternion angegeben. Die Methode, die zirkulare Trajektorien generiert, erhält als Argumente zusätzlich einen Hilfspunkt im kartesischen Raum, um den Kreisbogen zu definieren und eine Variable, die angibt, ob raum– oder bahnbezogen interpoliert werden soll.

```
virtual void generateLinTrajectory
             (const Vector &initialPos, const Vector &finalPos,
              const Quaternion &initialOri, const Quaternion &finalOri,
              double velocity, double acceleration) = 0

virtual void generateCircTrajectory
             (const Vector &initialPos, const Vector &finalPos,
              const Quaternion &initialOri, const Quaternion &finalOri,
              double velocity, double acceleration, int oriCha
              const Vector &auxPos) = 0
```

Darüber hinaus wurden Komponenten für das Rechnen mit Vektoren, Matrizen und Quaternionen entwickelt, die leicht für andere Robotertypen wiederverwendet werden können.

Zusammenfassung und Ausblick

Die Entwicklung einer Bewegungssteuerung für Roboter erforderte bisher eine hohe Erfahrung von Seiten der Entwickler. Ein Großteil dieser Software ist an die Geometrie des Roboters gekoppelt und kann nicht ohne Modifikationen bei anderen Robotertypen eingesetzt werden beziehungsweise muss für andere Bauarten neu entwickelt werden. Gleichzeitig haben es die Entwickler von Steuerungssoftware für Roboter mit einer Vielzahl von unterschiedlichen Robotertypen und kinematischen Aufbauarten zu tun. Die manuelle Entwicklung von Steuerungssoftware und insbesondere die Bestimmung von Robotermodellen ist ein sehr langwieriger und fehleranfälliger Vorgang. Aufgrund von Nichtlinearitäten können kinematische und dynamische Modelle nur sehr schwierig hergeleitet werden.

Ziel dieser Arbeit war die automatische Konfiguration der Bewegungssteuerung von beliebigen Industrierobotern. Insbesondere wurde keine Software entwickelt, die nur für einen speziellen Manipulator verwendet werden kann. Stattdessen wurde ein Konfigurationswerkzeug entwickelt, mit dem der Benutzer die mechanische Struktur eines Roboters mit einer einfach zu bedienenden graphischen Oberfläche komfortabel beschreiben kann. Dieser Konfigurator kann auch von Nichtexperten auf dem Gebiet der Robotik bedient werden. Zum Beispiel kann der Benutzer Gelenkachsen als translatorisch, als rotatorisch oder als passiv beziehungsweise unbeweglich definieren. Außerdem kann er kinematische Kenngrößen wie Richtungen und Abstände der einzelnen Gelenkachsen als auch dynamische Kenngrößen wie Massen und Trägheiten der einzelnen Manipulatorglieder angeben. Anhand dieser Beschreibung werden zuerst die mathematischen Modelle des Roboters automatisch generiert. Anschließend wird die Bewegungssteuerung nach dem Baukastenprinzip durch relativ einfache Kombination von Softwarekomponenten konfiguriert.

Bei der Herleitung der kinematischen und dynamischen Robotermodelle werden alle wesentlichen Schritte und Zwischenergebnisse in dem Textformatierungssystem LaTeX dokumentiert. Dadurch kann der Benutzer nachvollziehen, wie eine Lösung bestimmt wurde. Die generierten analytischen Modelle werden zudem als Softwarekomponenten in der Programmiersprache C/C++ ausgegeben. Da auch die Anzahl der erforderlichen Operationen angegeben werden, können die Laufzeiten vorhergesagt werden. Zudem werden auch etwaige Mehrfachlösungen für die inverse Kinematik bestimmt und deren Anzahl ausgegeben. Die Modelle erfüllen Echtzeitanforderungen, sind sehr effizient berechenbar sowie gut parallelisierbar. Durch die Wiederverwendbarkeit der einmal generierten Komponenten und Modelle reduziert sich zudem das Fehlerrisiko, was zu einer erheblichen Kosteneinsparung in der Softwareentwicklung führt. Bei einer Änderung der kinematischen Struktur eines Roboters, zum Beispiel bei Austausch des Werkzeugs oder bei Ausfall eines Gelenks, können die Modelle sehr leicht angepasst beziehungsweise neu hergeleitet werden.

Der Schwerpunkt dieser Arbeit lag auf der symbolischen Berechnung von Robotermodellen, da symbolische Modelle alle Lösungen beinhalten und sehr viel schneller als näherungsweise Lösungen in Echtzeit ausgewertet werden können. Probleme die es bei nume-

rischen Algorithmen gibt, wie beispielsweise Instabilität in der Nähe von Singularitäten, werden vermieden. Außerdem sind geschlossene Lösungen ebenfalls besser geeignet, um die Bahnplanung zu optimieren und Kollisionen zu erkennen, als Näherungslösungen. Bei seriellen Manipulatoren wird für die direkte Kinematik immer eine eindeutige Lösung durch symbolische Multiplikation der Denavit–Hartenberg Matrizen bestimmt.

Zur Bestimmung der inversen Kinematik werden die Gleichungen der direkten Kinematik von links und rechts mit invertierten Transformationsmatrizen multipliziert, um so weitere Gleichungen zu erhalten, die weniger unbekannte Gelenkvariablen enthalten und einfacher zu lösen sind. Auf diese Art wird bei einem sechsachsigen Roboter ein Gleichungssystem mit insgesamt 252 Gleichungen aufgestellt. Bei einfacheren Robotergeometrien werden die kinematischen Gleichungen mit wissensbasierten Methoden (Abschnitt 4.3) gelöst. Hierfür wurde eine neue Wissensbasis entwickelt, die das analytische Lösen von Gleichungen, die transzendente Funktionen enthalten, ermöglicht. Das Wissen über das Lösen von insgesamt zwölf Prototypgleichungen wurde symbolisch und in Regelform dargestellt. Regeln sind sowohl eine formale als auch eine für Menschen verständliche Form der Wissensrepräsentation. Regeln bestehen aus einem Bedingungs– und einem Aktionsteil und werden von links nach rechts abgearbeitet (Vorwärtsverkettung). Im Bedingungsteil wird das Vorhandensein von transzendenten Gleichungen geprüft und im Aktionsteil die Lösung dieser Gleichungen durchgeführt. Zur Auflösung von Konflikten bei mehreren anwendbaren Produktionsregeln sind den Regeln Prioritäten zugeordnet. Regeln, die eine eindeutige mathematische Lösung liefern, werden zuerst angewendet und erst danach Regeln, die zwei Lösungen liefern. Da bei jeder Regelanwendung die Anzahl der unbekannten Gelenkvariablen in dem Gleichungssystem verringert werden, können keine Endlosschleifen auftreten. Um eine schnelle und effiziente Ausführung der Regeln zu erzielen, wird als Inferenzmechanismus der RETE–Algorithmus eingesetzt. Dieses regelbasierte System wurde in der populären Expertensystemschale JESS (Java Expert System Shell) implementiert.

Es kann nicht jeder Robotertyp, der eine geschlossene Lösung hat, mit regelbasierten Methoden gelöst werden. Deswegen werden bei komplexeren Robotergeometrien algebraische Eliminationsmethoden, die auf der Berechnung von Gröbnerbasen beruhen, angewendet (Abschnitt 4.4). Zunächst werden die trigonometrischen Gleichungen, die untereinander gekoppelt sind, in eine Menge von Polynomgleichungen transformiert. Anschließend werden diese Polynomgleichungen in eine univariate Polynomgleichung konvertiert, sofern eine solche Gleichung existiert. Eine Gleichung mit einer einzigen Unbekannten kann in der Regel wesentlich leichter gelöst werden als die ursprünglichen Gleichungen in mehreren Unbekannten. Die trigonometrischen Gleichungen werden mit der gefundenen Lösung vereinfacht und anschließend die verbleibenden unbekannten Gelenkvariablen wiederum mit regelbasierten Methoden gelöst. Falls für einen Manipulator keine geschlossene Lösung ermittelt werden kann, so wird eine Näherungslösung angewendet.

Zur Modellierung von Manipulatoren wurden außer Denavit–Hartenberg Parametern (Abschnitt 4.1) auch Twist–Koordinaten (Abschnitt 4.5) verwendet. Diese beiden Methoden wurden gewählt, da mit beiden eine kompakte und kanonische Darstellung der Manipulatorgeometrie möglich ist. Der Vorteil von Twist–Koordinaten ist, dass nicht jeder Gelenkachse ein Koordinatensystem zugeordnet werden braucht, sondern nur das Basis– und das Endeffektorkoordinatensystem benötigt werden. Twist–Koordinaten können durch sehr einfache geometrische Betrachtungen bestimmt werden. Denavit–Hartenberg Parameter werden zudem durch Matrixmultiplikation automatisch in Twist–Koordinaten umgerechnet. Die mit diesen beiden Verfahren generierten Modelle können sehr leicht bezüg-

lich Korrektheit und Kompaktheit miteinander verglichen werden. Obwohl die Schritte zur Herleitung der direkten Kinematik und der Jacobi–Matrix bei beiden Verfahren sehr unterschiedlich sind, werden dieselben analytischen Ausdrücke generiert. Bezüglich der Herleitung der inversen Kinematik unterscheiden sich die beiden Verfahren sehr stark. Um die inverse Kinematik auf Basis von Twist–Koordinaten zu bestimmen, wird die direkte Kinematik in mehrere geometrische Teilprobleme zerlegt, deren geschlossene Lösungen in dieser Arbeit durch algebraische Umformungen hergeleitet wurden. Infolgedessen werden bei beiden Verfahren teilweise unterschiedliche analytische Lösungen für die inverse Kinematik generiert, da beispielsweise die Reihenfolge in der unbekannte Gelenkvariablen bestimmt werden, nicht immer eindeutig ist. Bei der numerischen Auswertung in einer Simulationsumgebung wurden jedoch dieselben Zahlenwerte berechnet. Deswegen dürfen diese sehr komplizierten Modelle als korrekt angenommen werden.

Nach der Bestimmung des kinematischen Robotermodells erfolgte die Bestimmung des dynamischen Modells nach dem Lagrange und dem Newton–Euler Verfahren. Eine geschlossene Lösung der inversen Dynamik ist teilweise noch schwieriger als eine geschlossene Lösung der inversen Kinematik zu bestimmen. Bereits bei sechsachsigen Robotern sind die Bewegungsgleichungen sehr umfangreich und es treten sehr viele trigonometrische Terme auf. Die manuelle Herleitung ist extrem aufwändig und nur im Prinzip möglich. Zunächst wird die inverse Dynamik auf Basis der Massen, Schwerpunktslagen und Trägheitstensoren der Manipulatorglieder symbolisch bestimmt. Um die wiederholte Berechnung der gleichen Teilausdrücke zu vermeiden und die Gleichungen numerisch effizienter auswerten zu können, werden Hilfsvariablen automatisch eingeführt. Durch Inversion der Massenmatrix und Umstellung der Gleichungen wird dann die direkte Dynamik erhalten. Die Herleitung der Bewegungsgleichungen nach Lagrange erfolgt in mehreren Schritten. Zuerst wird für jedes Gelenk die potentielle und kinetische Enerige berechnet. Durch Aufsummierung der potentiellen Energien wird der Gravitationsvektor und durch Aufsummierung der kinetischen Energien die Massenmatrix berechnet. Schließlich werden die Zentrifugal– und die Coriolismatrix durch Differentiation und Umformung der Massenmatrix gemäß den Lagrange Gleichungen berechnet. Zur Überprüfung und Validierung des dynamischen Modells werden die Gleichungen außerdem nach dem Newton–Euler Verfahren hergeleitet. In der Vorwärtsiteration werden die Gelenkachsen vom Basis– bis zum Endeffektorkoordinatensystem durchlaufen und mit dem Impuls– und Drehimpulssatz die Kräfte und Momente in den Schwerpunkten der Manipulatorglieder bestimmt. In der Rückwärtsiteration werden dann die Bewegungsgleichungen aufgestellt. Die entstehenden Differentialgleichungen werden durch symbolische Umformungen in eine Matrizendarstellung gebracht, die zu der des Lagrange Verfahrens äquivalent ist. Die analytische Lösung ist jedoch nur bei Robotern mit wenigen Freiheitsgraden möglich, da ansonsten die Gleichungen zu umfangreich werden. Deswegen wurde in dieser Arbeit das Newton–Euler Verfahren dahingehend modifiziert, dass in nun vier Iterationen kleinere Gleichungen erstellt werden. In der ersten Iteration wird der Gravitationsvektor, in der zweiten die Massenmatrix, in der dritten die Zentrifugalmatrix und in der vierten die Coriolismatrix berechnet. Durch die Aufteilung der Berechnungen auf mehrere Iterationen können auch sechsachsige Roboter analytisch mit dem Newton–Euler Verfahren gelöst werden.

Eine Hauptkomponente jeder Bewegungssteuerung ist der Interpreter für Roboterprogramme. In dieser Arbeit wurde ein roboterunabhängiger IRL–Interpreter (Industrial Robot Language) nach DIN 66312 entwickelt. Der Interpreter beherrscht alle notwendigen Konstrukte und Anweisungen, die für die Bewegungssteuerung erforderlich sind. Anhand

von Bewegungsbefehlen in einem Roboterprogramm erzeugt der IRL–Interpreter Funktionsaufrufe für die Trajektoriengenerierungskomponenten. Zur Berechnung von Trajektorien wurde eine Bibliothek von wiederverwendbaren Komponenten entwickelt. Diese Komponenten können für verschiedene Robotertypen konfiguriert und parametriert werden, ohne die Komponenten neu kompilieren zu müssen. Als Bewegungsarten werden Punkt–zu–Punkt Bewegungen und kartesische Bahnsteuerung unterstützt. Der Benutzer hat sehr viele Möglichkeiten, um diese Komponenten zu konfigurieren. Er kann zum Beispiel die maximale Geschwindigkeit und Beschleunigung sowie den maximalen Ruck für jede Gelenkachse vorgeben und zwischen trapez– und sinusförmigen Beschleunigungsprofil sowie zwischen asynchronen, synchronen oder vollsynchronen PTP–Bewegungen auswählen. Im kartesischen Raum können Geraden und Kreise mit konstanter oder variabler Orientierung interpoliert werden. Bei Kreisinterpolation kann zusätzlich die Orientierung raum– oder bahnbezogen interpoliert werden. Für die Berechnung von Orientierungsänderungen werden Quaternionen verwendet. Zuvor werden Orientierungen, die in Roboterprogrammen als Euler Winkel angegeben werden, von der Interpolationsvorbereitung in Quaternionen umgewandelt. Zu den Vorteilen von Quaternionen gehören, dass sie numerisch stabiler als Euler Winkel sind und sie weniger Rechenoperationen als Rotationsmatrizen benötigen.

Insgesamt betrachtet wird der Entwicklungszyklus von Steuerungssoftware für Manipulatoren durch diese Arbeit sehr vereinfacht. Im Vergleich zu einer Entwicklung der Software von Hand ergeben sich keine signifikanten Unterschiede hinsichtlich der Codegröße. Jedoch wird das Auftreten von Entwicklungs– und Programmierfehlern reduziert und es ergibt sich ein deutlicher Zeit– und Aufwandsgewinn. Die Ergebnisse dieser Arbeit können außer für die Steuerung auch für die Simulation und den Neuentwurf von Robotern verwendet werden. Die Resultate wurden publiziert und auf mehreren Internationalen Konferenzen und Workshops einer breiten Öffentlichkeit vorgestellt [183, 184, 185, 186, 187, 188]. Die Vorträge fanden in den begleitenden Diskussionen eine sehr positive Resonanz.

Hinsichtlich einer späteren Weiterentwicklung des Systems gibt es eine Reihe von Möglichkeiten. Die Softwarearchitektur der Bewegungssteuerung von Robotern kann flexibel um zusätzliche Komponenten und Funktionalitäten erweitert werden. Bei der Erzeugung der Bewegungsgleichungen können Reibungs– und Dämpfungskräfte berücksichtigt werden. An den Übergängen zwischen verschiedenen Interpolationsarten, zum Beispiel von Linear– nach Zirkularinterpolation, können Überschleifbereiche definiert werden. Für zeitoptimales Verfahren werden Zwischenpunkte nicht exakt angefahren, sondern schon bei Erreichen eines Überschleifbereichs zur nächsten Bewegung übergegangen. Den Interpolationskomponenten können weitere Algorithmen hinzugefügt werden, beispielsweise Algorithmen die durch vorgegebene Punkte verlaufen, wie Spline, Akima oder Bessel Interpolation oder Algorithmen, die sich Punkten nur annähern, wie Bezier, B–Spline oder NURBS Kurven (Non–Uniform–Rational–B–Splines). Außer den Komponenten zur Bewegungssteuerung (Interpolation, Kinematik, Lageregelung, ...) können auch Komponenten zur Technologiesteuerung (SPS–Funktionalitäten, Sensordatenverarbeitung, Prozesssteuerung, ...) und zur Bedienung (Visualisierung, Menüoberflächen) in eine Bibliothek integriert werden. In Zusammenarbeit mit einem Roboterhersteller können die Forschungsergebnisse genutzt werden, um reale Roboter anzusteuern. Eine mögliches Szenario ist, einen Roboter vor Ort durch relativ einfache Rekonfiguration um einen Drehtisch für die Teilepositionierung zu erweitern. Insbesondere hinsichtlich modularer rekonfigurierbarer Roboterstrukturen (Abschnitt 2.2.4) ist diese Arbeit sehr relevant.

Anhang A

Mathematischer Hintergrund

A.1 Darstellungen von Orientierungen

Die Orientierung des Endeffektors kann auf unterschiedliche Arten dargestellt werden, beispielsweise mit Euler Winkeln, Quaternionen oder Rotationsmatrizen.

A.1.1 Euler Winkel

Euler Winkel repräsentieren die Orientierung von Objekten im dreidimensionalen Raum durch ein Tripel (α, β, γ). Hierbei wird eine Orientierung als die Verkettung von drei aufeinander folgenden Rotationen dargestellt. Es gibt mehrere Konventionen, die sich in der Reihenfolge der Rotationen unterscheiden. In dieser Arbeit wird die Z–Y'–X" Konvention verwendet. Bei dieser Konvention wird zunächst mit dem Winkel α um die z–Achse des Ausgangskoordinatensystems gedreht. Es folgt eine Rotation mit dem Winkel β um die neue y–Achse und schließlich eine Rotation mit dem Winkel γ um die nach den beiden vorherigen Rotationen erhaltene x–Achse. Die Umrechnung von Euler Winkeln in eine Rotationsmatrix erfolgt durch entsprechende Matrizenmultiplikationen [157].

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \tag{A.1a}$$

$$= \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix} \tag{A.1b}$$

Für die Umrechnung einer Rotationsmatrix R in die äquivalenten Euler Winkel nach der Z–Y'–X" Konvention werden die folgenden Gleichungen verwendet.

$$R = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix} \Longrightarrow \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} \arctan_2(n_y, n_x) \\ \arctan_2(-n_z, \sqrt{n_x^2 + n_y^2}) \\ \arctan_2(o_z, a_z) \end{pmatrix} \tag{A.2}$$

Falls $n_x = n_y = 0$ beziehungsweise $o_z = a_z = 0$ gilt, so liegt eine singuläre Konfiguration vor und die Euler Winkel werden durch $(\alpha, \beta, \gamma) = (0, \pi/2, \arctan_2(o_x, o_y))$ berechnet.

A.1.2 Quaternionen

Im Jahr 1843 entdeckte der irische Mathematiker *William Rowan Hamilton* (1805–1865) die sogenannten *Quaternionen* [69], die auch zur Darstellung von Rotationen im dreidi-

mensionalen Raum verwendet werden können. Ein Quaternion ist eine hyperkomplexe Zahl, die aus einem reellen Anteil q_w und drei imaginären Anteilen q_x, q_y und q_z besteht. Die Rechenregeln für Quaternionen sind zu denen für komplexe Zahlen ähnlich. Zwei Quaternionen $\hat{q} = (q_w, q_x, q_y, q_z)^T$ und $\hat{p} = (p_w, p_x, p_y, p_z)^T$ werden wie folgt multipliziert:

$$\hat{q} * \hat{p} = \begin{pmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{pmatrix} * \begin{pmatrix} p_w \\ p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} q_w p_w - q_x p_x - q_y p_y - q_z p_z \\ q_w p_x + q_x p_w + q_y p_z - q_z p_y \\ q_w p_y + q_y p_w + q_z p_x - q_x p_z \\ q_w p_z + q_z p_w + q_x p_y - q_y p_x \end{pmatrix} \tag{A.3}$$

Die Multiplikation ist assoziativ, aber nicht kommutativ. Es gilt das Distributivgesetz. Ist der Betrag eines Quaterions $|\hat{q}| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$ gleich Eins, so spicht man von einem *Einheitsquaternion*. Die Multiplikation zweier normierter Quaternionen ergibt wiederum ein normiertes Quaternion. Ein Quaternion $\hat{q} = (q_w, q_x, q_y, q_z)^T$ lässt sich wie folgt in eine Rotation mit dem Winkel $\theta \in [0, 2\pi)$ um eine Drehachse $\vec{\omega} = (\omega_x, \omega_y, \omega_z)^T$ transformieren.

$$\theta = 2 \cdot \arccos q_w \qquad \vec{\omega} = \begin{cases} \frac{1}{\sin(\theta/2)} \cdot (q_x, q_y, q_z)^T & , \theta \neq 0 \\ \vec{0} & , \theta = 0 \end{cases} \tag{A.4}$$

Das zu einer Rotation um den Einheitsvektor $\vec{\omega} = (\omega_x, \omega_y, \omega_z)^T$ mit Drehwinkel $\theta \in [0, 2\pi)$ äquivalente Quaternion $\hat{q} = (q_w, q_x, q_y, q_z)^T$ kann ebenfalls sehr leicht berechnet werden.

$$q_w = \cos(\theta/2) \qquad (q_x, q_y, q_z)^T = \sin(\theta/2) \cdot \vec{\omega} \tag{A.5}$$

Ein Quaternion $\hat{q} = (q_w, q_x, q_y, q_z)^T$ mit $|\hat{q}| = 1$ wird wie folgt in eine äquivalente 3×3 Rotationsmatrix umgewandelt. Unter der Bedingung, dass es sich um ein Einheitsquaternion handelt, ist die entsprechende Rotationsmatrix orthonormal [60].

$$R = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_x q_y + 2q_w q_z & 2q_x q_z - 2q_w q_y \\ 2q_x q_y - 2q_w q_z & 1 - 2q_x^2 - 2q_z^2 & 2q_y q_z + 2q_w q_x \\ 2q_x q_z + 2q_w q_y & 2q_y q_z - 2q_w q_x & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix} \tag{A.6}$$

Insbesondere stellen $\hat{q}$ und $-\hat{q}$ dieselbe Rotation dar. Das hierzu inverse Problem ist die Bestimmung eines Quaternions $\hat{q} = (q_w, q_x, q_y, q_z)^T$ bei gegebener Rotationsmatrix R.

$$R = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix} \Longrightarrow \hat{q} = \begin{pmatrix} \frac{1}{2} \cdot \sqrt{1 + n_x + o_y + a_z} \\ \frac{1}{2} \cdot sign(a_y - o_z) \cdot \sqrt{1 + n_x - o_y - a_z} \\ \frac{1}{2} \cdot sign(n_z - a_x) \cdot \sqrt{1 - n_x + o_y - a_z} \\ \frac{1}{2} \cdot sign(o_x - n_y) \cdot \sqrt{1 - n_x - o_y + a_z} \end{pmatrix} \tag{A.7}$$

Ein Quaternion $\hat{q} = (q_w, q_x, q_y, q_z)^T$ mit $|\hat{q}| = 1$ wird in entsprechende Euler Winkel nach der Z–Y'–X" Konvention umgewandelt, indem zunächst das Quaternion nach Gleichung (A.6) in eine äquivalente Rotationsmatrix umgewandelt wird und anschließend die Rotationsmatrix nach Gleichung (A.2) in Euler Winkel transformiert wird.

Für die umgekehrte Umwandlung von Euler Winkeln nach der Z–Y'–X" Konvention in Quaternionen werden die drei Einheitsquaternionen $\hat{q}_1 = (\cos(\alpha/2), 0, 0, \sin(\alpha/2))^T$, $\hat{q}_2 = (\cos(\beta/2), 0, \sin(\beta/2), 0)^T$, $\hat{q}_3 = (\cos(\gamma/2), \sin(\gamma/2), 0, 0)^T$ miteinander multipliziert.

$$\hat{q} = \begin{pmatrix} \cos(\alpha/2) \cdot \cos(\beta/2) \cdot \cos(\gamma/2) + \sin(\alpha/2) \cdot \sin(\beta/2) \cdot \sin(\gamma/2) \\ \cos(\alpha/2) \cdot \cos(\beta/2) \cdot \sin(\gamma/2) - \sin(\alpha/2) \cdot \sin(\beta/2) \cdot \cos(\gamma/2) \\ \cos(\alpha/2) \cdot \sin(\beta/2) \cdot \cos(\gamma/2) + \sin(\alpha/2) \cdot \cos(\beta/2) \cdot \sin(\gamma/2) \\ \sin(\alpha/2) \cdot \cos(\beta/2) \cdot \cos(\gamma/2) - \cos(\alpha/2) \cdot \sin(\beta/2) \cdot \sin(\gamma/2) \end{pmatrix} \tag{A.8}$$

A.2 Lie Gruppen und Lie Algebren

Eine quadratische Matrix $\tilde{\omega}$ heißt antisymmetrisch oder auch schiefsymmetrisch, wenn die Matrix $-\tilde{\omega}$ und die transponierte Matrix $\tilde{\omega}^T$ gleich sind. Alle Diagonalelemente einer antisymmetrischen Matrix sind gleich Null und dadurch ist auch die Spur der Matrix Null. Zu jedem Vektor $\vec{\omega} \in \mathbb{R}^{3\times1}$ existiert eine eindeutige antisymmetrische Matrix $\tilde{\omega} \in \mathbb{R}^{3\times3}$ mit der das Kreuzprodukt durch Matrixmultiplikation berechnet werden kann. Die *Menge der antisymmetrischen Matrizen* (Dimension 3×3) wird als $so(3)$ bezeichnet.

$$so(3) = \left\{\tilde{\omega} | \tilde{\omega} \in \mathbb{R}^{3\times3}, -\tilde{\omega} = \tilde{\omega}^T\right\} \subset \mathbb{R}^{3\times3} \tag{A.9}$$

Eine Matrix wird als orthogonal bezeichnet, wenn die Spalten– und Zeilenvektoren jeweils eine orthonormierte Basis des $\mathbb{R}^n$ bilden. Die Determinante einer orthogonalen Matrix hat den Wert ±1. Die Menge der orthogonalen 3×3–Matrizen mit Determinante 1 bildet unter Matrixmultiplikation eine Gruppe, die sogenannte *spezielle orthogonale Gruppe* $SO(3)$. Das neutrale Element dieser Gruppe ist die Einheitsmatrix $I_{3\times3}$. Jede orthogonale Matrix R besitzt genau eine inverse Matrix R^{-1} die gleich der transponierten Matrix R^T ist.

$$SO(3) = \left\{R |\ R \in \mathbb{R}^{3\times3}, RR^T = R^TR = I_{3\times3}, \det(R) = 1\right\} \subset \mathbb{R}^{3\times3} \tag{A.10}$$

Die *Menge der Twists* wird als $se(3)$ bezeichnet. Die Darstellung eines Elements von $se(3)$ als sechsdimensionaler Vektor wird als Twist–Koordinate $\xi = (\vec{\nu}, \vec{\omega})^T$ bezeichnet.

$$se(3) = \left\{ \begin{pmatrix} \tilde{\omega} & \vec{\nu} \\ 0_{3\times1} & 0_{1\times1} \end{pmatrix} |\ \tilde{\omega} \in so(3), \vec{\nu} \in \mathbb{R}^{3\times1} \right\} \subset \mathbb{R}^{4\times4} \tag{A.11}$$

Eine Koordinatentransformation im dreidimensionalen Raum setzt sich aus einer Rotation und einer Translation zusammen. In einer homogenen Transformationsmatrix wird die Rotation durch eine orthogonale Matrix $R \in SO(3)$ und die Translation durch einen Richtungsvektor $\vec{p} \in \mathbb{R}^{3\times1}$ beschrieben. Die Menge der homogenen Transformationsmatrizen bildet mit der Matrixmultiplikation als Verknüpfung eine Gruppe, die auch als *spezielle euklidische Gruppe* $SE(3)$ bezeichnet wird. Neutrales Element ist die Einheitsmatrix $I_{4\times4}$.

$$SE(3) = \left\{ \begin{pmatrix} R & \vec{p} \\ 0_{3\times1} & 1_{1\times1} \end{pmatrix} |\ R \in SO(3), \vec{p} \in \mathbb{R}^{3\times1} \right\} \subset \mathbb{R}^{4\times4} \tag{A.12}$$

Bei $SO(3)$ und $SE(3)$ handelt es sich um Lie–Gruppen, bei $so(3)$ und $se(3)$ handelt es sich um die Lie–Algebren dieser Lie–Gruppen. Eine Lie–Algebra, benannt nach dem norwegischen Mathematiker Marius Sophus Lie (1842–1899), ist ein Vektorraum $\mathcal{L}$ zusammen mit einer bilinearen Abbildung $[\cdot,\cdot] : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$, welche auch als Lie–Klammer oder Lie–Produkt bezeichnet wird. Die Lie–Klammer besitzt die folgenden Eigenschaften:

i. $[\lambda x + \mu y, z] = \lambda[x, z] + \mu[y, z] \quad \forall\lambda, \mu \in \mathbb{R}, \quad \forall x, y \in \mathcal{L}$ (Linearität 1. Argument)

ii. $[x, y] = -[y, x] \quad \forall x, y \in \mathcal{L}$ (Antikommutativität)

iii. $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0 \quad \forall x, y, z \in \mathcal{L}$ (Jacobi–Identität)

Die Lie–Klammer ist auch im zweiten Argument linear. Sie ist auf $so(3)$ als einfache Matrixmultiplikation der beiden Argumente definiert $[\tilde{\omega}_1, \tilde{\omega}_2] = \tilde{\omega}_1 * \tilde{\omega}_2$ und auf $se(3)$ als:

$$[\hat{\xi}_1, \hat{\xi}_2] = \left[\begin{pmatrix} \tilde{\omega}_1 & \vec{\nu}_1 \\ 0_{3\times1} & 0_{1\times1} \end{pmatrix}, \begin{pmatrix} \tilde{\omega}_2 & \vec{\nu}_2 \\ 0_{3\times1} & 0_{1\times1} \end{pmatrix} \right] = \begin{pmatrix} \tilde{\omega}_1 * \tilde{\omega}_2 & \tilde{\omega}_1 * \vec{\nu}_2 - \tilde{\omega}_2 * \vec{\nu}_1 \\ 0_{3\times1} & 0_{1\times1} \end{pmatrix} \tag{A.13}$$

Die Bilinearität, Antikommutativität und Jacobi–Identität von dieser Lie–Klammer können leicht nachgerechnet werden. Hingegen gilt das Assoziativgesetz, wie auch im allgemeinen für Lie–Klammern, nicht. Eine Menge $\mathcal{G}$ mit einer inneren Verknüpfung $\circ : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ heißt Gruppe, wenn die Verknüpfung abgeschlossen ist, das Assoziativgesetz gilt, die Gruppe ein neutrales Element enthält und es zu jedem Element der Gruppe ein inverses Element gibt. Eine Lie–Gruppe ist eine Gruppe, die eine differenzierbare Mannigfaltigkeit ist, das heißt dass sowohl die Gruppenverknüpfung als auch die Inversenbildung jeweils differenzierbare Abbildungen sind. Den Zusammenhang zwischen einer Lie–Algebra und der zugehörigen Lie–Gruppe stellt die (Matrix–) Exponentialfunktion her (Abschnitt A.2.1). Die Matrixexponentialfunktion hat als Argument eine quadratische Matrix. Sie konvergiert für jede Matrix, ist stets invertierbar und beliebig oft differenzierbar.

A.2.1 Rotationsformel von Rodrigues

Die Rotationsformel von Olinde Rodrigues (1794–1851) stellt eine effiziente Methode dar, um diejenige Rotationsmatrix $R \in SO(3)$ zu berechnen, die zu einer Rotation um eine normierte Drehachse $\vec{\omega} = (\omega_x, \omega_y, \omega_z)^T$ mit Drehwinkel $\theta \in [0, 2\pi)$ äquivalent ist (Rechte–Hand–Regel). Dem Vektor $\vec{\omega}$ wird die schiefsymmetrische Matrix $\tilde{\omega} \in so(3)$ zugeordnet und dann das Matrixexponential mittels Potenzreihenentwicklung berechnet. Zur leichteren Lesbarkeit werden folgende Abkürzungen verwendet: $c_\theta = \cos\theta, s_\theta = \sin\theta$.

$$\begin{aligned}
\exp(\tilde{\omega}\theta) &= I_{3\times3} + \tilde{\omega}\theta + \frac{(\tilde{\omega}\theta)^2}{2!} + \frac{(\tilde{\omega}\theta)^3}{3!} + \ldots = I_{3\times3} + \tilde{\omega}s_\theta + \tilde{\omega}^2(1 - c_\theta) \qquad \text{(A.14)} \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} s_\theta + \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}^2 (1 - c_\theta) \\
&= \begin{pmatrix} c_\theta + \omega_x^2(1 - c_\theta) & \omega_x\omega_y(1 - c_\theta) - \omega_z s_\theta & \omega_y s_\theta + \omega_x\omega_z(1 - c_\theta) \\ \omega_z s_\theta + \omega_x\omega_y(1 - c_\theta) & c_\theta + \omega_y^2(1 - c_\theta) & -\omega_x s_\theta + \omega_y\omega_z(1 - c_\theta) \\ -\omega_y s_\theta + \omega_x\omega_z(1 - c_\theta) & \omega_x s_\theta + \omega_y\omega_z(1 - c_\theta) & c_\theta + \omega_z^2(1 - c_\theta) \end{pmatrix}
\end{aligned}$$

Diese Abbildung ist surjektiv. Zu jeder Rotationsmatrix $R \in SO(3)$ existiert mindestens eine antisymmetrische Matrix $\tilde{\omega} \in so(3)$ beziehungsweise eine Drehachse $\vec{\omega} \in \mathbb{R}^{3\times1}$ sowie ein Drehwinkel $\theta \in [0, 2\pi)$, so dass $R = \exp(\tilde{\omega}\theta)$ gilt. Aufgrund von Singularitäten ist diese Abbildung nicht eindeutig. Deswegen ist bei der Umrechnung einer Rotationsmatrix in eine Drehachse und einen Drehwinkel eine Fallunterscheidung erforderlich. Zunächst wird mit Gleichung (A.7) die Rotationsmatrix in ein Quaternion umgewandelt und anschließend dieses Quaternion mit Gleichung (A.4) in Achse und Winkel umgerechnet.

Anhang B

Exemplarische Konfiguration

In diesem Abschnitt wird die Bewegungssteuerung eines RRT–SCARA schrittweise konfiguriert. Die RRT–Konfiguration besteht aus zwei parallelen vertikalen Rotationsgelenken und einem dazu anti–parallel angeordneten Translationsgelenk. Die vierte Gelenkachse ist ein Rotationsgelenk zur Orientierung des Endeffektors. Tabelle B.1 gibt die DH–Parameter des Roboters wieder. Die variablen DH–Parameter sind θ_1, θ_2, d_3 und θ_4. Die restlichen DH–Parameter sind konstant und durch den mechanischen Aufbau bedingt.

Tabelle B.1 Denavit–Hartenberg Parameter eines RRT–SCARA

Gelenk i	θ_i	Offset	d_i	a_i	α_i
1	θ_1	0°	d_1	a_1	0°
2	θ_2	0°	0	a_2	180°
3	0°	0°	d_3	0	0°
4	θ_4	0°	0	0	0°

Anhand der DH–Parameter werden automatisch die äquivalenten Twist–Koordinaten bestimmt (Abschnitt 4.5.1). Die positive Drehrichtung der Rotationsgelenke im Basiskoordinatensystem des Roboters kann aus Tabelle B.2 direkt abgelesen werden. Das erste und zweite Gelenk dreht um die positive z–Achse und das vierte um die negative z–Achse. Das dritte Gelenk ist ein Translationsgelenk, das entlang der negativen z–Achse verschiebt.

Tabelle B.2 Twist Parameter des RRT–SCARA

Gelenk i	ν_1	ν_2	ν_3	ω_1	ω_2	ω_3
1	0	0	0	0	0	1
2	0	$-a_1$	0	0	0	1
3	0	0	-1	0	0	0
4	0	$a_1 + a_2$	0	0	0	-1

Für die Bestimmung des kinematischen Modells werden die Denavit–Hartenberg Matrizen (Gleichung 4.1) sowie die Matrixexponenten der Twist–Koordinaten (Gleichung 4.13) automatisch berechnet und die einzelnen Matrizen auch invertiert. Die Denavit–Hartenberg Matrizen sind etwas einfacher als die Matrixexponenten aufgebaut (Tabelle B.3).

Tabelle B.3 Transformations– und Exponentialmatrizen des RRT–SCARA

i	$T_{i-1,i}$	$\exp(\hat{\xi}_i q_i)$
1	$\begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & a_1 \cdot \cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & a_1 \cdot \sin\theta_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
2	$\begin{pmatrix} \cos\theta_2 & \sin\theta_2 & 0 & a_2 \cdot \cos\theta_2 \\ \sin\theta_2 & -\cos\theta_2 & 0 & a_2 \cdot \sin\theta_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_1 \cdot (1-\cos\theta_2) \\ \sin\theta_2 & \cos\theta_2 & 0 & -a_1 \cdot \sin\theta_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
3	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
4	$\begin{pmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos\theta_4 & \sin\theta_4 & 0 & (a_1+a_2) \cdot (1-\cos\theta_4) \\ -\sin\theta_4 & \cos\theta_4 & 0 & (a_1+a_2) \cdot \sin\theta_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Die Lage des Endeffektors, wenn sich der Manipulator in der Nullstellung befindet, beziehungsweise im allgemeinen ist durch die beiden folgenden 4×4 Matrizen gegeben.

$$T_{04}(\vec{0}) = \begin{pmatrix} 1 & 0 & 0 & a_1 + a_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad TCP = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die direkte Kinematik wird durch symbolische Multiplikation der Transformationsmatrizen (Gleichung 4.3) beziehungsweise der Matrixexponenten der Twistkoordinaten (Gleichung 4.8) bestimmt. Bei beiden Verfahren wird das folgende Modell generiert.

$$T_{04} = \begin{pmatrix} \cos(\theta_1 + \theta_2 - \theta_4) & \sin(\theta_1 + \theta_2 - \theta_4) & 0 & a_1 \cdot \cos\theta_1 + a_2 \cdot \cos(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2 - \theta_4) & -\cos(\theta_1 + \theta_2 - \theta_4) & 0 & a_1 \cdot \sin\theta_1 + a_2 \cdot \sin(\theta_1 + \theta_2) \\ 0 & 0 & -1 & d_1 - d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Als nächstes wird die inverse Kinematik bestimmt. Dafür wird das kinematische Gleichungssystem durch entsprechende Multiplikationen der Transformationsmatrizen generiert (Abbildung 4.2). Gleichungen, die keine unbekannten Gelenkvariablen enthalten, tragen nicht zur Lösung bei und werden zur Verbesserung der Übersichtlichkeit weggelassen. Sind zwei oder mehrere der Gleichungen redundant, so wird nur eine dieser Gleichungen aufgeführt. Das zu lösende Gleichungssystem besteht aus sechs Gleichungen mit einer, elf Gleichungen mit zwei und 34 Gleichungen mit drei unbekannten Gelenkvariablen.

Gleichungen mit einer Unbekannten:

$$d_3 = d_1 - p_z \quad (1)$$

$$a_z \cdot d_3 = p_z - d_1 \quad (2)$$

$$a_x \cdot \cos\theta_1 + a_y \cdot \sin\theta_1 = 0 \quad (3)$$
$$a_x \cdot \sin\theta_1 - a_y \cdot \cos\theta_1 = 0 \quad (4)$$
$$n_z \cdot \sin\theta_4 + o_z \cdot \cos\theta_4 = 0 \quad (5)$$
$$n_z \cdot \cos\theta_4 - o_z \cdot \sin\theta_4 = 0 \quad (6)$$

Gleichungen mit zwei Unbekannten:

$$a_1 \cdot \cos\theta_1 + a_2 \cdot \cos(\theta_1 + \theta_2) = p_x \quad (7)$$
$$a_1 \cdot \sin\theta_1 + a_2 \cdot \sin(\theta_1 + \theta_2) = p_y \quad (8)$$
$$n_z \cdot \cos\theta_2 \cdot \cos\theta_4 + n_z \cdot \sin\theta_2 \cdot \sin\theta_4 - o_z \cdot \cos\theta_2 \cdot \sin\theta_4 + o_z \cdot \cos\theta_4 \cdot \sin\theta_2 = 0 \quad (9)$$
$$n_z \cdot \cos\theta_2 \cdot \sin\theta_4 - n_z \cdot \cos\theta_4 \cdot \sin\theta_2 + o_z \cdot \cos\theta_2 \cdot \cos\theta_4 + o_z \cdot \sin\theta_2 \cdot \sin\theta_4 = 0 \quad (10)$$
$$a_2 \cdot n_z \cdot \cos\theta_4 - a_2 \cdot o_z \cdot \sin\theta_4 + a_z \cdot d_3 = p_z - d_1 \quad (11)$$
$$a_2 \cdot \cos\theta_2 - p_x \cdot \cos\theta_1 - p_y \cdot \sin\theta_1 = -a_1 \quad (12)$$
$$a_2 \cdot \sin\theta_2 + p_x \cdot \sin\theta_1 - p_y \cdot \cos\theta_1 = 0 \quad (13)$$
$$a_x \cdot \cos(\theta_1 + \theta_2) + a_y \cdot \sin(\theta_1 + \theta_2) = 0 \quad (14)$$
$$a_x \cdot \sin(\theta_1 + \theta_2) - a_y \cdot \cos(\theta_1 + \theta_2) = 0 \quad (15)$$
$$a_1 \cdot \cos\theta_2 - p_x \cdot \cos(\theta_1 + \theta_2) - p_y \cdot \sin(\theta_1 + \theta_2) = -a_2 \quad (16)$$
$$a_1 \cdot \sin\theta_2 - p_x \cdot \sin(\theta_1 + \theta_2) + p_y \cdot \cos(\theta_1 + \theta_2) = 0 \quad (17)$$

Gleichungen mit drei Unbekannten:

$$\cos(\theta_1 + \theta_2 - \theta_4) = n_x \quad (18)$$
$$\sin(\theta_1 + \theta_2 - \theta_4) = n_y \quad (19)$$
$$\sin(\theta_1 + \theta_2 - \theta_4) = o_x \quad (20)$$
$$\cos(\theta_1 + \theta_2 - \theta_4) = -o_y \quad (21)$$
$$\cos(\theta_1 + \theta_2) - n_x \cdot \cos\theta_4 + o_x \cdot \sin\theta_4 = 0 \quad (22)$$
$$\sin(\theta_1 + \theta_2) - n_y \cdot \cos\theta_4 + o_y \cdot \sin\theta_4 = 0 \quad (23)$$
$$\sin(\theta_1 + \theta_2) - n_x \cdot \sin\theta_4 - o_x \cdot \cos\theta_4 = 0 \quad (24)$$
$$\cos(\theta_1 + \theta_2) + n_y \cdot \sin\theta_4 + o_y \cdot \cos\theta_4 = 0 \quad (25)$$
$$a_1 \cdot \cos\theta_1 + a_2 \cdot \cos(\theta_1 + \theta_2) + a_x \cdot d_3 = p_x \quad (26)$$
$$a_1 \cdot \sin\theta_1 + a_2 \cdot \sin(\theta_1 + \theta_2) + a_y \cdot d_3 = p_y \quad (27)$$
$$n_x \cdot \cos\theta_2 \cdot \cos\theta_4 + n_x \cdot \sin\theta_2 \cdot \sin\theta_4$$
$$-o_x \cdot \cos\theta_2 \cdot \sin\theta_4 + o_x \cdot \cos\theta_4 \cdot \sin\theta_2 - \cos\theta_1 = 0 \quad (28)$$
$$n_y \cdot \cos\theta_2 \cdot \cos\theta_4 + n_y \cdot \sin\theta_2 \cdot \sin\theta_4$$
$$-o_y \cdot \cos\theta_2 \cdot \sin\theta_4 + o_y \cdot \cos\theta_4 \cdot \sin\theta_2 - \sin\theta_1 = 0 \quad (29)$$
$$n_x \cdot \cos\theta_2 \cdot \sin\theta_4 - n_x \cdot \cos\theta_4 \cdot \sin\theta_2$$
$$+o_x \cdot \cos\theta_2 \cdot \cos\theta_4 + o_x \cdot \sin\theta_2 \cdot \sin\theta_4 - \sin\theta_1 = 0 \quad (30)$$
$$n_y \cdot \cos\theta_2 \cdot \sin\theta_4 - n_y \cdot \cos\theta_4 \cdot \sin\theta_2$$
$$+o_y \cdot \cos\theta_2 \cdot \cos\theta_4 + o_y \cdot \sin\theta_2 \cdot \sin\theta_4 + \cos\theta_1 = 0 \quad (31)$$
$$a_1 \cdot \cos\theta_1 + a_2 \cdot n_x \cdot \cos\theta_4 - a_2 \cdot o_x \cdot \sin\theta_4 + a_x \cdot d_3 = p_x \quad (32)$$
$$a_1 \cdot \sin\theta_1 + a_2 \cdot n_y \cdot \cos\theta_4 - a_2 \cdot o_y \cdot \sin\theta_4 + a_y \cdot d_3 = p_y \quad (33)$$

$$n_x \cdot \cos\theta_1 + n_y \cdot \sin\theta_1 - \cos\theta_2 \cdot \cos\theta_4 - \sin\theta_2 \cdot \sin\theta_4 = 0 \quad (34)$$

$$n_x \cdot \sin\theta_1 - n_y \cdot \cos\theta_1 - \cos\theta_2 \cdot \sin\theta_4 + \cos\theta_4 \cdot \sin\theta_2 = 0 \quad (35)$$

$$o_x \cdot \cos\theta_1 + o_y \cdot \sin\theta_1 + \cos\theta_2 \cdot \sin\theta_4 - \cos\theta_4 \cdot \sin\theta_2 = 0 \quad (36)$$

$$o_x \cdot \sin\theta_1 - o_y \cdot \cos\theta_1 - \cos\theta_2 \cdot \cos\theta_4 - \sin\theta_2 \cdot \sin\theta_4 = 0 \quad (37)$$

$$n_x \cdot \cos\theta_1 \cdot \cos\theta_4 + n_y \cdot \cos\theta_4 \cdot \sin\theta_1$$
$$-o_x \cdot \cos\theta_1 \cdot \sin\theta_4 - o_y \cdot \sin\theta_1 \cdot \sin\theta_4 - \cos\theta_2 = 0 \quad (38)$$

$$n_x \cdot \cos\theta_4 \cdot \sin\theta_1 - n_y \cdot \cos\theta_1 \cdot \cos\theta_4$$
$$-o_x \cdot \sin\theta_1 \cdot \sin\theta_4 + o_y \cdot \cos\theta_1 \cdot \sin\theta_4 + \sin\theta_2 = 0 \quad (39)$$

$$n_x \cdot \cos\theta_1 \cdot \sin\theta_4 + n_y \cdot \sin\theta_1 \cdot \sin\theta_4$$
$$+o_x \cdot \cos\theta_1 \cdot \cos\theta_4 + o_y \cdot \cos\theta_4 \cdot \sin\theta_1 - \sin\theta_2 = 0 \quad (40)$$

$$n_x \cdot \sin\theta_1 \cdot \sin\theta_4 - n_y \cdot \cos\theta_1 \cdot \sin\theta_4$$
$$+o_x \cdot \cos\theta_4 \cdot \sin\theta_1 - o_y \cdot \cos\theta_1 \cdot \cos\theta_4 - \cos\theta_2 = 0 \quad (41)$$

$$a_2 \cdot \cos\theta_2 + a_x \cdot d_3 \cdot \cos\theta_1 + a_y \cdot d_3 \cdot \sin\theta_1 - p_x \cdot \cos\theta_1 - p_y \cdot \sin\theta_1 = -a_1 \quad (42)$$

$$a_2 \cdot \sin\theta_2 - a_x \cdot d_3 \cdot \sin\theta_1 + a_y \cdot d_3 \cdot \cos\theta_1 + p_x \cdot \sin\theta_1 - p_y \cdot \cos\theta_1 = 0 \quad (43)$$

$$n_x \cdot \cos(\theta_1 + \theta_2) \cdot \cos\theta_4 + n_y \cdot \cos\theta_4 \cdot \sin(\theta_1 + \theta_2)$$
$$-o_x \cdot \cos(\theta_1 + \theta_2) \cdot \sin\theta_4 - o_y \cdot \sin(\theta_1 + \theta_2) \cdot \sin\theta_4 = 1 \quad (44)$$

$$n_x \cdot \cos\theta_4 \cdot \sin(\theta_1 + \theta_2) - n_y \cdot \cos(\theta_1 + \theta_2) \cdot \cos\theta_4$$
$$-o_x \cdot \sin(\theta_1 + \theta_2) \cdot \sin\theta_4 + o_y \cdot \cos(\theta_1 + \theta_2) \cdot \sin\theta_4 = 0 \quad (45)$$

$$n_x \cdot \cos(\theta_1 + \theta_2) \cdot \sin\theta_4 + n_y \cdot \sin(\theta_1 + \theta_2) \cdot \sin\theta_4$$
$$+o_x \cdot \cos(\theta_1 + \theta_2) \cdot \cos\theta_4 + o_y \cdot \cos\theta_4 \cdot \sin(\theta_1 + \theta_2) = 0 \quad (46)$$

$$n_x \cdot \sin(\theta_1 + \theta_2) \cdot \sin\theta_4 - n_y \cdot \cos(\theta_1 + \theta_2) \cdot \sin\theta_4$$
$$+o_x \cdot \cos\theta_4 \cdot \sin(\theta_1 + \theta_2) - o_y \cdot \cos(\theta_1 + \theta_2) \cdot \cos\theta_4 = 1 \quad (47)$$

$$n_x \cdot \cos(\theta_1 + \theta_2) + n_y \cdot \sin(\theta_1 + \theta_2) - \cos\theta_4 = 0 \quad (48)$$

$$n_x \cdot \sin(\theta_1 + \theta_2) - n_y \cdot \cos(\theta_1 + \theta_2) - \sin\theta_4 = 0 \quad (49)$$

$$o_x \cdot \cos(\theta_1 + \theta_2) + o_y \cdot \sin(\theta_1 + \theta_2) + \sin\theta_4 = 0 \quad (50)$$

$$o_x \cdot \sin(\theta_1 + \theta_2) - o_y \cdot \cos(\theta_1 + \theta_2) - \cos\theta_4 = 0 \quad (51)$$

Um so mehr DH–Parameter von Null verschieden sind, umso komplexer werden die Gleichungen und umso schwieriger ist es, das inverse kinematische Modell zu bestimmen. Das kinematische Gleichungssystem dieses Manipulators kann jedoch geschlossen gelöst werden. Durch Pattern Matching mit der Wissensbasis wird aus diesen Gleichungen die folgenden Lösungen für die unbekannten Gelenkvariablen automatisch bestimmt. Zur Vermeidung redundanter Mehrfachberechnungen wird dabei die Hilfsvariable h eingeführt.

$$h = (-a_1^2 + a_2^2 - p_x^2 - p_y^2)/(2 \cdot a_1)$$

$$\theta_1{}^{(1,2)} = \arctan_2\left(h, \pm\sqrt{p_x^2 + p_y^2 - h^2}\right) - \arctan_2(-p_x, -p_y)$$

$$\theta_2 = \arctan_2(-p_x \cdot \sin\theta_1 + p_y \cdot \cos\theta_1, -a_1 + p_x \cdot \cos\theta_1 + p_y \cdot \sin\theta_1)$$

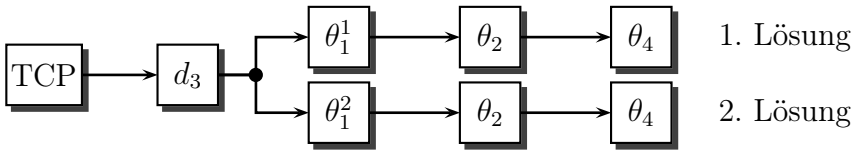
$$d_3 = d_1 - p_z$$

$$\theta_4 = \theta_1 + \theta_2 - \arctan_2(n_y, n_x)$$

Wegen des ±–Zeichens in der Gleichung für θ_1, gibt es insgesamt zwei Lösungen. Diese Mehrfachlösung kann beispielsweise dazu genutzt werden, ein Hindernis zu umgreifen. Die

Reihenfolge der Berechnungen, um die beiden möglichen Konfigurationen dieses Roboters zu bestimmen, ist in Abbildung B.1 dargestellt. Die eindeutige Lösung für d_3 und die beiden Lösungen für θ_1 werden nur in Abhängigkeit der Endeffektorlage (engl.: Tool Center Point, TCP) bestimmt. Nachdem diese Lösungen bekannt sind, werden die Lösungen für die unbekannte Gelenkvariable θ_2 in Abhängigkeit des TCP und den zwei Lösungen für die erste Gelenkvariable bestimmt. Die Lösungen für θ_4 werden auf die gleiche Weise in Abhängigkeit des TCP und den bereits bestimmten Lösungen für θ_1 und θ_2 ermittelt.

Abbildung B.1 Auflösungsfluß bei der Berechnung der inversen Kinematik



Zur Ermittlung singulärer Konfigurationen wird die Jacobi–Matrix $\mathcal{J}$ des Manipulators analytisch aufgestellt und die Gelenkkonfigurationen bestimmt, bei denen die Jacobi–Matrix singulär ist. Die Herleitung erfolgt sowohl über Denavit–Hartenberg Parameter als auch über Twist–Koordinaten. Beide Verfahren führen zu genau derselben Matrix.

$$\mathcal{J} = \begin{pmatrix} -a_1 \cdot \sin\theta_1 - a_2 \cdot \sin(\theta_1+\theta_2) & -a_2 \cdot \sin(\theta_1+\theta_2) & 0 & 0 \\ a_1 \cdot \cos\theta_1 + a_2 \cdot \cos(\theta_1+\theta_2) & a_2 \cdot \cos(\theta_1+\theta_2) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

Alle singulären Konfigurationen liegen bei diesem Roboter auf dem Rand des Arbeitsraumes. Wenn der Arm vollständig ausgestreckt beziehungsweise eingefaltet ist, reduziert sich die Anzahl der Richtungen, in denen der Endeffektor bewegt werden kann.

$$\det(\mathcal{J}) = a_1 \cdot a_2 \cdot \sin\theta_2 \stackrel{!}{=} 0 \Longleftrightarrow \theta_2 = 0^\circ + k \cdot 180^\circ$$

Als nächstes wird das dynamische Modell des SCARA in symbolischer Darstellung automatisch generiert. Für die Bestimmung der Bewegungsgleichungen werden neben den DH–Parametern auch die konzentrierten Massen, die Schwerpunktslagen und die Trägheitstensoren von allen Manipulatorgliedern benötigt (Tabelle B.4).

Tabelle B.4 Tabelle mit Parametern für die Roboterdynamik

Glied i	Masse	Schwerpunktslagen			Trägheitstensoren					
1	m_1	rx_1	0	rz_1	Ixx_1	Iyy_1	Izz_1	Ixy_1	Ixz_1	Iyz_1
2	m_2	rx_2	0	rz_2	Ixx_2	Iyy_2	Izz_2	Ixy_2	Ixz_2	Iyz_2
3	m_3	0	0	rz_3	Ixx_3	Iyy_3	Izz_3	Ixy_3	Ixz_3	Iyz_3
4	m_4	0	0	rz_4	Ixx_4	Iyy_4	Izz_4	Ixy_4	Ixz_4	Iyz_4

Die dynamischen Gleichungen des Manipulators werden zuerst nach dem Lagrange Verfahren bestimmt (Abschnitt 5.1). Für diesen speziellen Manipulator ist die einzige Variable,

die in der Massen–, Coriolis– und Zentrifugalmatrix vorkommt, der Gelenkwinkel θ_2. Der Gravitationsvektor ist konstant, da er nicht von den Werten der Gelenkvariablen abhängt. Die Bewegungsgleichungen lauten in Matrizenschreibweise:

$$\begin{pmatrix}\tau_1\\ \tau_2\\ \tau_3\\ \tau_4\end{pmatrix} = \begin{pmatrix}\mathcal{M}_{11} & \mathcal{M}_{12} & 0 & \mathcal{M}_{14}\\ \mathcal{M}_{12} & \mathcal{M}_{22} & 0 & \mathcal{M}_{24}\\ 0 & 0 & \mathcal{M}_{33} & 0\\ \mathcal{M}_{14} & \mathcal{M}_{24} & 0 & \mathcal{M}_{44}\end{pmatrix} * \begin{pmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\\ \ddot{d}_3\\ \ddot{\theta}_4\end{pmatrix} + \begin{pmatrix}\mathcal{CO}_{11} & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix} * \begin{pmatrix}\dot{\theta}_1\dot{\theta}_2\\ \dot{\theta}_1\dot{d}_3\\ \vdots\\ \dot{d}_3\dot{\theta}_4\end{pmatrix}$$
$$+ \begin{pmatrix}0 & \mathcal{CF}_{12} & 0 & 0\\ \mathcal{CF}_{21} & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix} * \begin{pmatrix}\dot{\theta}_1\dot{\theta}_1\\ \dot{\theta}_2\dot{\theta}_2\\ \dot{d}_3\dot{d}_3\\ \dot{\theta}_4\dot{\theta}_4\end{pmatrix} + \begin{pmatrix}0\\ 0\\ \mathcal{G}_3\\ 0\end{pmatrix}$$

$$\begin{aligned}
\mathcal{M}_{11} &= Izz_1 + Izz_2 + Izz_3 + Izz_4 + a_2^2 \cdot (m_2 + m_3 + m_4) + 2 \cdot a_2 \cdot m_2 \cdot rx_2 + m_2 \cdot rx_2^2\\
&\quad + a_1^2 \cdot (m_1 + m_2 + m_3 + m_4) + 2 \cdot a_1 \cdot m_1 \cdot rx_1 + m_1 \cdot rx_1^2\\
&\quad + 2 \cdot a_1 \cdot a_2 \cdot \cos\theta_2 \cdot (m_2 + m_3 + m_4) + 2 \cdot a_1 \cdot m_2 \cdot rx_2 \cdot \cos\theta_2\\
\mathcal{M}_{12} &= Izz_2 + Izz_3 + Izz_4 + a_2^2 \cdot (m_2 + m_3 + m_4) + 2 \cdot a_2 \cdot m_2 \cdot rx_2 + m_2 \cdot rx_2^2\\
&\quad + a_1 \cdot a_2 \cdot \cos\theta_2 \cdot (m_2 + m_3 + m_4) + a_1 \cdot m_2 \cdot rx_2 \cdot \cos\theta_2\\
\mathcal{M}_{22} &= Izz_2 + Izz_3 + Izz_4 + a_2^2 \cdot (m_2 + m_3 + m_4) + 2 \cdot a_2 \cdot m_2 \cdot rx_2 + m_2 \cdot rx_2^2\\
\mathcal{M}_{33} &= m_3 + m_4\\
\mathcal{M}_{14} &= -Izz_4\\
\mathcal{M}_{24} &= -Izz_4\\
\mathcal{M}_{44} &= Izz_4\\
\mathcal{CO}_{11} &= -2 \cdot a_1 \cdot a_2 \cdot \sin\theta_2 \cdot (m_2 + m_3 + m_4) - 2 \cdot a_1 \cdot m_2 \cdot rx_2 \cdot \sin\theta_2\\
\mathcal{CF}_{21} &= a_1 \cdot a_2 \cdot \sin\theta_2 \cdot (m_2 + m_3 + m_4) + a_1 \cdot m_2 \cdot rx_2 \cdot \sin\theta_2\\
\mathcal{CF}_{12} &= -a_1 \cdot a_2 \cdot \sin\theta_2 \cdot (m_2 + m_3 + m_4) - a_1 \cdot m_2 \cdot rx_2 \cdot \sin\theta_2\\
\mathcal{G}_3 &= -(m_3 + m_4) \cdot g
\end{aligned}$$

Außerdem wird das vollständige Modell der Roboterdynamik nach dem Newton–Euler Verfahren (Abschnitt 5.2) hergeleitet. Die generierten Differentialgleichungen entsprechen denen des Lagrange Verfahrens und sind zu der Matrizendarstellung äquivalent.

$$\begin{aligned}
\tau_1 &= \mathcal{M}_{11} \cdot \ddot{\theta}_1 + \mathcal{M}_{12} \cdot \ddot{\theta}_2 + \mathcal{M}_{14} \cdot \ddot{\theta}_4 + \mathcal{CO}_{11} \cdot \dot{\theta}_1\dot{\theta}_2 + \mathcal{CF}_{12} \cdot \dot{\theta}_2\dot{\theta}_2\\
\tau_2 &= \mathcal{M}_{12} \cdot \ddot{\theta}_1 + \mathcal{M}_{22} \cdot \ddot{\theta}_2 + \mathcal{M}_{24} \cdot \ddot{\theta}_4 + \mathcal{CF}_{21} \cdot \dot{\theta}_1\dot{\theta}_1\\
\tau_3 &= \mathcal{M}_{33} \cdot \ddot{d}_3 + \mathcal{G}_3\\
\tau_4 &= \mathcal{M}_{14} \cdot \ddot{\theta}_1 + \mathcal{M}_{24} \cdot \ddot{\theta}_2 + \mathcal{M}_{44} \cdot \ddot{\theta}_4
\end{aligned}$$

Anhang C

XML–Beschreibung von Robotern

Die einheitliche Beschreibung der mechanischen Struktur von Manipulatoren erfolg in XML (extensible Markup Language). Unter anderem enthält diese Beschreibung Angaben über Anzahl und Typen der Gelenkachsen eines Roboters. Da die manuelle Erstellung der Manipulatorbeschreibung unkomfortabel und fehleranfällig ist, wurde hierfür ein graphisches Konfigurationswerkzeug entwickelt. Listing C.1 zeigt das XML–Schema mit dem die roboterspezifischen XML–Dokumente validiert werden. Beispielsweise wird beim Einlesen eines XML–Dokuments in das Konfigurationswerkzeug sichergestellt, dass die Datei wohlgeformt ist und der Aufbau der vorgegebenen Struktur des XML–Schemas entspricht. Nur in diesem Fall kann der Benutzer das Dokument weiterverarbeiten oder ändern. Bei der Generierung der Robotermodelle und der Konfiguration der Bewegungssteuerung wird ebenfalls zuerst das XML–Dokument anhand des Schemas validiert.

Listing C.1: XML–Schema der Manipulatorbeschreibung

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
  <xs:element name="robot-configuration">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="kinematic-structure" type="xs:string"/>
        <xs:element name="number-of-joints" type="xs:positiveInteger"/>
        <xs:element name="application-areas">
          <xs:element name="task" type="xs:string" minOccurs="0" maxOccurs=
              "20"/>
        </xs:element>
        <xs:sequence minOccurs="2" maxOccurs="20">
          <xs:element name="joint">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="joint-type" type="xs:string"/>
                <xs:element name="joint-angle" type="xs:string"/>
                <xs:element name="joint-offset" type="xs:string"/>
                <xs:element name="link-offset" type="xs:string"/>
                <xs:element name="link-length" type="xs:string"/>
                <xs:element name="link-twist" type="xs:string"/>
                <xs:element name="theta-min" type="xs:string"/>
                <xs:element name="theta-max" type="xs:string"/>
                <xs:element name="velocity-max" type="xs:string"/>
                <xs:element name="acceleration-max" type="xs:string"/>
                <xs:element name="jerk-max" type="xs:string"/>
```

```
            <xs:element name="force-max" type="xs:string"/>
            <xs:element name="rotation-translation-axis" type="
                xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="adjacent-frame-arrangement">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="direction" type="xs:string"/>
            <xs:element name="displacement" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:element name="link" minOccurs="2" maxOccurs="20">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="mass" type="xs:string"/>
          <xs:element name="rx"  type="xs:string"/>
          <xs:element name="ry"  type="xs:string"/>
          <xs:element name="rz"  type="xs:string"/>
          <xs:element name="Ixx" type="xs:string"/>
          <xs:element name="Iyy" type="xs:string"/>
          <xs:element name="Izz" type="xs:string"/>
          <xs:element name="Ixy" type="xs:string"/>
          <xs:element name="Ixz" type="xs:string"/>
          <xs:element name="Iyz" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="trajectory-generation">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="joint-space-trajectories">
            <xs:element name="motion" type="xs:string" minOccurs="0"
                maxOccurs="1"/>
          </xs:element>
          <xs:element name="cartesian-space-trajectories">
            <xs:element name="motion" type="xs:string" minOccurs="0"
                maxOccurs="10"/>
          </xs:element>
          <xs:element name="acceleration-profile" type="xs:string"/>
          <xs:element name="interpolation-clock" type="
              xs:positiveInteger"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Literaturverzeichnis

[1] E. L. Allgower and K. Georg. *Introduction to Numerical Continuation Methods*, volume 45 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pennsylvania, USA, 2003.

[2] M. Anantharaman and M. Hiller. Numerical simulation of mechanical systems using methods for differential–algebraic equations. *International Journal for Numerical Methods in Engineering*, 32(8):1531–1542, 1991.

[3] S. Anton. Inverse Kinematik am Robotersimulationsprogramm EASY–ROB. Scientific Reports 4, Institut für Automatisierungstechnik, Hochschule Mittweida (FH), Mittweida, Germany, October 2004.

[4] G. Ars, J.-C. Faugère, H. Imai, M. Kawazoe, and M. Sugita. Comparison between xl and groebner basis algorithms. In P. J. Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 338–353. Springer, 2004.

[5] M. Attolico and P. Masarati. A multibody user–space hard real–time environment for the simulation of space robots. In *Proceedings of the Fifth Real–Time Linux Workshop*, Valencia, Spain, November 2003.

[6] R. S. Ball. *A treatise on the theory of screws*. Cambridge University Press, London, UK, 1900.

[7] B. Barenbrug. *Designing a class library for interactive simulation of rigid body dynamics*. PhD thesis, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, April 2000.

[8] D. Batory. The LEAPS algorithm. Technical report, The University of Texas at Austin, Department of Computer Sciences, Austin, Texas, USA, 1994.

[9] D. S. Batory, J. Thomas, and M. Sirkin. Reengineering a complex application using a scalable data structure compiler. In D. S. Wile, editor, *Proceedings of the 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 111–120, New Orleans, Louisiana, USA, December 1994. ACM Press.

[10] R. M. Bhatt and V. Krovi. DynaFlexPro for Maple. *IEEE Control Systems Magazine*, 26(6):127–138, December 2006.

[11] R. W. Brockett. Robotic manipulators and the product of exponentials formula. In P. A. Fuhrman, editor, *Mathematical theory of networks and systems*, volume 58, pages 120–129. Springer, New York, USA, 1984.

[12] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, Germany, 6. edition, 2005.

[13] H. Bruyninckx. Open robot control software: the OROCOS project. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 2523–2528, Seoul, Korea, May 2001.

[14] H. Bruyninckx. A free software framework for advanced robot control. In *Proceedings of the 7th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA 2002)*, Noordwijk, The Netherlands, November 2002.

[15] H. Bruyninckx, P. Soetens, and B. Koninckx. The real–time motion control core of the OROCOS project. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 2766–2771, Taipei, Taiwan, September 2003.

[16] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965.

[17] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of groebner bases. In E. W. Ng, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 3–21, Marseille, France, June 1979.

[18] B. Buchberger. Groebner–bases: An algorithmic method in polynomial ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory – Progress, Directions and Open Problems in Multidimensional Systems*, chapter 6, pages 184–232. Reidel Publishing Company, 1985.

[19] B. Buchberger. Applications of gröbner bases in non–linear computational geometry. In J.R. Rice, editor, *Mathematical Aspects of Scientific Software*, volume 14 of *IMA Volume in Mathematics and its Applications*, pages 59–87. Springer–Verlag, 1988.

[20] B. Buchberger. An implementation of gröbner bases in mathematica. Technical Report 90–58, RISC–Linz, Johannes Kepler University, Linz, Austria, 1990.

[21] B. Buchberger. Groebner bases in mathematica: Enthusiasm and frustration. In P. W. Gaffney and E. N. Houstis, editors, *Proceedings of the IFIP Working Conference on Programming Environments for High–Level Symbolic Computation*, pages 80–91, Karlsruhe, Germany, September 1991.

[22] S. Campbell, J. P. Chancelier, and R. Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer Verlag, New York, USA, 2005.

[23] I-M. Chen and Y. Gao. Closed–form inverse kinematics solver for reconfigurable robots. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 2395–2400, Seoul, Korea, May 2001.

[24] F. S. Cheng. A methodology for developing robotic workcell simulation models. In J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference*, pages 1265–1271, Orlando, Florida, USA, December 2000.

[25] C. L. Collins, J. M. McCarthy, A. Perez, and H. Su. The structure of an extensible java applet for spatial linkage synthesis. *Journal of Computing and Information Science in Engineering*, 2(1):45–49, March 2002.

[26] P. I. Corke. A computer tool for simulation and analysis: the robotics toolbox for MATLAB. In *Proceedings of the 1995 National Conference of the Australian Robot Association*, pages 319–330, Melbourne, Australia, July 1995.

[27] P. I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, March 1996.

[28] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology – EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques*, pages 392–407, Bruges, Belgium, May 2000.

[29] N. T. Courtois and J. Patarin. About the XL algorithm over $GF(2)$. In M. Joye, editor, *Proceedings of Topics in Cryptology, Cryptographers' Track at the RSA Conference (CT–RSA)*, volume 2612 of *Lecture Notes in Computer Science*, pages 141–157, San Francisco, California, USA, April 2003.

[30] D. A. Cox. Introduction to gröbner bases. In D. A. Cox and B. Sturmfels, editors, *Proceedings of the 53th Symposium in Applied Mathematics (PSAM)*, volume 53, pages 1–24. American Mathematical Society, 1998.

[31] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Prentice Hall, 3rd edition, 2004.

[32] R. Davies. Writing a matrix package in C++. In *Proceedings of the second annual object–oriented numerics conference (OON–SKI)*, pages 207–213, April 1994.

[33] K. Demura. *Robot Simulation – Robot programming with Open Dynamics Engine*. Morikita Publishing Co. Ltd., Tokyo, Japan, 2007.

[34] J. Denavit and R. S. Hartenberg. A kinematic notation for lower–pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, 22:215–221, June 1955.

[35] J. Denavit and R. S. Hartenberg. *Kinematic Synthesis of Linkages*. McGraw–Hill, New York, USA, 1964.

[36] Deutsches Institut für Normung e.V., editor. *DIN 66 025: Programmaufbau für numerisch gesteuerte Arbeitsmaschinen*. Beuth, Berlin, Germany, 1988.

[37] Deutsches Institut für Normung e.V., editor. *DIN 66 313: Industrial Robot Data (IRDATA)*. Beuth, Berlin, Germany, 1991.

[38] Deutsches Institut für Normung e.V., editor. *DIN 66 312: Programmiersprache Industrial Robot Language (IRL)*. Beuth, Berlin, Germany, 1993.

[39] Deutsches Institut für Normung e.V., editor. *DIN EN 775 (ISO 10218): Industrieroboter – Sicherheit*. Beuth, Berlin, August 1993.

[40] R. Dillmann and M. Huck. *Informationsverarbeitung in der Robotik*. Springer Verlag, Berlin, Germany, 1991.

[41] W. Dong, H. Li, and X. Teng. Off–line programming of spot–weld robot for car–body in white based on robcad. In *Proceedings of the IEEE International Conference on Mechatronics and Automation*, pages 763–768, Harbin, China, August 2007.

[42] H. Elmquist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.

[43] H. Elmqvist. Object–oriented modeling and automatic formula manipulation in Dymola. In *Proceedings of the 35th Simulation Conference (SIMS)*, Kongsberg, Norway, June 1993. Scandinavian Simulation Society.

[44] H. Elmqvist and D. Brück. Composite constructs for object–oriented modeling. In F. Breitenecker and I. Husinsky, editors, *Proceedings of the Eurosim Simulation Congress*, Vienna, Austria, September 1995. Elsevier Publishing Company.

[45] G. Erdõs. *A New Symbolic Approach for the Geometric, Kinematic and Dynamic Modeling of Industrial Robots*. PhD thesis, Hungarian Academy of Sciences, Budapest, Hungary, 2000.

[46] G. Erdõs. *LinkageDesigner: User Manual*. Budapest, Hungary, August 2005.

[47] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, June 1999.

[48] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In T. Mora, editor, *ISSAC 2002: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83, Lille, France, July 2002. ACM Press.

[49] J.-C. Faugère and L. Perret. Polynomial equivalence problems: Algorithmic and theoretical aspects. In *Advances in Cryptology – EUROCRYPT 2006*, pages 30–47. Springer, 2006.

[50] P. Fisette, T. Postiau, L. Sass, and J.C. Samin. Fully symbolic generation of complex multibody models. *Mechanics of Structures and Machines*, 30(1):31–82, 2002.

[51] P. Fisette and J.-C. Samin. Robotran: symbolic generation of multibody system dynamic equations. In W. Schiehlen, editor, *Advanced Multibody System Dynamics, Simulation and Software Tools*, pages 373–378. Kluwer Academic Publishers, Dordrecht, Boston, London, 1993.

[52] C. L. Forgy. *On the Efficient Implementation of Production Systems*. PhD thesis, Computer Science Department, Carnegie–Mellon University, Pittsburgh, Pennsylvania, USA, February 1979.

[53] C. L. Forgy. Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, September 1982.

[54] E. Freund, A. Hypki, and D. H. Pensky. New architecture for corporate integration of simulation and production control in industrial applications. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 806–811, Seoul, Korea, May 2001.

[55] E. Freund and D. H. Pensky. COSIMIR factory: Extending the use of manufacturing simulations. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 2805–2810, Washington, DC, USA, May 2002.

[56] E. Freund and D. H. Pensky. Distributing 3d manufacturing simulations to realize the digital plant. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 1705–1710, Taipei, Taiwan, September 2003.

[57] E. Friedman-Hill. *Jess in Action: Rule–Based Systems in Java*. Manning Publications Company, Greenwich, Connecticut, USA, 2003.

[58] A. Fumagalli, G. Gaias, and P. Masarati. A simple approach to kinematic inversion of redundant mechanisms. In *Proceedings of the ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Las Vegas, Nevada, USA, September 2007.

[59] A. Fumagalli, P. Masarati, and A. Ercoli Finzi. Feedback dynamic compensation via multibody models and applications to robot control. In C. Botasso, editor, *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics*, Milano, Italy, June 2007.

[60] J. Funda, R. Taylor, and R. Paul. On homogeneous transforms, quaternions, and computational efficiency. *IEEE Transactions on Robotics and Automation*, 6(3):382–388, June 1990.

[61] Y. Gao. Decomposable geometric inverse kinematics for reconfigurable robots using product–of–exponential formula. Master of engineering thesis, School of Mechanical and Production Engineering, Nanyang Technological University, 2000.

[62] J. Giarratano and G. Riley. *Expert Systems – Principles and Programming*. Thomson / PWS Publishing Company, 4th edition, 2004.

[63] A. A. Goldenberg, B. Benhabib, and R. Fenton. A complete generalized solution to the inverse kinematics of robots. *IEEE Journal of Robotics and Automation*, 1(1):14–20, March 1985.

[64] V. E. Gough and S. G. Whitehall. Universal tyre test machine. In *Proceedings of the 9th International Technical Congress FISITA*, pages 117–137, May 1962.

[65] R. Gourdeau. Object oriented programming for robotic manipulators simulation. *IEEE Robotics and Automation Magazine*, 4(3):21–29, September 1997.

[66] F. S. Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, March 1998.

[67] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: nonstiff problems*. Springer Verlag, New York, USA, 2nd revised edition, 1993.

[68] D. Halperin. Automatic kinematic modelling of robot manipulators and symbolic generation of their inverse kinematics solutions. In *Proceedings of the 2nd International Workshop on Advances in Robot Kinematics*, pages 310–317, Linz, 1990.

[69] W. R. Hamilton. On quaternions or on a new system of imaginaries in algebra. *Philosophical Magazine*, 25:10–13, July 1844.

[70] L. G. Herrera-Bendezu, E. Mu, and J. T. Cain. Symbolic computation of robot manipulator kinematics. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 993–998, Philadelphia, Pennsylvania, USA, 1988.

[71] W. Hirschberg and D. Schramm. Application of neweul in robot dynamics. *Journal of Symbolic Computation*, 7(2):199–204, February 1989.

[72] J. M. Hollerbach. A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):730–736, November 1980.

[73] J. M. Hollerbach. Optimum kinematic design for a seven degree of freedom manipulator. In H. Hanafusa and H. Inoue, editors, *Proceedings of the Second International Symposium on Robotics Research*, pages 341–349, Kyoto, Japan, August 1984.

[74] R. Höpler and P. J. Mosterman. Model integrated computing in robot control to synthesize real–time embedded code. In *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA)*, pages 767–772, Mexico City, Mexico, September 2001.

[75] R. Höpler and M. Otter. A versatile c++ toolbox for model based, real time control systems of robotic manipulators. In *Proceedings of the IEEE/RSJ International International Conference on Robotics and Systems (IROS)*, pages 2208–2214, Maui, Hawaii, USA, October 2001.

[76] R. Houde, M. Blain, and J. Côté. *Manuel de l'usager pour Microb.* Institut de recherche d'Hydro–Québec, Varennes, Québec, Canada, 2000.

[77] M. L. Husty. An algorithm for solving the direct kinematics of general stewart–gough platforms. *Mechanism and Machine Theory*, 31(4):365–380, May 1996.

[78] A. Jaramillo-Botero, A. Matta-Gomez, J. F. Correa-Caicedo, and W. Perea-Castro. Robomosp. *IEEE Robotics and Automation Magazine*, 13(4):62–73, December 2006.

[79] C. Jia, H. Zhang, and J. Lu. Planning and simulation of virtual production system. In *Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 452–456, Luoyang, China, June 2006.

[80] G. L. Steele Jr. *Common Lisp: The Language.* Digital Press, Burlington, Massachusetts, USA, 1984.

[81] D. L. Jung, W. E. Dixon, and F. G. Pin. Automated kinematic generator for surgical robotic systems. In *Proceedings of the 12th Annual Medicine Meets Virtual Reality Conference*, pages 144–146, Newport Beach, California, USA, January 2004.

[82] E. Jung, C. Kapoor, and D. Batory. Automatic code generation for actuator interfacing from a declarative specification. In *Proceedings of the IEEE/RSJ International International Conference on Robotics and Systems (IROS)*, pages 3207–3212, Edmonton, Canada, August 2005.

[83] W. Kahan. Lectures on computational aspects of geometry. Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, USA, 1983. Unpublished manuscript.

[84] T. R. Kane and D. A. Levinson. Multibody dynamics. *Journal of Applied Mechanics, Transactions of ASME*, 50(4b):1071–1078, December 1983.

[85] C. Kapoor, M. Cetin, M. Pryor, C. Cocca, T. Harden, and D. Tesar. A software architecture for multi–criteria decision making for advanced robotics. In *Proceedings of the International Joint Conference on Science and Technology of Intelligent Systems*, pages 525–530, Gaithersburg, Maryland, USA, September 1998.

[86] L. Kelmar and P. Khosla. Automatic generation of kinematics for a reconfigurable modular manipulator system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 663–668, Philadelphia, Pennsylvania, USA, April 1988.

[87] H. Y. Lee and C. G. Liang. A new vector theory for the analysis of spatial mechanisms. *Journal of Mechanism and Machine Theory*, 23(3):209–217, 1988.

[88] G. Legnani, F. Casolo, P. Righettini, and B. Zappa. A homogeneous matrix approach to 3d kinematics and dynamics. part 1: theory. *Mechanism and Machine Theory*, 31(5):573–587, July 1996.

[89] G. Legnani, B. Zappa, and P. Righettini. A homogeneous matrix approach to 3d kinematics and dynamics. part 2: Application to chains of rigid bodies and serial manipulators. *Mechanism and Machine Theory*, 31(5):589–605, July 1996.

[90] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, July 1944.

[91] R. Lot and M. Da Lio. A symbolic approach for automatic generation of the equations of motion of multibody systems. *Multibody System Dynamics*, 12(2):147–172, September 2004.

[92] J. Y. Luh, M. W. Walker, and R. P. Paul. On–line computational scheme for mechanical manipulators. *Journal of Dynamics Systems, Measurement and Control, Transactions of ASME*, 102(2):69–76, June 1980.

[93] F. S. Macaulay. *The Algebraic Theory of Modular Systems*. Cambridge University Press, Cambridge, UK, 1916.

[94] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, USA, May 1992.

[95] D. Manocha and J. F. Canny. Real time inverse kinematics for general 6r manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 383–389, Nice, France, May 1992.

[96] D. Manocha and J. F. Canny. Efficient inverse kinematics for general 6r manipulators. *IEEE Transactions on Robotics and Automation*, 10(5):648–657, 1994.

[97] R. Manseur. A software package for computer–aided robotics education. In *Proceedings of the 26th Annual Conference Frontiers in Education Conference (FIE)*, volume 3, pages 1409–1412, Salt Lake City, Utah, USA, November 1996.

[98] R. Manseur. Interactive visualization tools for robotics education. In *Proceedings of the International Conference on Engineering Education (ICEE)*, Gainesville, Florida, USA, October 2004.

[99] R. Manseur. Virtual reality in science and engineering education. In *Proceedings of the 35th Annual Conference Frontiers in Education Conference (FIE)*, pages F2E 8–13, Indianapolis, Indiana, USA, October 2005.

[100] R. Manseur and K. L. Doty. A robot manipulator with 16 real inverse kinematic solution sets. *International Journal of Robotics Research*, 8(5):75–79, 1989.

[101] R. Manseur and K. L. Doty. A complete kinematic analysis of four–revolute–axis robot manipulators. *ASME Mechanisms and Machines*, 27(5):575–586, 1992.

[102] R. Manseur and K. L. Doty. Fast inverse kinematic analysis of five–revolute–axis robot manipulators. *ASME Mechanisms and Machines*, 27(5):587–597, 1992.

[103] P. March, R. Taylor, C. Kapoor, and D. Tesar. Decision making for remote robotic operations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2764–2769, New Orleans, Louisiana, USA, April 2004.

[104] D.W. Marhefka. *Fuzzy control and dynamic simulation of a quadruped galloping machine*. PhD thesis, Department of Electrical Engineering, Ohio State University, Columbus, Ohio, USA, 2000.

[105] D. W. Marquardt. An algorithm for least–squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, June 1963.

[106] E. A. Maxwell. *General Homogeneous Coordinates in Space of Three Dimensions*. Cambridge University Press, Cambridge, UK, 1951.

[107] E. Mayr and A. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305–329, 1982.

[108] E. W. Mayr. Some complexity results for polynomial ideals. *Journal of Complexity*, 13(3):303–325, September 1997.

[109] S. McMillan. *Computational Dynamics for Robotic Systems on Land and Under Water*. PhD thesis, The Ohio State University, Columbus, Ohio, USA, 1994.

[110] S. McMillan, D. E. Orin, and R. B. McGhee. Object–oriented design of a dynamic simulation for underwater robotic vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1886–1893, Nagoya, Japan, 1995.

[111] S. McMillan, D. E. Orin, and R. B. McGhee. A computational framework for simulation of underwater robotic vehicle systems. *Special Issue of the Journal of Autonomous Robots on Autonomous Underwater Robots*, 3:253–268, 1996.

[112] S. McMillan, P. Sadayappan, and D. E. Orin. Efficient dynamic simulation of multiple manipulator systems with singular configurations. *IEEE Transactions on Systems, Man, and Cybernetics*, 24:306–313, February 1994.

[113] J. McPhee, C. Schmitke, and S. Redmond. Dynamic modelling of mechatronic multibody systems with symbolic computing and linear graph theory. *Mathematical and Computer Modelling of Dynamical Systems*, 10(1):1–23, 2004.

[114] D. P. Miranker. Treat: A better match algorithm for ai production systems. In *Proceedings of Sixth National Conference on Artificial Intelligence*, pages 42–47, Seattle, Washington State, USA, August 1987.

[115] D. P. Miranker. *Treat: A new and efficient match algorithm for AI production systems*. Morgan Kaufmann Publishers Inc., San Francisco, California, USA, 1990.

[116] E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395, 1920.

[117] M. Morandini, P. Masarati, and P. Mantegazza. A real–time hardware–in–the–loop simulator for robotics applications. In J. M. Goicolea, J. Cuadrado, and J. C. Garcia Orden, editors, *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics*, Madrid, Spain, June 2005.

[118] S. Morgan. *Programming Microsoft Robotics Studio Developer Reference*. Microsoft Press, New York, USA, 2008. to appear.

[119] K. Morisse and G. Oevel. New developments in MuPAD. In J. Fleischer, J. Grabmeier, F. W. Hehl, and W. Küchlin, editors, *Computer Algebra in Science and Engineering*, pages 44–56, Singapore, 1995. World Scientific Publishing.

[120] H. D. Nayar. Robotect: serial–link manipulator design software for modeling, visualization and performance analysis. In *Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision (ICARCV 2002)*, volume 3, pages 1359–1364, Singapore, December 2002.

[121] J. F. Nethery. Robotica: A structured environment for computer aided design and analysis of robots. Master's thesis, University of Illinois at Urbana–Champaign, Department of Electrical and Computer Engineering, June 1994.

[122] J. F. Nethery and M. W. Spong. Robotica: A Mathematica package for robot analysis. *IEEE Robotics and Automation Magazine*, 1(1):13–20, March 1994.

[123] J. F. Nethery and M. W. Spong. A mathlink–based front end for the robotica package. *The Mathematica Journal*, 5(2):72–79, 1995.

[124] N. M. Newmark. A method of computation for structural dynamics. *ASCE Journal of Engineering Mechanics Division*, 85(EM3):67–94, March 1959.

[125] I. Newton. *Mathematical Principles of Natural Philosophy*, volume 34 of *Great Books of the Western World*. Encyclopaedia Britannica, Chicago, Illinois, USA, 1687.

[126] D. E. Orin and W. W. Schrader. Efficient computation of the jacobian for robot manipulators. *International Journal of Robotics Research*, 3(4):66–75, 1984.

[127] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, USA, 2000.

[128] M. Otter, H. Elmqvist, and F. E. Cellier. Modeling of multibody systems with the object–oriented modeling language Dymola. In *Proceedings of the NATO Advanced Study Institute on Computer Aided Analysis of Rigid and Flexible Mechanical Systems*, pages 91–110, Troia, Portugal, June 1993.

[129] B. Paden. *Kinematics and Control of Robot Manipulators*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, USA, 1986.

[130] B. Paden and S. Sastry. Optimal kinematic design of 6r manipulators. *International Journal of Robotics Research*, 7(2):43–61, March 1988.

[131] F. C. Park. Computational aspects of the product–of–exponentials formula for robot kinematics. *IEEE Transactions on Automatic Control*, 39(3):643–647, March 1994.

[132] T. Parr. *The Definitive ANTLR Reference: Building Domain–Specific Languages (Pragmatic Programmers)*. Pragmatic Bookshelf, May 2007.

[133] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, Massachusetts, USA and London, UK, 1981.

[134] R. P. Paul, M. Renaud, and C. N. Stevenson. A systematic approach for obtaining the kinematics of recursive manipulators based on homogeneous transformations. In M. Brady and R. Paul, editors, *Proceedings of the First International Symposium on Robotics Research*, pages 707–726, Cambridge, Massachusetts, USA, August 1984. MIT Press.

[135] R. P. Paul and H. Zhang. Computationally efficient kinematics for manipulators with spherical wrists based on the homogeneous transformation representation. *International Journal of Robotics Research*, 5(2):32–44, June 1986.

[136] R. A. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[137] A. Perez, H. J. Su, and J. M. McCarthy. Synthetica 2.0: Software for the synthesis of constrained serial chains. In *Proceedings of the ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference (DETC)*, Salt Lake City, Utah, USA, September 2004.

[138] L. R. Petzold. A description of dassl: A differential/algebraic system solver. *IMACS Transactions on Scientific Computing*, 1:65–68, 1983.

[139] T. Pfirrmann. Xj3D–basierte Visualisierung von Industrierobotern. Pre–master's thesis, Institut für Prozessrechentechnik, Automation und Robotik, Universität Karlsruhe (TH), February 2008.

[140] D. L. Pieper. *The Kinematics of Manipulators under Computer Control*. PhD thesis, Stanford University, Department of Mechanical Engineering, October 1968.

[141] F. G. Pin, L. J. Love, and D. L. Jung. Automated kinematic equations generation and constrained motion planning resolution for modular and reconfigurable robots. In *Proceedings of the 227th American Chemical Society National Meeting*, Anaheim, California, USA, March 2004.

[142] V. Poiré, S. Lemieux, P. Bigras, and M. Blain. Migration procedure of control laws from a graphical simulation environment to a robotic framework. In C.-Y. Su, editor, *Proceedings of the 8th IASTED International Conference on Control and Aplications (CA 2006)*, Montreal, Quebec, Canada, May 2006. ACTA Press.

[143] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes: the art of scientific computing*. Cambridge University Press, New York, USA, 1986.

[144] E. J. F. Primrose. On the input–output equation of the general 7r–mechanism. *Mechanisms and Machine Theory*, 21(6):509–510, 1986.

[145] M. Pryor, R. Taylor, C. Kapoor, and D. Tesar. Generalized software components for reconfiguring hyper–redundant manipulators. *IEEE/ASME Transactions on Mechatronics*, 7(4):475–478, December 2002.

[146] M. Raghavan. The Stewart platform of general geometry has 40 configurations. *Journal of Mechanical Design, Transactions of ASME*, 115(2):277–282, June 1993.

[147] M. Raghavan and B. Roth. Kinematic analysis of the 6r manipulator of general geometry. In H. Miura and S. Arimoto, editors, *The fifth international symposium on Robotics research*, pages 263–269, Cambridge, Massachusetts, USA, 1990. MIT Press.

[148] H. Rieseler and F. M. Wahl. Fast symbolic computation of the inverse kinematics of robots. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 462–467, Cincinnati, Ohio, USA, May 1990.

[149] M. F. Robinette and R. Manseur. Robot–draw, an internet–based visualization tool for robotics education. *IEEE transactions on education*, 44(1):29–34, February 2001.

[150] O. Rodrigues. Des lois geometriques qui regissent les deplacements d'un systeme solide independamment des causes qui peuvent les produire. *Journal De Mathematiques Pures et Appliquees*, 5:380–440, 1840.

[151] R. R. Ryan. Adams–multibody system analysis software. In W. O. Schiehlen, editor, *Multibody Systems Handbook*, pages 361–402. Springer–Verlag, Berlin, 1990.

[152] L. Sass. *Symbolic modeling of electromechanical multibody systems*. PhD thesis, Center for Research in Mechatronics, Université catholique de Louvain, Louvain–la–Neuve, Belgium, January 2004.

[153] D. L. Schneider. Review of the Robotica software package for robotic manipulators. *IEEE Robotics and Automation Magazine*, 1(1):21–22, March 1994.

[154] E. M. Schwartz, R. Manseur, and K. L. Doty. Algebraic properties of noncommensurate systems. In *Proceedings of the Florida Conference on Recent Advances in Robotics (FCRAR 1999)*, Gainesville, Florida, April 1999.

[155] J. M. Selig. Active versus passive transformations in robotics. *IEEE Robotics and Automation Magazine*, 13(1):79–84, March 2006.

[156] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, volume 19, pages 245–254, San Francisco, California, USA, July 1985. ACM Press.

[157] K. Shoemake. Euler angle conversion. In P. Heckbert, editor, *Graphics Gems IV*, pages 222–229. Academic Press Professional, San Diego, California, USA, 1994.

[158] W. M. Silver. On the equivalence of lagrangian and newton–euler dynamics for manipulators. *International Journal of Robotics Research*, 1(2):60–70, 1982.

[159] B. Sivaraman, T. Burks, and J. Schueller. Using modern robot synthesis and analysis tools for the design of agricultural manipulators. *Agricultural Engineering International: the CIGR Ejournal*, 8, January 2006. Invited Overview Paper No. 2.

[160] G. S. Soh, A. Perez, and J. M. McCarthy. The kinematic synthesis of mechanically constrained planar 3r chains. In M. Husty and H.-P. Schröcker, editors, *Proceedings of the 1st European Conference on Mechanism Science (EuCoMeS)*, Obergurgl, Austria, February 2006.

[161] M. W. Spong. *Robot Dynamics and Control*. John Wiley and Sons, Inc., New York, USA, 1989.

[162] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley and Sons, New York, USA, 2005.

[163] D. Stewart. A platform with six degrees of freedom. In *Proceedings of the Institute of Mechanical Engineering*, volume 180, pages 371–386, 1965.

[164] J. Swevers, C. Ganseman, X. Chenut, and J.-C. Samin. Experimental identification of robot dynamics for control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 241–246, San Francisco, California, USA, 2000.

[165] M. Thomas and D. Tesar. Dynamic modelling of serial manipulator arms. *Journal of Dynamic Systems, Measurement, and Control, ASME Transactions*, 104(9):218–228, September 1982.

[166] D. Tolani. *Analytic Inverse Kinematics Techniques for Anthropometric Limbs*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, USA, January 1998.

[167] D. Tolani, A. Goswami, and N. I. Badler. Real–time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models and Image Processing*, 62(5):353–388, September 2000.

[168] V. D. Tourassis and M. H. Ang. Task decoupling in robot manipulators. *Journal of Intelligent and Robotic Systems*, 14(3):283–302, November 1995.

[169] L. W. Tsai and A. P. Morgan. Solving the kinematics of the most general six– and five–degree–of–freedom manipulators by continuation methods. *ASME Journal of Mechanisms, Transmission, and Automation in Design*, 107:189–200, 1985.

[170] J. J. Uicker. Dynamic force analysis of spatial linkages. *Journal of Applied Mechanics, Transactions of ASME*, 34(89):418–424, June 1967.

[171] J. J. Uicker, J. Denavit, and R. S. Hartenberg. An interactive method for the displacement analysis of spatial mechanisms. *Journal of Applied Mechanics, Transactions of ASME*, 86(2):309–314, June 1964.

[172] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools (JGT)*, 2(4):1–14, April 1997.

[173] Verein Deutscher Ingenieure VDI VDI–Gesellschaft Produktionstechnik, editor. *VDI Richtlinie 2860: Montage– und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole*. Beuth, Berlin, Germany, May 1990.

[174] O. Verlinden, P. Dehombreux, C. Conti, and S. Boucher. Formulation for the direct dynamic simulation of flexible mechanisms based on the newton–euler inverse method. *International Journal for Numerical Methods in Engineering*, 37(19):3363–3387, February 1994.

[175] O. Verlinden, G. Kouroussis, and C. Conti. EasyDyn: A framework based on free symbolic and numerical tools for teaching multibody systems. In J.M. Goicolea, J. Cuadrado, and J.C. Garcia Orden, editors, *Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics 2005*, Madrid, Spain, June 2005.

[176] O. Verlinden, G. Kouroussis, S. Datoussaid, and C. Conti. Open source symbolic and numerical tools for the simulation of multibody systems. In *Proceedings of the 6th National Congress on Theoretical and Applied Mechanics (on CD)*, Ghent, Belgium, May 2003.

[177] D. N. Vila-Rosado and J. A. Dominguez-Lopez. A matlab toolbox for robotic manipulators. In *Proceedings of the Sixth Mexican International Conference on Computer Science (ENC 2005)*, pages 256–265, Puebla, Mexico, September 2005. IEEE Computer Society.

[178] D. N. Vila-Rosado and J. A. Dominguez-Lopez. A matlab toolbox for the optimal design of robot manipulators using evolutionary techniques. In D. J. Dorantes-González, editor, *Proceedings the 10th International Conference on Mechatronic Technology (ICMT 2006)*, pages 256–265, Mexico City, Mexico, November 2006.

[179] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamics Systems, Measurement and Control, Transactions of ASME*, 104(3):205–211, September 1982.

[180] C. W. Wampler and A. P. Morgan. Solving the kinematics of general 6r manipulators using polynomial continuation. In K. Warwick, editor, *Robotics: Applied Mathematics and Computational Aspects*. Clarendon Press, Oxford, UK, March 1993.

[181] Z. Wang, D. Zhang, and H. Han. Design and kinematics analysis of oblique axis non–spherical 3r wrist. In *Proceedings of the IEEE International Conference on Automation and Logistics*, pages 995–1000, Jinan, Shandong, China, August 2007.

[182] W. Weber and S. Rothenbücher. Mandy: Tool for fast development of open chain multibody systems. In *Proceedings of the 6th International Workshop on Research and Education in Mechatronics (REM)*, pages 287–291, Annecy, France, July 2005.

[183] M. Wenz and H. Wörn. Automatic configuration of the dynamic model for common industrial robots. In C. Hochberger and R. Liskowsky, editors, *Proceedings of the GI Jahrestagung*, GI–Edition Lecture Notes in Informatics (LNI), pages 137–144, Dresden, Germany, October 2006.

[184] M. Wenz and H. Wörn. Entwicklung einer selbstkonfigurierenden und selbstorganisierenden Bewegungssteuerung für Industrieroboter. In G. Brandenburg, W. Schumacher, R. D. Schraft, and K. Bender, editors, *Tagungsband SPS/IPC/Drives 2006*, pages 619–627, Nürnberg, Germany, November 2006.

[185] M. Wenz and H. Wörn. Rule–based generation of motion control software for general serial–link robot manipulators. In *Proceedings of the 8th International Workshop on Computer Science and Information Technologies (CSIT 2006)*, pages 16–21, Karlsruhe, Germany, September 2006.

[186] M. Wenz and H. Wörn. Automatische Konfiguration der Bewegungssteuerung von seriellen Kinematiken. In J. Gausemeier, F. Rammig, W. Schäfer, A. Trächtler, and J. Wallaschek, editors, *Entwurf mechatronischer Systeme*, volume 210 of *HNI–Verlagsschriftenreihe*, pages 57–68, Paderborn, Germany, March 2007.

[187] M. Wenz and H. Wörn. Computing symbolic–algebraic models of industrial manipulator arms in a fully automatic way. In K. Schilling, editor, *Proceedings of the 13th IASTED International Conference on Robotics and Applications (RA 2007)*, pages 334–339, Würzburg, Germany, August 2007. ACTA Press.

[188] M. Wenz and H. Wörn. Solving the inverse kinematics problem symbolically by means of knowledge–based and linear algebra–based methods. In *Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2007)*, pages 1346–1353, Patras, Greece, September 2007. IEEE Press.

[189] R. L. Williams. Inverse kinematics and singularities of manipulators with offset wrist. *IASTED International Journal of Robotics and Automation*, 14(1):1–8, 1999.

[190] D. W. Wloka and K. Blug. Simulation der Dynamik von Robotern nach dem Verfahren von Kane. *Robotersysteme*, 1(4):211–216, 1985.

[191] W. A. Wolovich. *Robotics: Basic Analysis and Design*. CBS College Publishing, New York, USA, 1987.

[192] W. A. Wolovich and H. Elliot. A computational technique for inverse kinematics. In *Proceedings of the 32nd IEEE Conference on Decision and Control*, pages 1359–1362, Las Vegas, Nevada, USA, December 1984.

[193] H. Wörn and U. Brinkschulte. *Echtzeitsysteme*, volume 1 of *eXamen.press*. Springer Verlag, Institut für Prozessrechentechnik, Automation und Robotik, Universität Karlsruhe (TH), 1st edition, Februar 2005.

[194] L. Zlajpah. Integrated environment for modelling, simulation and control design for robotic manipulators. *Journal of Intelligent and Robotic Systems*, 32(2):219–234, October 2001.