THM | CAMPUS GIESSEN | MNI
Mathematik, Naturwissenschaften und Informatik
TECHNISCHE HOCHSCHULE MITTELHESSEN

Modul II 2004 im Studiengang
BSc Ingenieur-Informatik

Grundlagen der Robotik

Prof. Dr. Klaus Wüst

Skriptum

Stand: 25. Juni 2018
Autor: K.Wüst

Inhaltsverzeichnis

1. Allgemeines zur Robotik

Woher kommt das Wort Roboter?

- Ein Roboter ist eine technische Apparatur, die üblicherweise dazu dient, dem Menschen mechanische Arbeit abzunehmen. Roboter können sowohl stationäre als auch mobile Maschinen sein und werden von Computerprogrammen gesteuert. (Wikipedia)
- Der Name stammt vom tschechischen Wort *robota* das so etwas bedeutet wie harte Arbeit, Frondienst. Wurde benutzt in einem Roman von Josef Capek.
- Gemeint sind also Automaten, die dem Menschen harte Arbeit abnehmen.

Definition von Robotern

- Nicht einheitlich, in jedem Land anders.
- Nach der VDI Richtlinie 2860:
  „Industrieroboter sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei (d. h. ohne mechanischen bzw. menschlichen Eingriff) programmierbar und gegebenenfalls sensorgeführt sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- und/oder Fertigungsaufgaben ausführen."
- Nach der Robotic Industries Association:
  „A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks"

Roboter sind flexibel!

„An automated machine that does just one thing is not a robot. It is simply automation. A robot should have the capability of handling a range of jobs in factory." (Roboterpionier Joseph Engelberger)

Dies Flexibilität macht Roboter so wertvoll, denn für Umstellungen im Fertigungsprozess muss man nur die Software, evtl. noch die tools, ändern, nicht aber den Roboter selbst.

Welche Arten Roboter gibt es?

Stationäre Roboter An einem Standort fest montiert, klar abgegrenzter Arbeits– und Kollisionsraum, in der Regel fester Bewegungsablauf, meist über Busse und Signalleitungen vernetzt, arbeiten oft ohne Sensoren, anderer Name: Industrieroboter

Mobile Roboter Können sich fortbewegen und damit Ihren Standort verlassen, prinzipiell unbegrenzter Arbeitsraum, brauchen Sensoren, arbeiten autonom, wenn vernetzt dann drahtlos, haben Akku und Ladestation.

Service-Roboter Roboter, der außerhalb des Industriebereichs nützliche Dinge für Menschen oder Sachen tut.

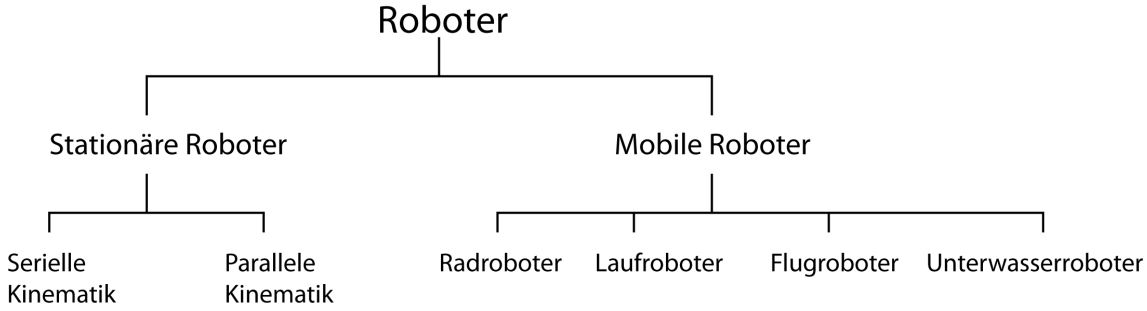


Abbildung 1.1.: Unterteilung der Roboter

Trends bei Industrierobotern

- Der häufigste Einsatzort für Industrieroboter ist nach wie vor die Autoindustrie
- Industrieroboter werden allmählich immer preiswerter.
- Industrieroboter werden allmählich immer leistungsfähiger.
- Industrieroboter dringen ständig in neue Anwendungsbereiche vor.
- Zunehmend parallele Kinematiken

[1]

[1] Abbildungen in diesem Abschnitt mit freundlicher Genehmigung der International Federation of Robotics (IFR), www.ifr.org, Dokument „World Robotics Survey: Industrial robots are conquering the world“, 2015 [16]

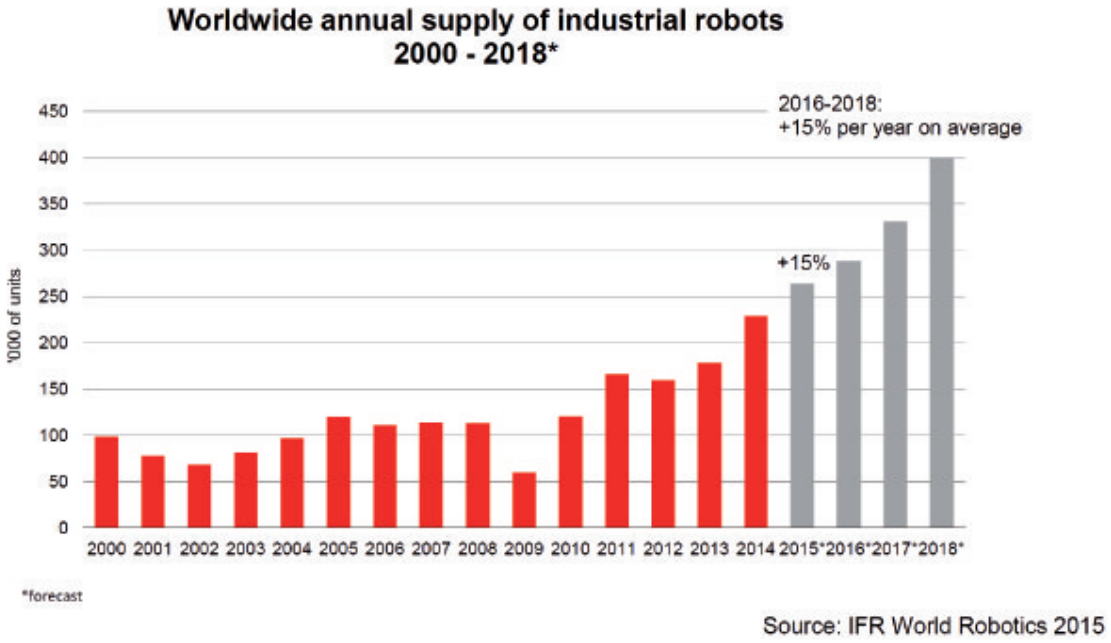


Abbildung 1.2.: Weltweit betriebene Roboter.

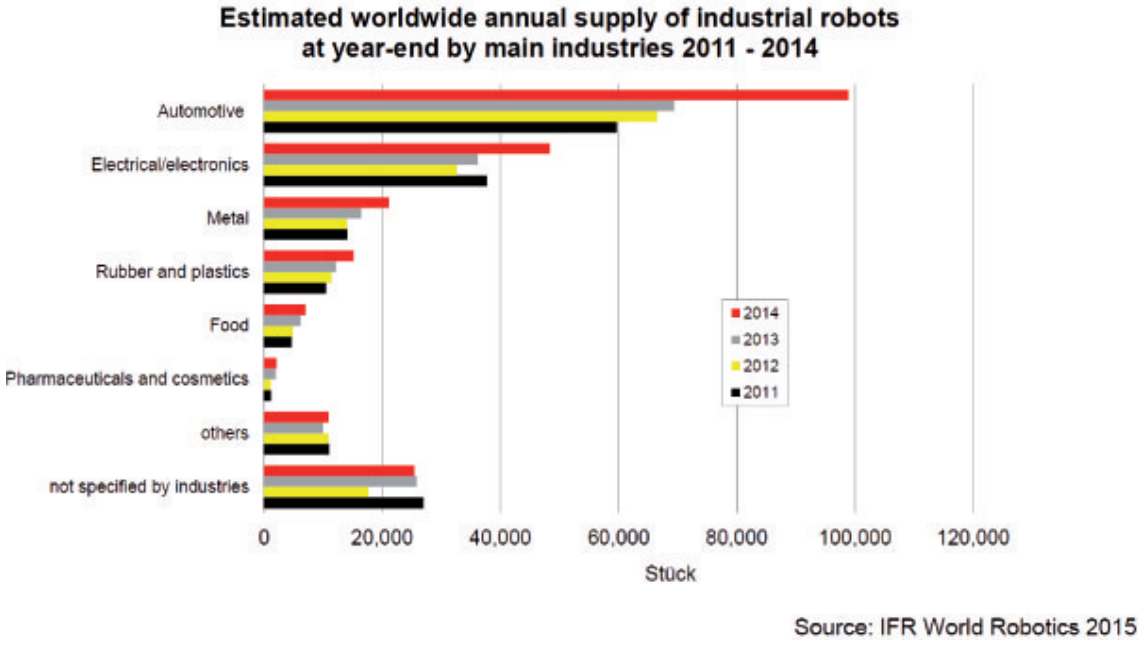


Abbildung 1.3.: Roboter nach Branchen (Quelle IFR)

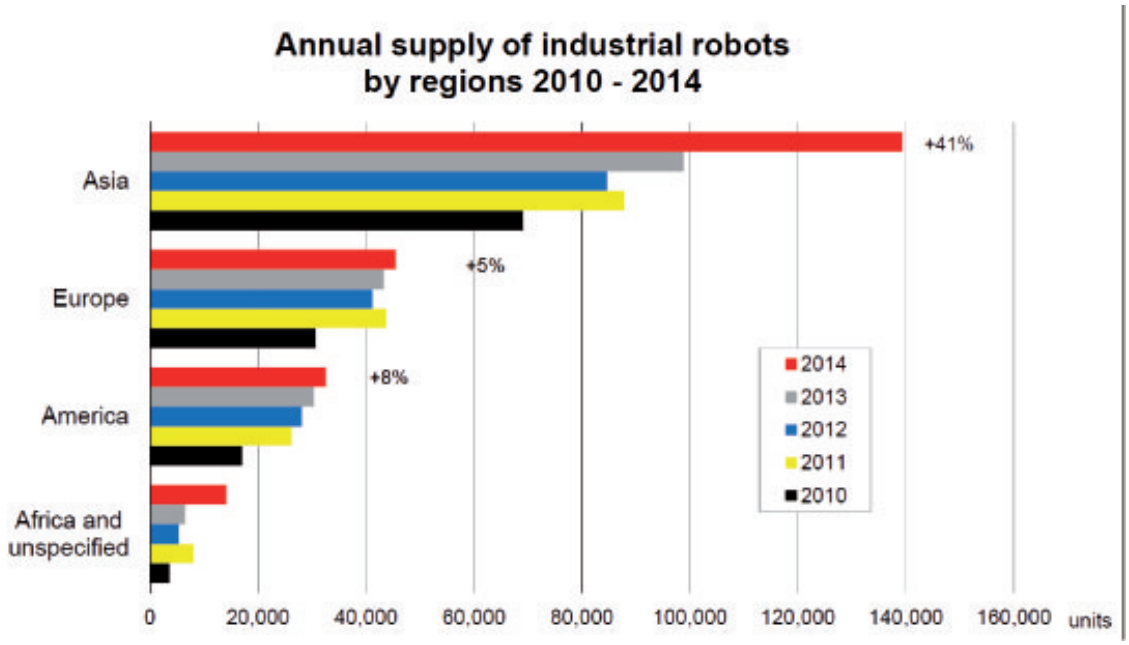


Abbildung 1.4.: Jährlicher Bedarf an Robotern (Quelle IFR)

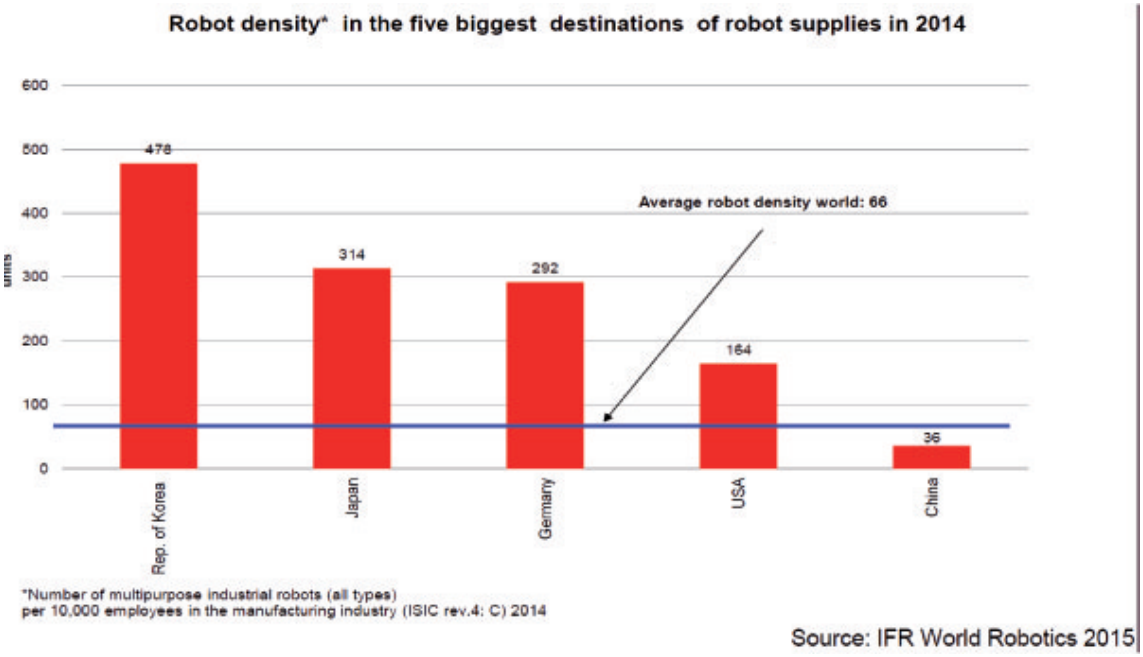


Abbildung 1.5.: Roboterdichte in der Industrie (Quelle IFR)

Neuer Trend: Kollaborative Leichtbauroboter

Abbildung 1.6.: Leichtbauroboter iiwa von KUKA (Quelle KUKA)

Trend: Serviceroboter

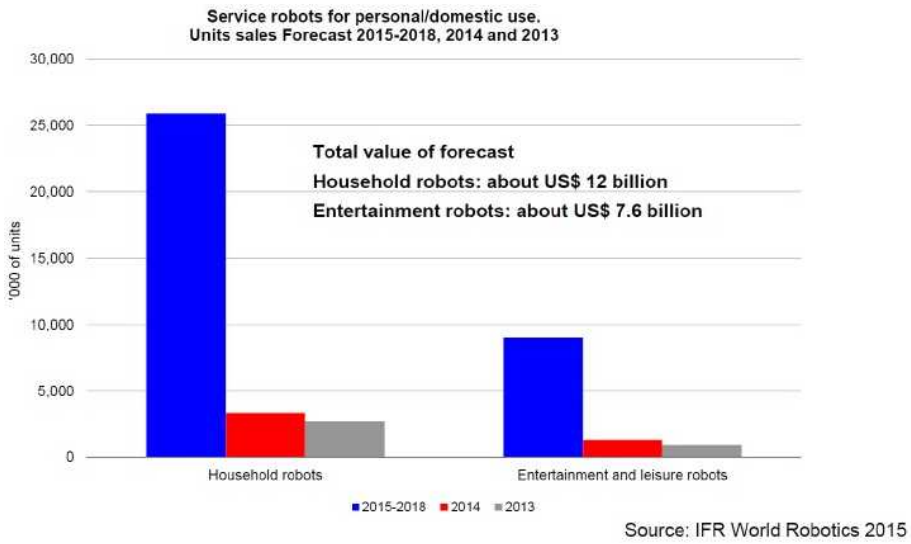


Abbildung 1.7.: Auslieferung Serviceroboter jährlich.

Einsatzfelder für Serviceroboter sind:

- Reinigung Boden, Scheiben u.a.
- Mähen
- Unterhaltung und Freizeit
- Unterstützung von älteren Menschen und Menschen mit Handycaps

2. Kurze Einführung in die Automatisierung

2.1. Allgemeines

Warum Automatisierung?

- Verbesserung der Qualität und Erhöhung der Produktion
- Senkung der Kosten
- Bessere Ausnutzung von Rohstoffen und Energie
- Realisierung von Fertigungsmethoden, die manuell nicht mehr machbar sind
- Verbesserung der Betriebsbereitschaft durch Verminderung der Bedienfehler und Reparaturen
- Weniger Umweltbelastung durch optimale Prozesssteuerung
- Weniger Gesundheitsgefährdung und -belastung des Personals
- Mengenerfassung für Resourcenmanagement und Kalkulation

Prozessautomatisierungssystem und Prozessleitsystem

Ein Automatisierungssystem steht zwischen der eigentlichen Produktionsanlage und den menschlichen Bedienern. Es hat nach oben eine Bedienungsschnittstelle (Men-Machine-Interface, MMI) für das Bedienungspersonal und nach unten eine technische Schnittstelle zur Anlage.

Mehrere Automatisierungssysteme können durch ein *Prozessleitsystem* zentral angesteuert werden.

Bei noch größeren Anlagen wird Prozessleitebene, Gruppenleitebene und Einzelleitebene (Automatisierungssystem) unterschieden.

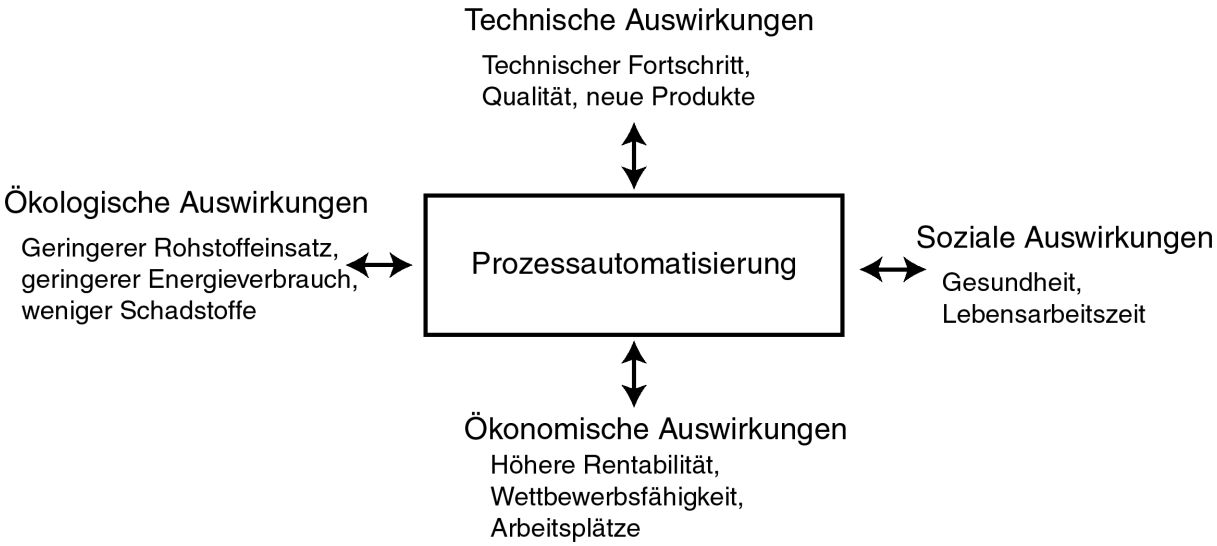


Abbildung 2.1.: Auswirkungen der Automatisierung

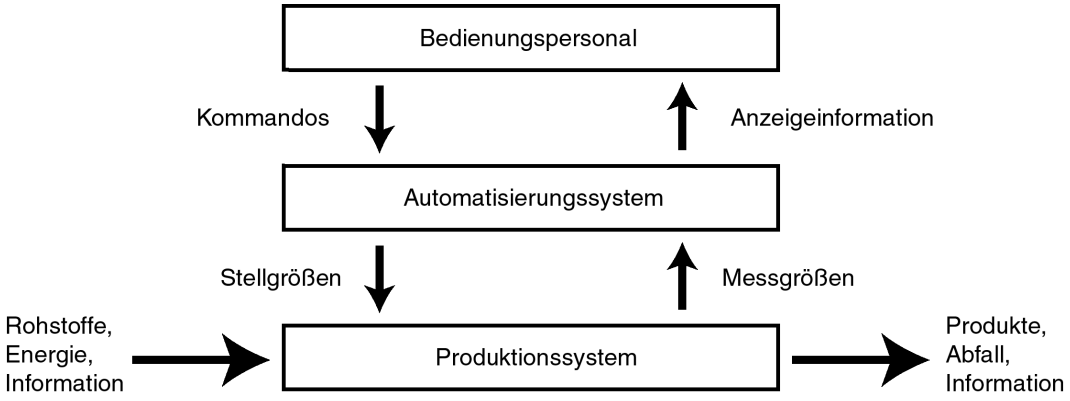


Abbildung 2.2.: Grundstruktur eines Automatisierungssystems

Aufgaben und Teilaufgaben eines Automatisierungssystems

Prozessbedienung Mensch-Maschine-Schnittstelle, Anzeige der Prozesszustände, Bedienelemente

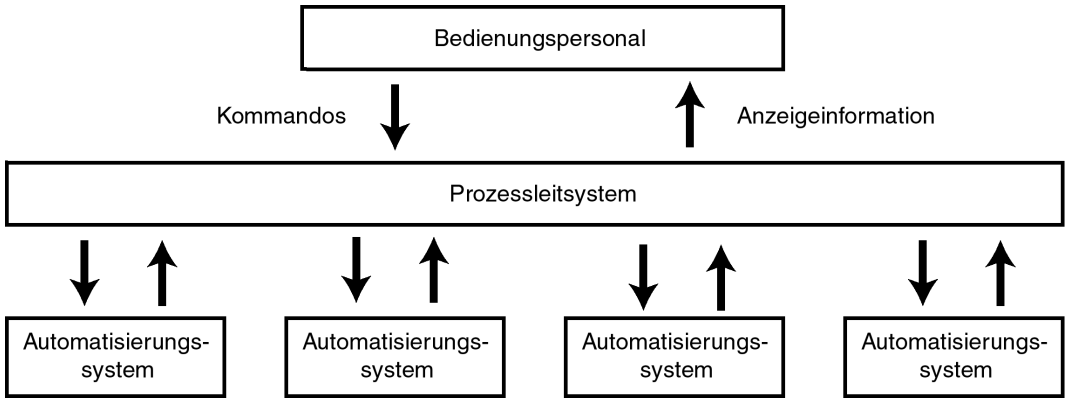


Abbildung 2.3.: Ein Prozessleitsystem leitet mehrere Automatisierungssysteme

Hardwareelemente: Tasten, Touchscreen, Sprachsteuerung, Computerschnittstelle zum Leitrechner, zur Anzeige Lampen, LED, Displays, LCD

Prozessführung Prozess schrittweise abarbeiten, Teilprozesse koordinieren
Hardwareelemente: Mikroprozessor oder SPS (Speicherprogrammierbare Steuerung) mit Programm, digitale und analoge Signalausgänge für Stellelemente

Prozessüberwachung und Stabilisierung Erfassen und nachführen (regeln) aller Größen
Hardwareelemente: Sensoren, Messverstärker, Analog/Digital-Wandler, Regler oder Regelungssoftware in der Software

Prozesssicherung Erkennung gefährlicher Systemzustände, Einleitung von Gegenmaßnahmen
Hardwareelemente: Sensoren und Stellelemente mit eigenem Steuerkreis

Prozessbilanzierung Erfassung und Protokollierung der Mengen und Zeiten, (Herstellerhaftung!)
Hardwareelemente: Massenspeicher (Festplatte, Band), Verbindung zum Leitrechner

Prozessoptimierung Optimierung der Qualität, optimale Ausnutzung von Rohstoffen und Energie
Hardwareelemente: Softwarealgorithmen

2.2. Ein Zwei-Komponentenreaktor als einführendes Beispiel

Als einführendes Beispiel wird nun ein Zwei-Komponentenreaktor vorgestellt. In dem diskontinuierlich (portionsweise) arbeitenden Reaktor sollen zwei Komponenten in einem endothermen Prozess (Wärmezufuhr nötig) zu einem Zwischenprodukt chemisch reagieren. Komponente 1 wird aufbereitet aus einem zerkleinerten Feststoff und einem Lösungsmittel. Komponente 2 ist flüssig und wird aus einem Tank zugeführt. Für die Zuführung der Komponenten werden eine Zerkleinerungsanlage, Pumpen, Ventile und Motoren gebraucht.

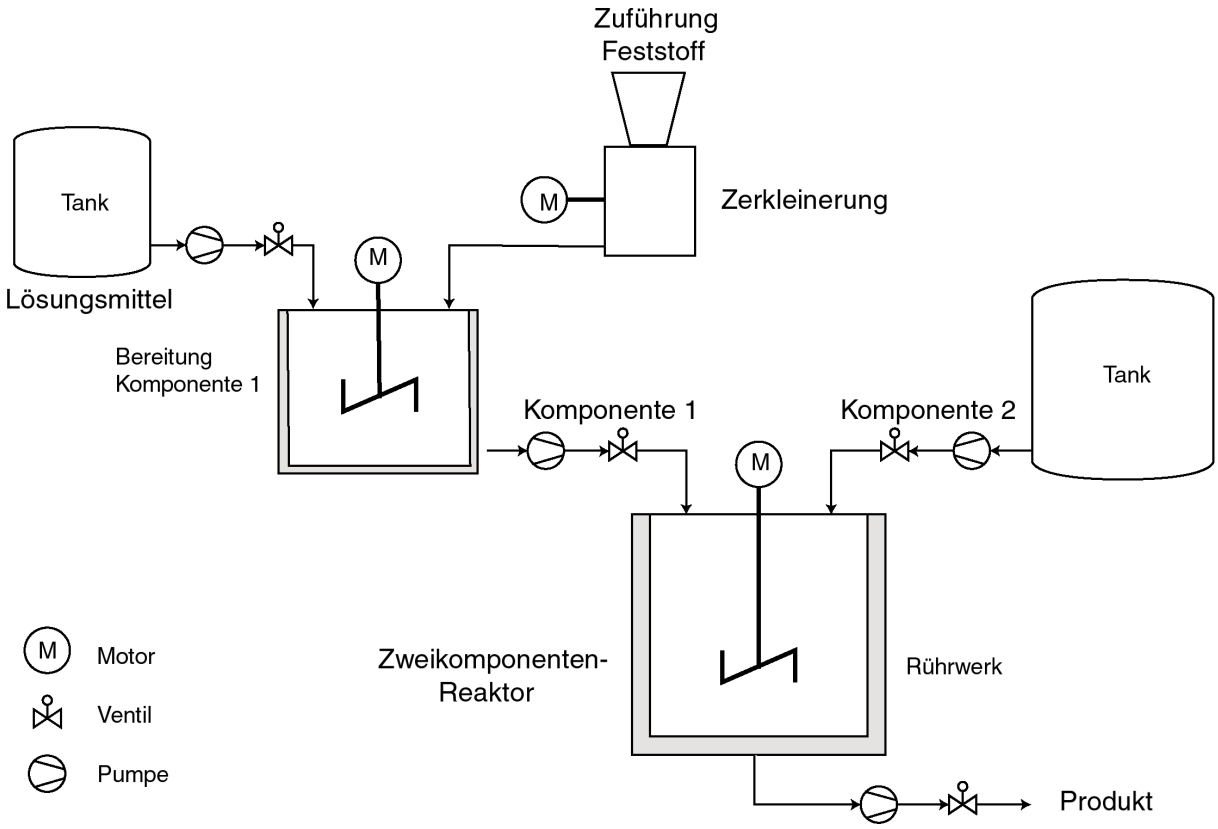


Abbildung 2.4.: Ein Zweikomponentenreaktor mit Zuführungseinheiten.

Die Reaktion wird über die Temperatur gesteuert, der Reaktor muss daher mit einem Heiz- und einem Kühlsystem ausgestattet sein. Die Verfahrensvorschrift gibt im Einzelnen folgenden Ablauf vor:

- die zwei Komponenten K1 und K2 müssen zu gleichen Volumenanteilen im Reaktor vorliegen
- die Reaktion läuft bei 40°C an
- die Temperatur muss unter 80°C bleiben, weil das Produkt sonst aushärtet
- diese Temperatur wird 10 Minuten gehalten

- während dieser Zeit muss Wärme zugeführt werden (endotherme Reaktion) und das Rührwerk laufen
- danach muss der Reaktorinhalt auf 20°C abgekühlt werden wodurch eine weitere Reaktion gestoppt wird.

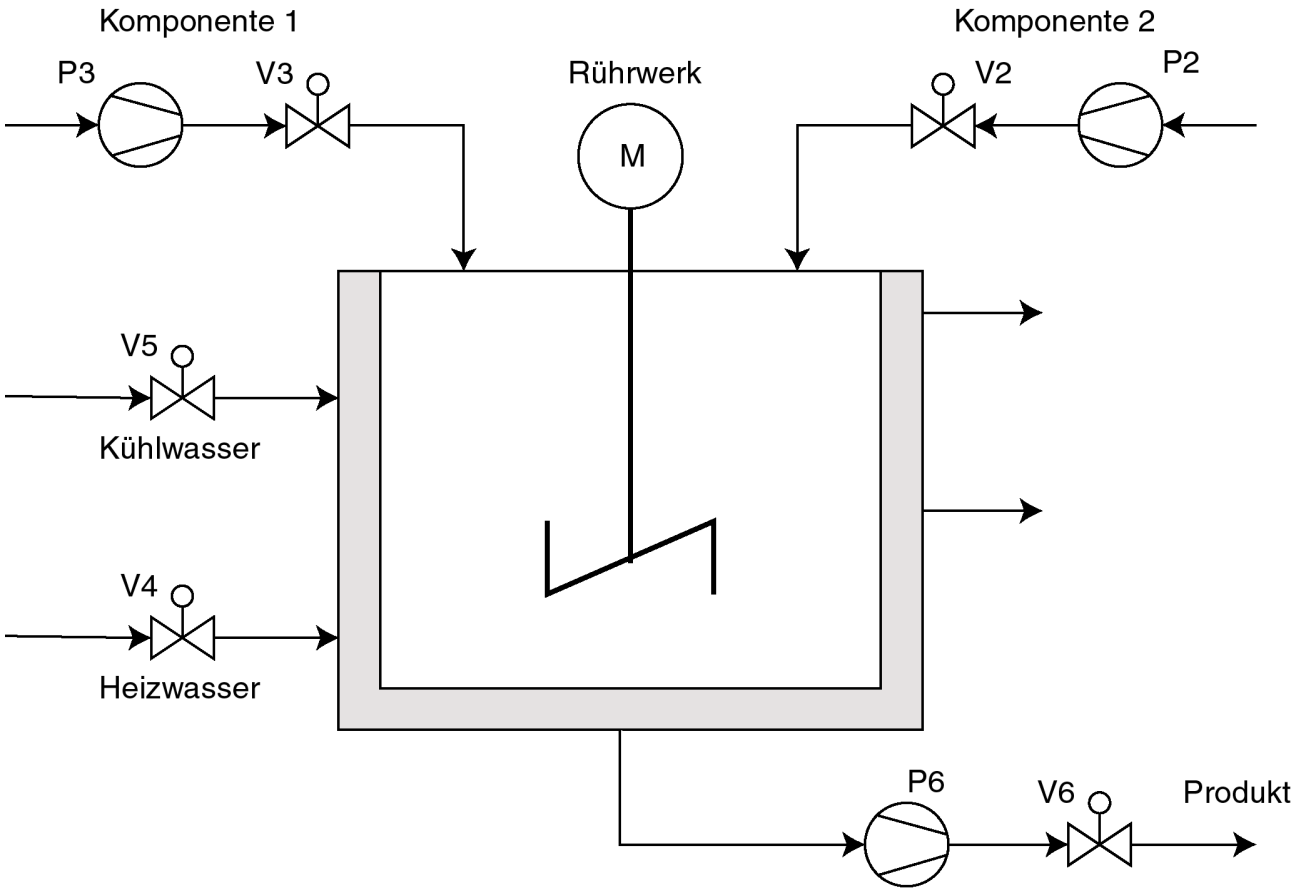


Abbildung 2.5.: Der Zweikomponentenreaktor mit Heizung, Kühlung Rührwerk und Fühlern.

Übung: 2.1 Welche Sensoren, Anzeigeelemente und Bedienelemente werden gebraucht?

Übung: 2.2 Was kann bei manueller Bedienung alles schief gehen und zu Qualitätsverlust führen?

Übung: 2.3 Was kann bei manueller Bedienung evtl. zur Anlagenbeschädigung führen?

Übung: 2.4 Was kann bei manueller Bedienung zu unnötigem Energieverlust führen?

Zusatzanforderungen des Auftraggebers könnten sein:

- Registrierung aller Mengen an Ein- und Ausgängen
- Not-Aus (Vorschrift)
- Handsteuerung muss immer noch möglich sein

2.3. Kurze Erwähnung: Steuern und Regeln

Steuern

Von einem Bedienelement geht eine Wirkung auf ein Stellelement aus, dieses beeinflusst den Prozess. Keine Messung des Istwerts, offene Wirkungskette.

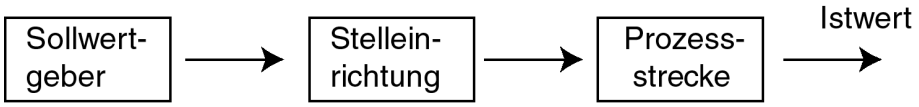


Abbildung 2.6.: Eine Steuerkette ist offen.

Beispiele: Einfache Herdplatte, Gaspedal im Auto, Dampflok

Regeln

Von einem Bedienelement wird der gewünschte Wert für die Regelgröße auf ein Vergleichselement (Subtrahierglied) gegeben. Dieses bildet die Differenz zwischen Sollwert und Istwert, der an der Regelstrecke gemessen wird. Die Regeldifferenz wird an den Regler übergeben, der nach bestimmten Verfahren (viele Typen!) ein Ausgangssignal für die Stelleinrichtung bildet. Dieses wirkt auf die Regelstrecke. Geschlossener Wirkungskreis, die Regelgröße wird nachgeführt auf den Sollwert, Stabilisierung!

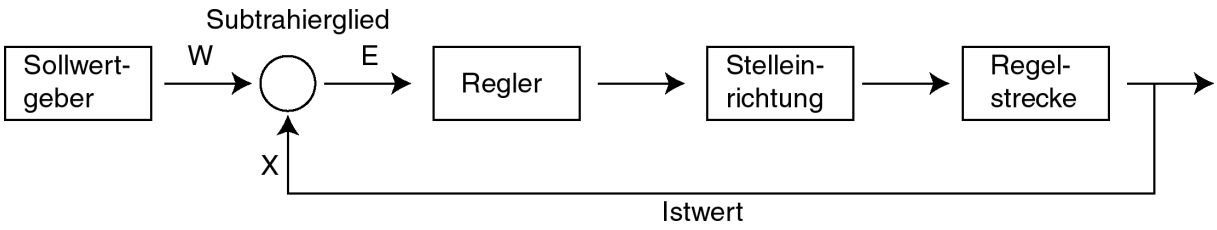


Abbildung 2.7.: Ein Regelkreis ist eine geschlossene Wirkungskette.

Beispiele: Elektrischer Backofen mit Temperaturwahl, Tempomat, Fahrtregelung bei Lokomotiven, Marktpreise,

Regelkreise sind ein sehr komplexes und vielfältiges Thema. In der Robotertechnik werden zur Steuerung der Antriebe kaskadierte Regelkreise eingesetzt. Dabei kann z.B. der Ausgangswert des Lagereglers als Sollwert für den Drehzahlregler benutzt werden; dessen Ausgangswert wiederum ist der Sollwert für den Stromregler. Wenn alle Regler gut eingestellt sind, führt das dazu, dass die Sollposition schnell, aber sanft und genau angefahren wird. (Abb. 2.8)

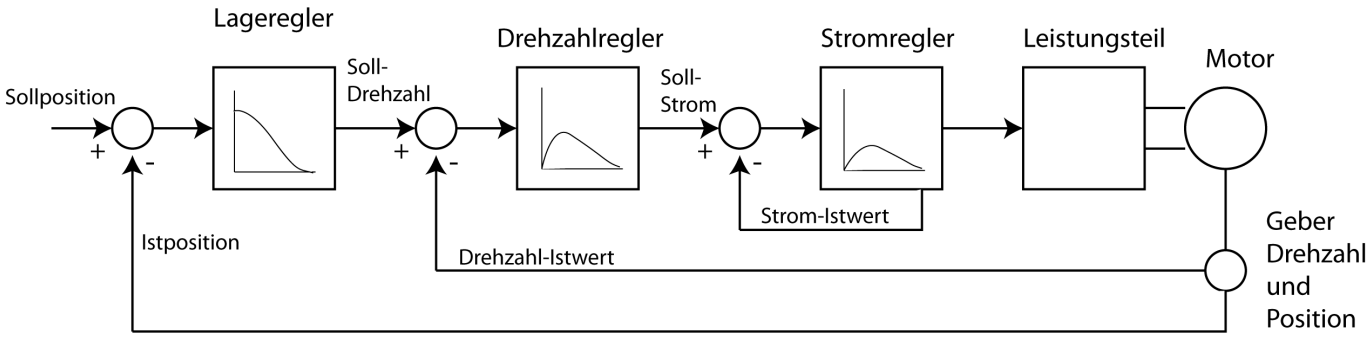


Abbildung 2.8.: Kaskadierte Regler in der Antriebssteuerung.

3. Elementare Begriffe zu stationären Robotern

3.1. TCP, Effektor, Armglieder, Gelenke, Freiheitsgrade

Effektor (auch Tool oder Hand) Bearbeitungswerkzeug am Ende des Roboterarmes, wird vom Roboter zum Werkstück geführt. Große Vielfalt an Effektoren, wird oft vom Anwender speziell gefertigt.

Tool Center Point (TCP) Der Hauptbezugspunkt des Effektors, z.B. bei Schweißwerkzeugen der Aufsetzpunkt des Schweißbogens, bei Klebewerkzeugen der Austrittspunkt des Klebers, bei Greifern der Mittelpunkt der Verbindungslinie der Greifbacken.

Armteile Bewegliche Verbindungsglieder des Roboters. Jedes Armteil ist über zwei Gelenke mit anderen Armteilen verbunden. Eine Ausnahme ist das letzte Armteil bei serieller Kinematik, dort ist statt des zweiten Gelenks der Effektor. Die Armteile werden durch Parameter beschrieben: Länge des Armteils und Lage der Gelenkachsen. Hat man alle Parameter der Gelenke, so ist damit die Kinematik eines Roboters vollständig beschrieben. Ein sehr systematischer Ansatz dazu sind die so genannten *Denavit-Hartenberg-Parameter* (s.Abschnitt A)

Gelenktypen

Translatorische Gelenke (Linearachsen): nächstes Armglied bewegt sich auf gerader Bahn vorwärts. Die Montage erfolgt auf Roll oder Gleitschienen, der Antrieb z.B. hydraulisch.

Rotatorische Gelenke (Rotationsachsen, Drehachsen), das jeweils nächste Armglied dreht sich um eine Rotationsachse. Montage erfolgt auf Gelenkbolzen. Rotatorische Gelenke finden wir z.B. auch bei Mensch und Bagger.

Freiheitsgrade (Degrees of Freedom, DOF) Ein im Raum frei beweglicher Körper hat 6 Freiheitsgrade: ($f = 6$)

- 3 Freiheitsgrade für die Position im Raum (Ort) (x, y, z).
- 3 Freiheitsgrade für die Orientierung (ϕ_x, ϕ_y, ϕ_z).

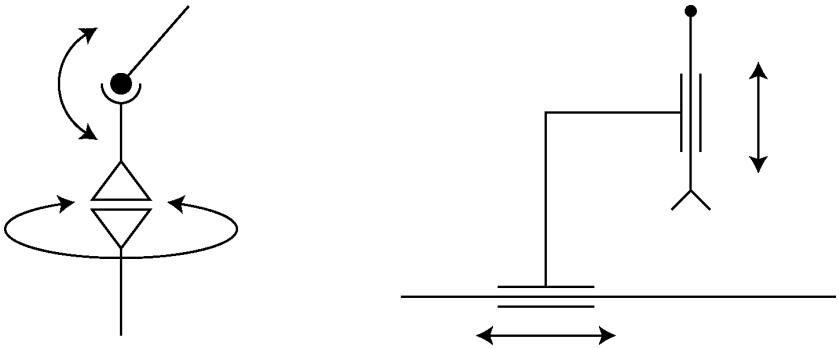


Abbildung 3.1.: Symbole für Roboterachsen nach DIN 2861. Links: Geometrie mit zwei Rotationsachsen, rechts: Geometrie mit zwei Translationsachsen.

Jedes Robotergelenk verleiht dem Roboter auch einen Freiheitsgrad. Deshalb gilt: Die Anzahl der Robotergelenke F ist auch die Anzahl f der Freiheitsgrade des Roboters . Ausnahme: In ungünstigen Stellungen, den so genannten Singularitäten, (z.B. zwei Achsen deckungsgleich) entfällt ein Freiheitsgrad und es kommt zu Problemen in der Bewegung.

$$f = F$$

Antriebe, Motoren Damit der Roboter sich bewegen kann braucht er für seine Gelenke Antriebe. Das sind meistens Elektromotoren die so angesteuert werden, das die gewünschte Position genau und weich angefahren wird. Beim Roboter mit serieller Kinematik haben alle Gelenke einen Antrieb, bei paralleler Kinematik gibt es auch passiver Gelenke zur Führung.

3.2. Parallele und serielle Kinematiken, kinematische Ketten

Serielle Kinematik Gekennzeichnet durch eine *offene kinematische Kette*: Jedes Armteil ist genau an einer Stelle mit dem nachfolgenden Armteil verbunden. Das letzte Armteil ist mit dem Effektor verbunden, dort endet die offene kinematische Kette. Es gibt nur einen Verbindungspunkt zum Sockel.

Parallele Kinematik Gekennzeichnet durch eine geschlossene kinematische Kette, die kein Ende hat. Der Effektor lässt sich vom Sockel aus auf mehreren Wegen erreichen. Es gibt mehrere Verbindungspunkte zum Sockel.

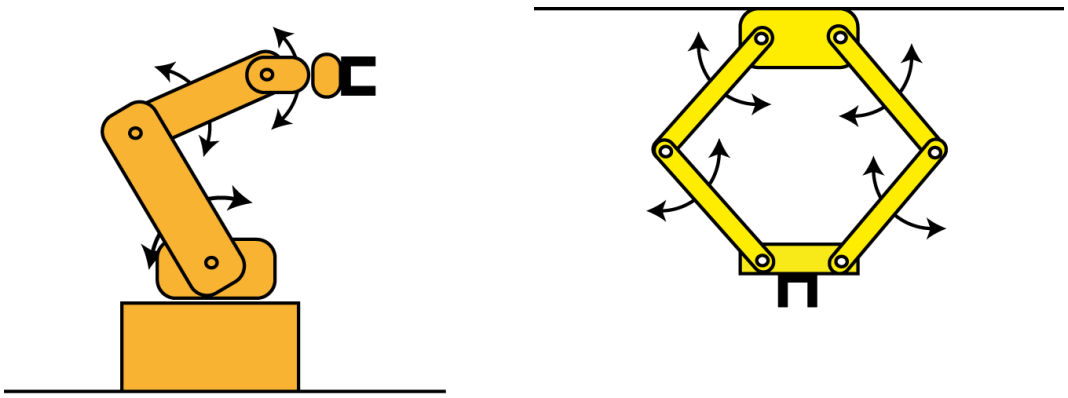


Abbildung 3.2.: Grundtypen der Kinematik von Industrierobotern: Serielle und parallele Kinematik

3.3. Roboter mit 4,5,6 oder mehr Achsen

Roboter mit 6 Achsen

Ein Roboter mit 6 beweglichen Armgliedern hat 6 Gelenkfreiheitsgrade ($F = 6$, 6-DOF). (Der Greifermotor ist kein Gelenk im Sinne der Kinematik). Im Regelfall resultieren daraus auch 6 Freiheitsgrade für den Effektor: $f = F = 6$.

Der 6-achsige Roboter kann daher in seinem Arbeitsraum jeden Punkt mit dem Effektor in beliebiger Orientierung erreichen, soweit nicht Einschränkungen im Achsschwenkwinkel bestehen. das macht den 6-achsigen Roboter universell geeignet, er ist heute der Standardtyp im Industriebereich. Die Achsen sind rotatorisch oder translatorisch, viele Möglichkeiten der Anordnung.

Achsenbezeichnungen

3 Hauptachsen (Grundachsen) Achse 1 – Achse 3, ermöglichen das Erreichen der gewünschten Position im Raum mit dem Effektor.

3 Handachsen (Nebenachsen) Achse 4 – Achse 6, ermöglichen die Einstellung der gewünschten Orientierung der Hand; tragen nur wenig zu Positionsänderungen bei.

Aufbau der offenen kinematischen Kette beim 6-achsigen seriellen Roboter (Standardtyp der Industrie):
Sockel (Armteil 0) – Achse1 – Armteil1 – Achse2 – Armteil2 – Achse3 – Armteil3 –Achse4 – Armteil4 – Achse5 – Armteil5 – Achse6 – Effektor.

Die Bezeichnung der Bauart beschreibt in der Regel die Grundachsen, z.B. RRT, TRR. RRR = Knickarm. *Der 6-achsige RRR-Industrieroboter ist heute der meistverwendete Roboter, eine Art Standard der Industrie.*

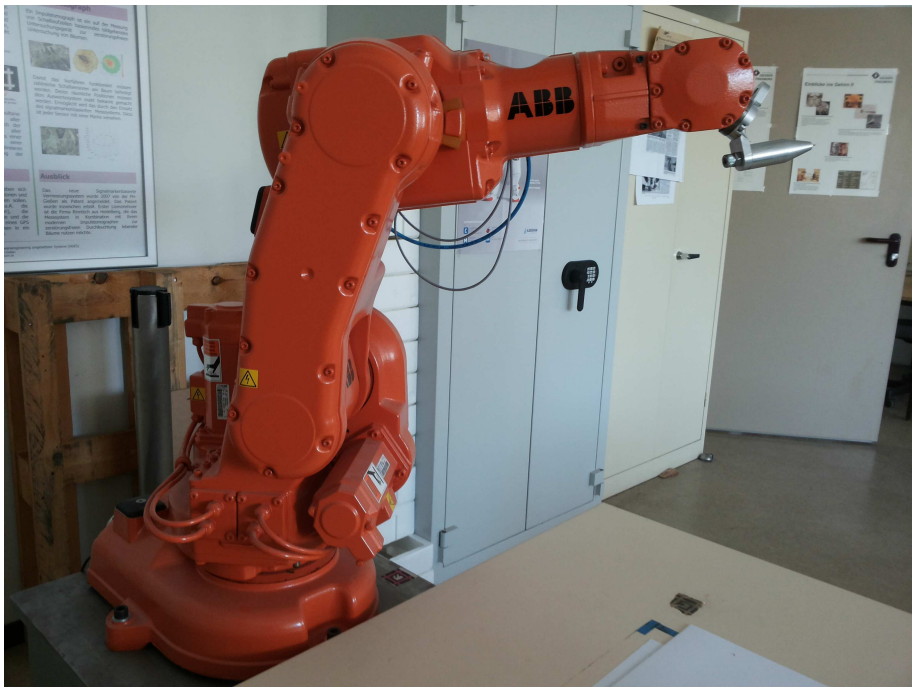


Abbildung 3.3.: Stationärer 6-achsiger Industrieroboter ABB IRB140 (Abb. KW)

Roboter mit 4 oder 5 Achsen

Auch hier gibt es 3 Grundachsen, aber nur ein oder zwei Handachsen. Es besteht daher nur eingeschränkte Effektor-Orientierbarkeit. Dies kann oft akzeptiert werden, z.B. beim Lackieren, Bohren, Klebemittelstrang auftragen, Bestücken, u.a.m.

Beispiele: SCARA (Selective Compliance Assembly Robot Arm), mit 4 Achsen ($f = F = 4$) einsetzbar für Pick and Place, oder mit 5 Achsen ($f = F = 5$) einsetzbar für lackieren, auftragen von Klebestränge, schweißen und nieten.

Roboter mit mehr als 6 Achsen

Selten vorkommend, diese Roboter besitzen natürlich auch nur 6 Freiheitsgrade im kartesischen Raum, z.B. $f = 6, F = 7$, es gibt redundante (überflüssige) Achsen. Bessere Steuerbarkeit, alternative Gelenkstellungen verfügbar (=Mehrdeutigkeiten).

Beispiele für Redundanz: Parallele Translationsachsen, parallele Rotationsachsen.

3.4. Arbeitsraum, Kollisionsraum

Kollisionsbereich Der Raumbereich, der von dem Roboter insgesamt bei Ausführung aller Bewegungen und Berücksichtigung aller Teile benutzt wird. Der Kollisionsraum ist größer als der Arbeitsraum, weil es immer auch überstehende Teile am Roboter gibt.

Arbeitsbereich Der Raumbereich, der mit dem TCP erreicht werden kann (DIN-Def. etwas anders). Nur ein Teil des Arbeitsraumes kann mit frei wählbarer Orientierung erreicht werden.

Roboter mit Translationsachsen haben

- Bewegungen, die für den Menschen gut vorstellbar sind,
- einfache Koordinatentransformationen (Orientierung nachfolgender Achsen bleibt erhalten),
- ein Koordinatensystem, das gut zur Geometrie passt, falls kartesisch.

Roboter mit Rotationsachsen haben

- ein gutes Verhältnis Arbeitsraum – Kollisionsraum,
- gute Steifigkeit, kleines Gelenkspiel,
- hohe Arbeitsgeschwindigkeit.

Trend: Knickarmroboter, kann in Öffnungen greifen und ist flexibel einsetzbar.

Beispiel: TR-Roboter

Sockel Fest montiert, Armteil 0, darauf befindet sich die Achse 1

Achse 1 Translationsachse, z.B. Armteil auf Rollschiene

Armteil 1 über Achse 1 mit Sockel verbunden

Achse 2 An Armteil 1 befestigte Rotationsachse

Armteil 2 über die Achse 2 an Armteil 1 befestigt

Effektor am Ende von Armteil 2 befestigt, dort TCP

Ende der kinematischen Kette

Ein solcher Roboter (obwohl nur zwei Freiheitsgrade) hat schon praktischen Wert, kann z.B. auf einer ebenen Fläche Bohrungen vornehmen oder Nieten setzen. Wir wollen ihn den TR-Roboter nennen. Der TR-Roboter wird im Laufe

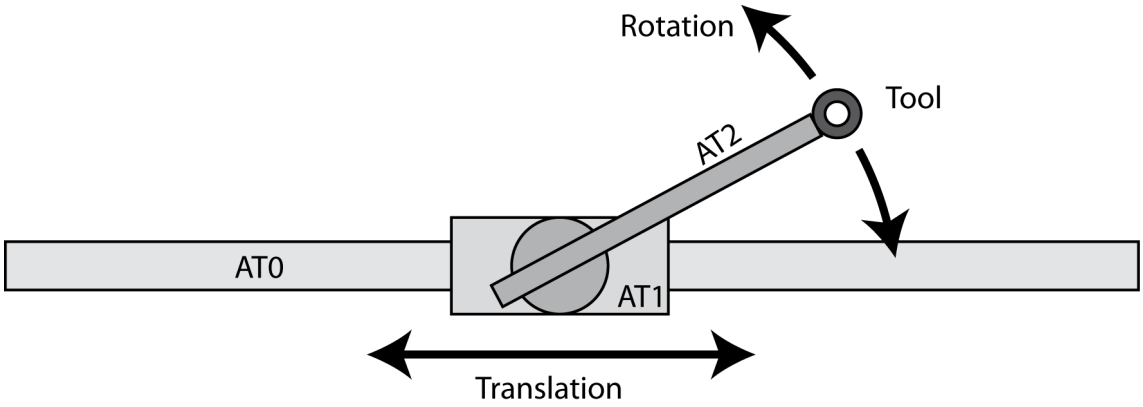


Abbildung 3.4.: Ein einfacher zweiachsiger Roboter mit TR-Kinematik.

dieses Kurses immer wieder auftauchen, weil er relativ einfach ist aber man trotzdem viele allgemein wichtige Zusammenhänge daran verstehen kann.

An dem TR-Roboter kann man Arbeits und Kollisionsraum ermitteln, das Ergebnis ist in Abb.3.5 gezeigt. Der Arbeitsraum besteht aus der Menge der Raumpunkte, die der TCP erreichen kann, hier eine ebene Fäche mit ovaler Umrandung. Wegen der Ausdehnung des Tools ist der Kollisionsraum ein wenig größer als der Arbeitsraum.

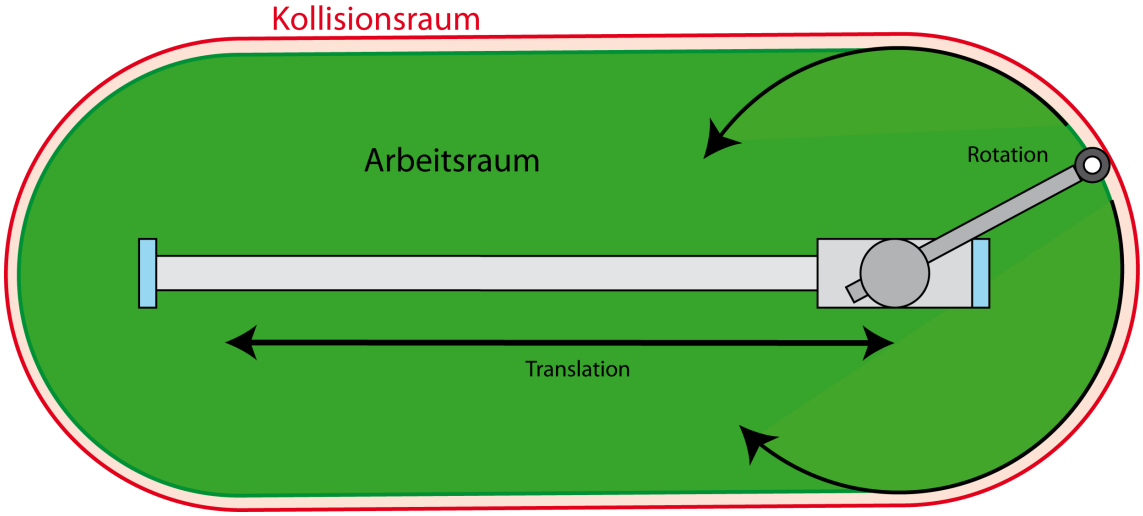


Abbildung 3.5.: Arbeits- und Kollisionsraum des zweiachsigern TR-Roboters.

4. Bahnsteuerung

Die Kinematik hat sich nur statisch mit Beschreibung und Berechnung von Armstellungen befasst. Die Bahnsteuerung befasst sich nun mit dem Bewegungsablauf, mit dem ein Roboterarm die berechnete Stellung erreicht.

4.1. Interpolation

Die Stellung einer Achse wird bei einer Rotationsachse als Winkelkoordinate beschrieben, bei einer Linearachse als Wegkoordinate. Die folgenden Überlegungen gelten für beide Arten von Achsen, der Bahnparameter $s(t)$ ist je nach Achsentyp als Winkel oder Weg aufzufassen.

Man erhält eine bessere Kontrolle über die Bewegung, wenn man Zwischenpunkte im Gelenkkoordinatenraum berechnet. Die Zwischenpunkte werden in Gelenkkoordinaten formuliert und in einer Tabelle abgelegt. Von dort werden sie nacheinander an die Servokreise übergeben. Ein günstiges Verhalten (flüssige Bewegung) erhält man, wenn man den neuen Sollwert übergibt, kurz bevor der aktuelle Sollwert erreicht ist (sog. „Hunderennen-Technik“, [6]). Die Interpolationswerte werden an die Steuerung übergeben, die dann auf Hardware-Ebene (Servokreise), evtl. mit vorheriger Feininterpolation, die Ansteuerung der Motoren vornimmt.

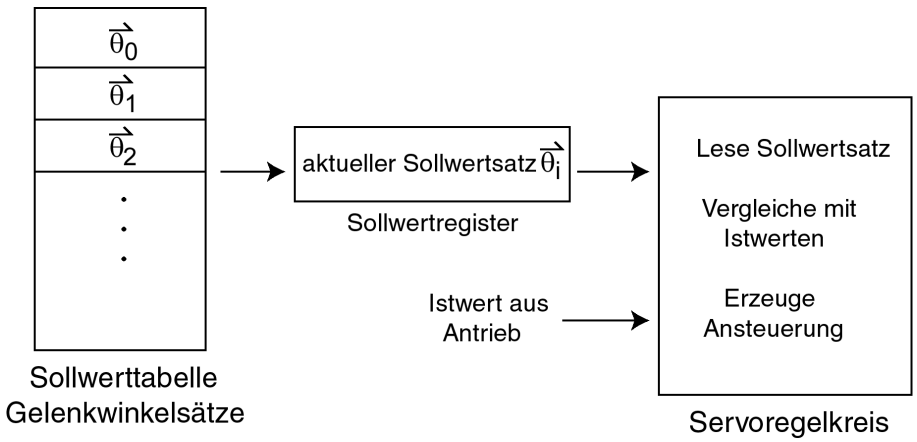


Abbildung 4.1.: Übergabe der Sollwerte an den Servokreis (ohne Feininterpolation).

Falls man dabei versucht, die Strecke in gleichmäßige Intervall aufzuteilen, treten im ersten Intervall u.U. sehr hohe Kräfte auf.

Beispiel Aus dem Stillstand einer Linearachse im 1.Intervall 5 mm Weg in 5 ms. Dann ist

$$a = \frac{2s}{t^2} = \frac{10mm}{25ms^2} = \frac{10^{-2}m}{25 \cdot 10^{-6}\, s^2} = 400\, m/s^2$$

Der Fehler liegt darin, dass keine Beschleunigungen berücksichtigt wurden. Man arbeitet besser mit einem Geschwindigkeitsprofil, bei dem die Geschwindigkeit zunächst Null ist, dann auf einen Maximalwert gesteigert wird, eine Zeitlang maximal bleibt, und dann wieder auf Null verringert wird. Beschleunigungs- und Bremsphase entsprechen sich spiegelbildlich. Für die Beschleunigungsprofile gibt es mehrere Möglichkeiten:

Rampenprofil Die Geschwindigkeit wird durch eine konstante Beschleunigung a während einer Beschleunigungszeit t_B gleichmäßig bis zum Maximalwert gesteigert: $\dot{s}(t) = at$. Die Endgeschwindigkeit ist at_B. Die Beschleunigung und damit die Kraft ändert sich dabei sprunghaft (unstetig) von 0 auf den Maximalwert und wieder auf Null.

Sinoidenprofil Hierbei wird die Beschleunigung mit einem $\sin^2$-Profil weich bis zum Maximalwert gesteigert und fällt dann weich wieder auf Null ab:

$$\ddot{s}(t) = \sin^2\left(\pi \frac{t}{t_B}\right) \quad \text{für} \quad 0 \leq t \leq t_B$$

Beim Sinoidenprofil ist die Veränderung der Kraft stetig und weich, die Bewegung wird ruckfrei und flüssig.

Bahnsteuerungspolynome Man kann für den Bahnparameter ein Steuerungspolynom 4. Ordnung verwenden [6]. Dann werden die Kräfte durch ein Polynom 2. Ordnung beschrieben, was ebenfalls einen weichen Übergang zwischen zwei beliebigen Geschwindigkeitswerten ermöglicht.

4.2. Punkt-zu-Punkt-Steuerung (PTP)

PTP steht für *point to point control*, also Punkt-zu-Punkt-Steuerung. dazu müssen nur die anzufahrenden Punkte einer Rückwärtstransformation unterworfen werden, zwischen diesen Punkten wird nur mit Achskoordinaten gearbeitet. Bei einer einfachen PTP-Steuerung werden die Sollwerte der Achskoordinaten einfach an die Regelungen übergeben, so dass die Servomotoren die gewünschten Sollwerte anfahren können. Dabei ergeben sich folgende Probleme:

- Die Bahn des Effektors zum Zielpunkt ist unkontrolliert und nicht vorhersehbar,
- die auftretenden Geschwindigkeiten und Beschleunigungen hängen von der Einstellung der Servokreise ab und sind nicht vorhersehbar.

Asynchrone und synchrone PTP-Steuerung

Bei der asynchronen PTP-Steuerung werden alle Achsen gleichzeitig gestartet um den neuen Sollwert zu erreichen und alle laufen mit maximaler Geschwindigkeit. Das hat zur Folge, dass die Achsen nacheinander das Ziel erreichen. Bei diesem Verfahren weicht die Bahn des Effektors u.U. relativ stark von der direkten Verbindungslinie zwischen Start- und Zielpunkt ab.

Man hat daher die synchrone PTP-Steuerung entwickelt, bei der die Achsen gleichzeitig starten und gleichzeitig stoppen (s. Abb. 4.2). Dazu geht man wie folgt vor:

1. Für jedes Gelenk wird aus Verfahrstrecke, maximaler Geschwindigkeit, und Interpolationsregeln die Bewegungsdauer berechnet.
2. Die Achse mit der größten Bewegungsdauer wird die Leitachse.
3. Die übrigen Achsen werden in ihrer Bewegung verlangsamt, so dass ihre Bewegungsdauer gleich mit der Leitachse wird.
4. Bei *vollsynchroner PTP-Steuerung* werden auch die Beschleunigungsphasen angepasst, so dass sie gleich lang dauern wie bei der Leitachse
5. die Interpolationstabelle wird für alle Achsen berechnet.
6. Eintragung der Gelenkkoordinaten in die Sollwerttabelle, von dort Übergabe an die Steuerungshardware,
7. Evtl. Feininterpolation durch die Steuerungshardware, Übergabe an Servokreise, von dort Motoransteuerung.

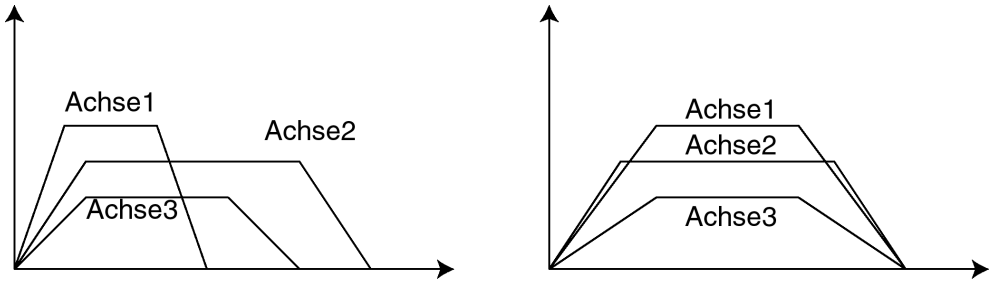


Abbildung 4.2.: Links: Asynchrone PTP-Steuerung, Rechts: Synchrone PTP-Steuerung.

Die synchrone PTP-Steuerung erzeugt Bewegungen, die nicht so stark von der direkten Verbindungslinie Start–Ziel abweichen. Die synchrone PTP-Steuerung ist heute üblich und fast jeder Roboter kennt diese Steuerungsart. Man muss aber wissen, das auch hier erheblich Bahnabweichungen auftreten können. Um ein Gefühl dafür zu bekommen, betrachten wir auch hier wieder unseren TR-Roboter. (siehe Abschn. **??**) In Abb.4.3 ist eine PTP-Bewegung dargestellt von der Startposition (Pos.1) ganz rechts zur Zielposition (Pos.5) ganz

links. Es gibt drei Zwischenpositionen (Pos.2–4) Die Translationskoordinate a ändert sich bei jedem Schritt um den Betrag $-x_a$, der Winkel γ ändert sich jeweils um 20^o. Auf der Ebene der Achsensteuerung besteht also eine ganz gleichmäßige Bewegung. Im kartesischen Raum dagegen ergibt sich eine krumme Bahn (Blaue Linie). Das wird sich auch nicht ändern, wenn man mehr Interpolationspunkte, also Zwischenpositionen einführt. Bei Robotern

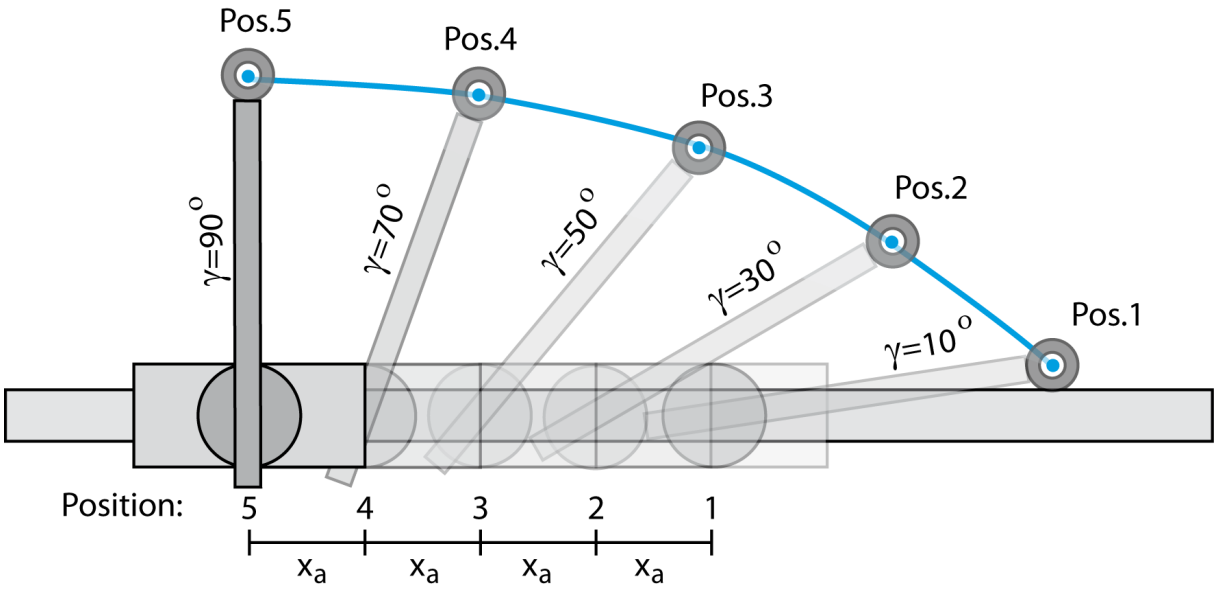


Abbildung 4.3.: Bewegung mit PTP-Steuerung am TR-Roboter in 5 Bewegungsphasen, Erläuterung im Text.

mit mehr Achsen, z.B. 5 oder 6, werden diese Effekte noch stärker. Es kann zu regelrechten „Überschwingern“ und Schleifen kommen, dabei bewegt sich der TCP zunächst über das Ziel hinaus und dann wieder zurück. Damit ist die PTP-Steuerung für alle Bewegungen ungeeignet, bei denen es auf eine genaue Bahnführung ankommt. Beispiele dafür sind Schweißen, Montage, Kleben, Falzen, Entgraten usw.

Eine versehentliche Programmierung von PTP-Steuerung anstelle einer geraden Bahn kann zu einem Crash führen!

Andererseits ist PTP eine gute Wahl, wenn genug Platz für die Bewegung vorhanden und die Bahn zweitrangig ist. Denn PTP ist eine einfache Steuerungsvariante, die in jedem Punkt des Arbeitsraumes problemlos funktioniert. Das trifft auf die nachfolgend beschriebene CP-Steuerung nicht zu.

4.3. Kartesische Bahnsteuerung (Continous Path)

Für viele Bewegungen eines Roboters ist eine exakte Einhaltung der gewünschten Bahn notwendig. Das trifft z.B. dann zu, wenn mit der Bewegung eine Bearbeitung des Werkstücks verbunden ist, z.B. Schweißen, Kleben usw. Eine PTP-Steuerung geht hier nicht. (siehe oben)

Wie kann nun eine gerade Bahn erreicht werden? Wir betrachten dazu wieder unseren TR-Roboter. Man kann zwischen Startpunkt und Zielpunkt eine gerade Bahn ziehen und auf dieser Bahn in gleichmäßigen Abständen Zwischenpunkte (Interpolationspunkte) eintragen. Im Beispiel (Abb. 4.4) sind es

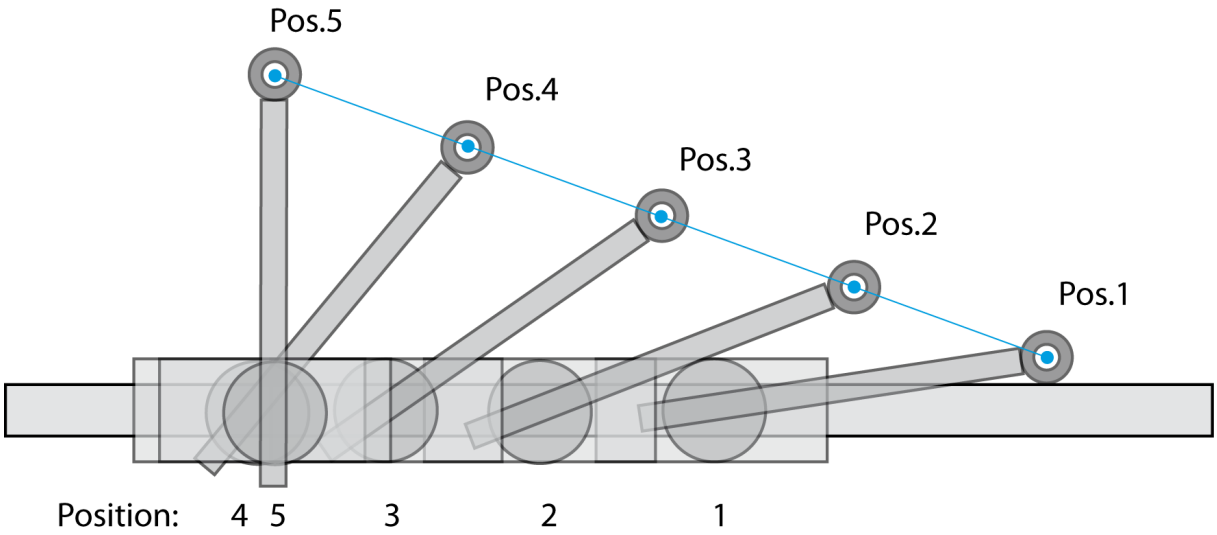


Abbildung 4.4.: Bewegung mit PTP-Steuerung am TR-Roboter in 5 Bewegungsphasen

drei Zwischenpositionen. Nun muss für jede Zwischenposition eine Roboterstellung gefunden werden, das heißt einen Satz von Gelenkwinkeln, der den TCP jeweils genau ins Ziel bringt. Für unseren TR-Roboter ergibt sich, das die Gelenkkoordinaten (Translation+Rotation) sich dabei ungleichmäßig ändern müssen. Der Winkel ändert sich am Anfang wenig und am Ende stärker. Noch interessanter ist die Bewegung auf der Translationsachse: Am Ende der Bewegung stoppt der Schlitten sogar kurz und fährt ein Stück zurück.

Wenn man ausrechnet, wie die beiden Maschinenachsen d und α sich bewegen müssen, damit sich der TCP genau auf dieser gewünschten geraden Bahn bewegt. Mit den gewonnenen Gleichungen kann man diesen Effekt dann genauer untersuchen. Und in der Tat zeigt sich auch dabei die leichte Rückwärtsbewegung der d-Achse. Diese Art der Berechnung heißt *kinematische Rückwärtstransformation.*

Bei der *kartesischen Bahnsteuerung* wird ein Roboter so angesteuert, dass der TCP sich im kartesischen Raum exakt auf einer Geraden oder einem Kreisbogen bewegt. Dabei wird eine Anzahl von Interpolationspunkten im kartesischen Raum (Weltkoordinaten) auf der gewünschten Bahnkurve berechnet und vom Effektor durchfahren. Die Steuerung muss also für alle diese Punkte die richtigen Gelenkwinkel bestimmen. Dies geschieht mathematisch mit der so genannten Rückwärtstransformation. Die Bewegung zwischen den Interpolationspunkten kann dann mit PTP-Steuerung erfolgen. Im Extremfall liegen die Interpolationspunkte so dicht, dass man von einer *kontinuierliche Steuerung* oder *continuous path (CP)* spricht. Der Effektor des Roboters bewegt

sich dann – bis auf kleine Restabweichungen – exakt auf der interpolierten Linie. Um eine volle Kontrolle über $s(t), v(t), a(t)$ zu erhalten und z.B. sanfte Beschleunigung und ein sanftes Abbremsen zu erreichen, werden die Interpolationspunkte am Anfang und am Ende der Bewegung entsprechend einem der oben erwähnten Beschleunigungsprofile dichter liegen. Der Algorithmus zur Durchführung einer kartesischen Interpolation ist:

1. Berechnung der Translationsstrecke und der Orientierungsänderung,
2. Berechnung der Bewegungsdauer als das Maximum aus den beiden Bewegungsdauern,
3. Festlegung der Geschwindigkeits- und Beschleunigungswerte,
4. Festlegung der Interpolationspunkte,
5. Ermittlung der Gelenkkoordinaten in den Interpolationspunkten durch Rückwärtstransformation
6. Eintragung der Gelenkkoordinaten in die Sollwerttabelle, von dort Übergabe an die Steuerungshardware,
7. Evtl. Feininterpolation durch die Steuerungshardware, Übergabe an Servokreise, von dort Motoransteuerung.

Die Funktion der Steuerung ist also auf mehrere Ebenen verteilt (Abb. 4.5). Etwas anders muss vorgegangen werden, wenn die Bewegung spontan umgesetzt werden soll, etwa beim Teach-In. In diesem Fall werden die Gelenkkoordinaten ständig online berechnet. Für die Bahnführung bei kartesischer Steuerung gibt es zwei Hauptvarianten:

1. **Linearinterpolation** Die Interpolationspunkte liegen auf einer Geraden, die Bahn wird definiert durch Startpunkt und Zielpunkt.

2. **Zirkularinterpolation** Die Interpolationspunkte liegen auf einem Kreissegment. Die Bahn wird definiert durch Startpunkt, Zielpunkt und einen Zwischenpunkt, der auf dem Kreissegment liegt.

In Abb. 4.6 sind die bei verschiedenen Steuerungsarten entstehenden Bahnen noch einmal quantitativ angedeutet. Da die meisten Werkstücke aus geraden und kreisförmigen Abschnitten aufgebaut sind, reichen diese Interpolationsarten für die meisten Aufgaben aus. Ein Beispiel dafür ist in Abb. 4.7 gezeigt, bei dem die Bahn aus zwei geraden und zwei kreisförmigen Abschnitten besteht.

Die Interpolation legt Zwischenstellungen fest mit Translationsanteil *und* Rotationsanteil. Ein Beispiel für eine Linearinterpolationen mit und ohne Orientierungsänderung (Rotationsanteil) ist in Abb. 4.8 gezeigt. Beispiele für Zirkularinterpolation mit und ohne Orientierungsänderung zeigt Abb. 4.9.

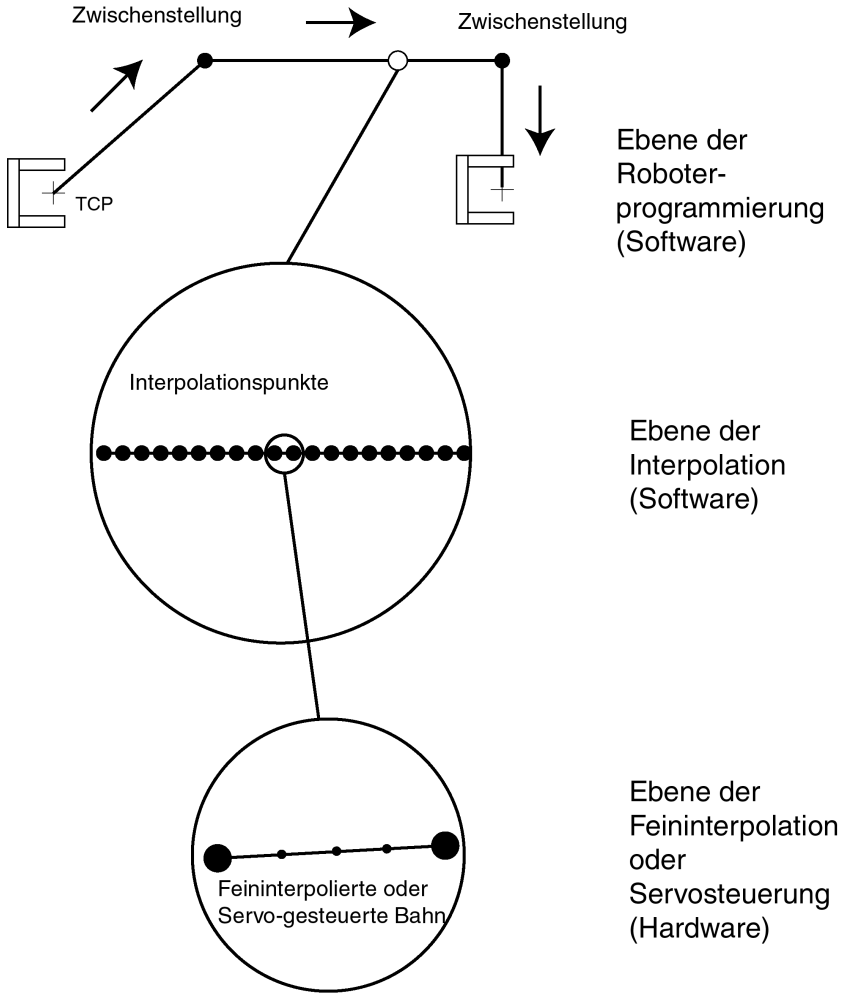


Abbildung 4.5.: Die Steuerung eines Roboters ist in mehrere Ebenen strukturiert.

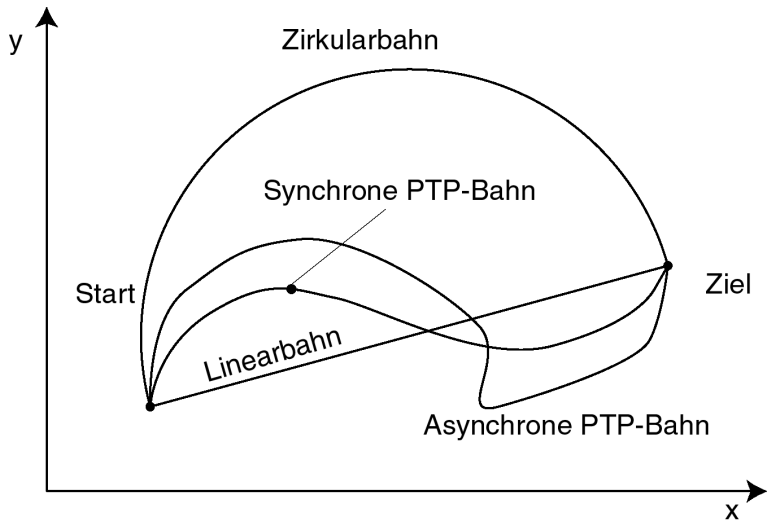


Abbildung 4.6.: Die bei den üblichen Steuerungsarten entstehenden Bahnen.

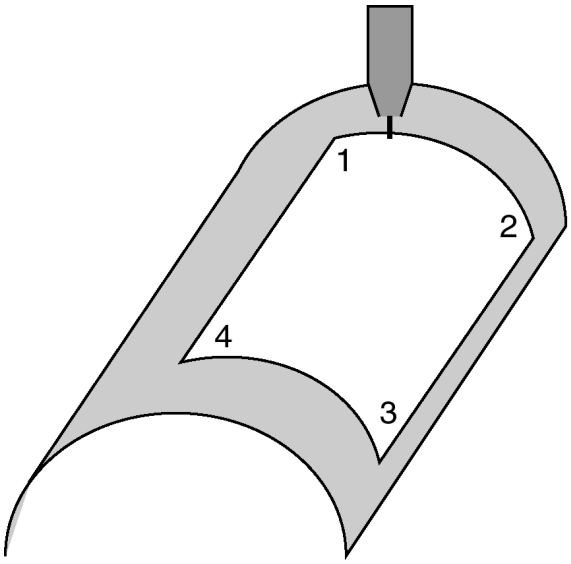
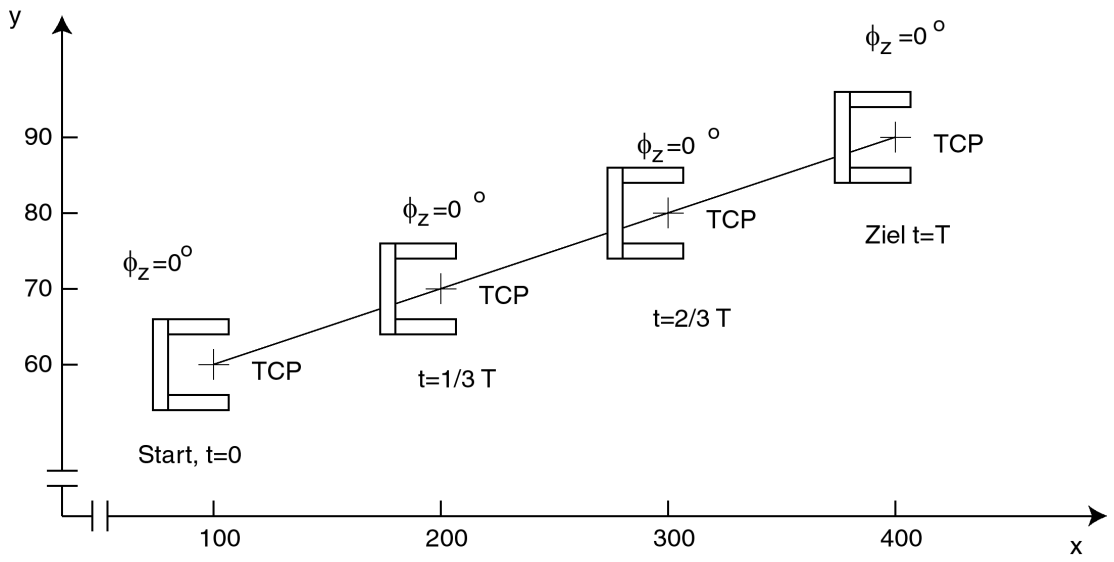
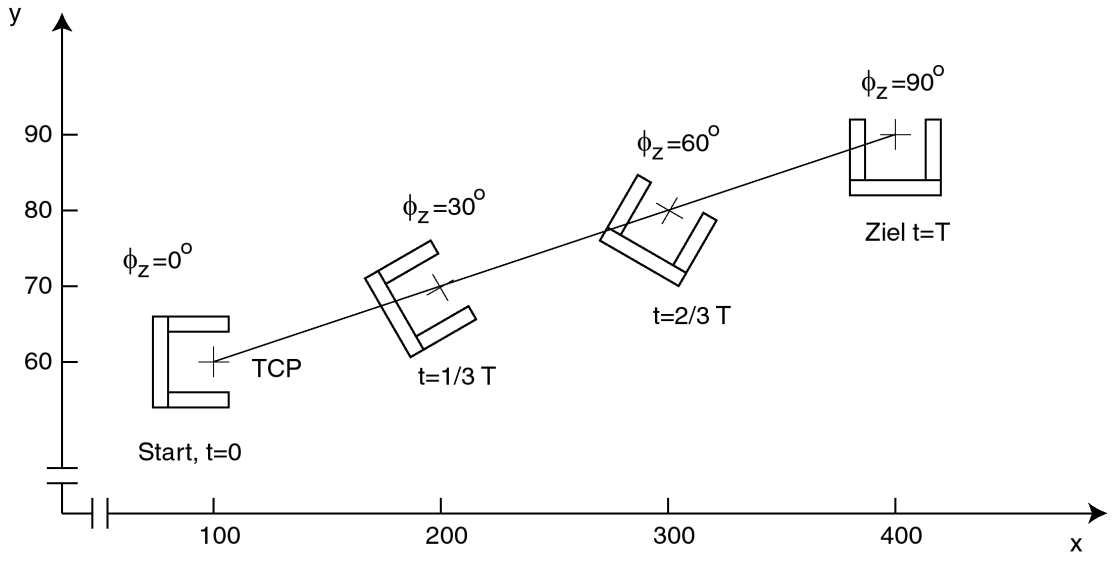


Abbildung 4.7.: Aus einem Halbzylinder wird ein Ausschnitt herausgesägt. Dabei wird der TCP auf den Bahnsegmenten 2–3 und 4–1 auf linearer CP-Bahn bewegt und auf den Abschnitten 1–2 und 3–4 auf zirkularer CP-Bahn.

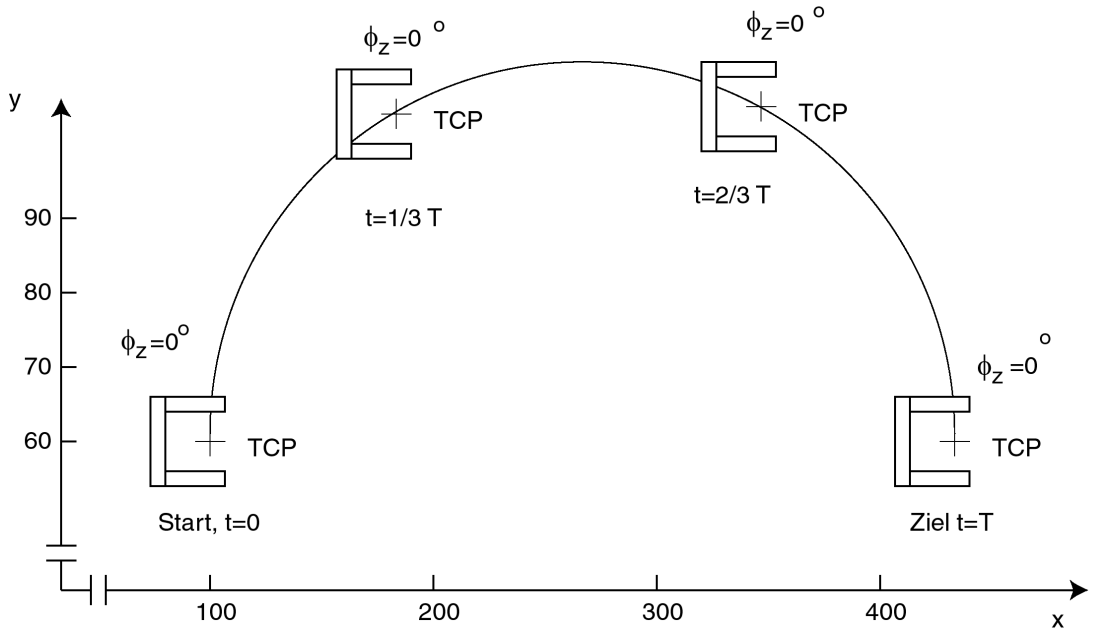


Lineare kartesische Interpolation (lineare CP-Bahn) ohne Orientierungsänderung

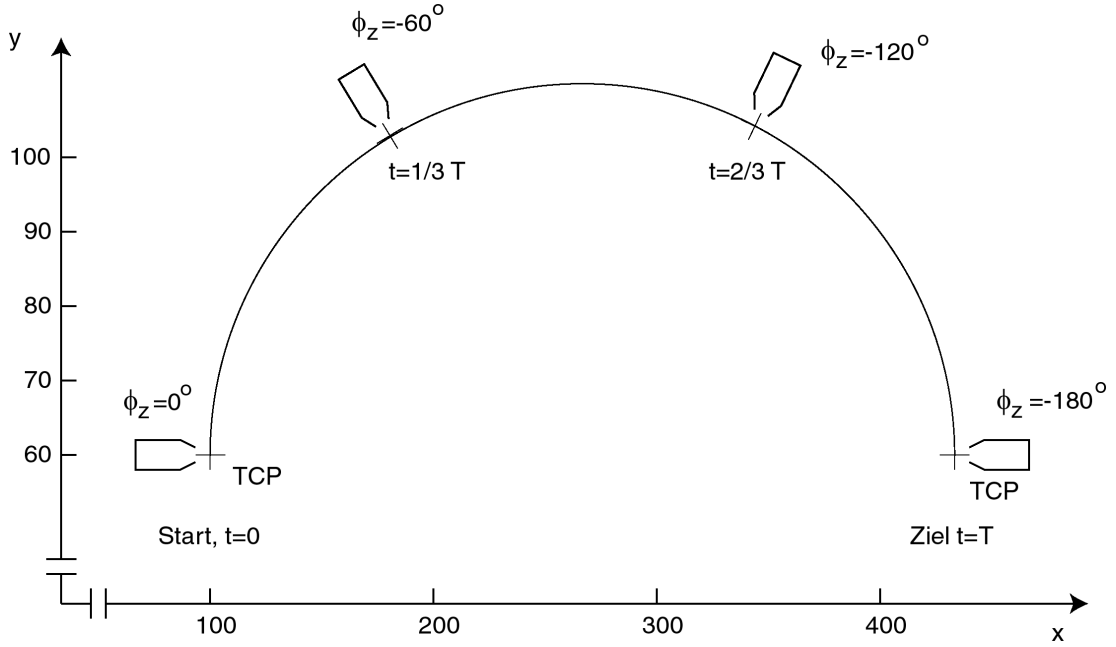


Lineare kartesische Interpolation (lineare CP-Bahn) mit Orientierungsänderung

Abbildung 4.8.: Lineare kartesische Interpolation (lineare CP-Bahn) ohne (oben) und mit (unten) Orientierungsänderung. Jeder Zwischenpunkt ist mit Position und Orientierung festgelegt. Im unteren Beispiel ist die Startstellung $x = 100, y = 60, \phi_z = 0^o$ und die Zielstellung $x = 400, y = 90, \phi_z = 90^o$. Es sind nur vier Interpolationspunkte gezeichnet, um eine gute lineare Bahn zu erhalten wird die Steuerung mehr als vier Punkte berechnen und durchfahren.



Zirkulare kartesische Interpolation (zirkulare CP-Bahn) ohne Orientierungsänderung



Zirkulare kartesische Interpolation (zirkulare CP-Bahn) mit Orientierungsänderung

Abbildung 4.9.: Zirkulare Interpolation ohne (oben) und mit Orientierungsänderung (unten). Es sind nur zwei Interpolationspunkte eingezeichnet, für eine gute kartesische Bahnsteuerung braucht man viel mehr Punkte.

4.4. Durchfahren von Zwischenstellungen ohne Anhalten

Bei den durch den Programmierer eingegebenen Zwischenstellungen einer Bahn gibt es zwei Möglichkeiten:

1. Am Zwischenpunkt ist ein Halt mit Stillstand des Roboters erforderlich. Gründe dafür können das Warten auf ein Signal, der Übergang in einen anderen Bearbeitungsgang, ein Werkzeugwechsel oder Ähnliches sein.

2. Am Zwischenpunkt ist kein Halt erforderlich. Das ist zum Beispiel der Fall, wenn der Zwischenpunkt eingefügt wurde um ein Hindernis zu umgehen.

Im zweiten Fall wäre ein Stillstand sogar unerwünscht, weil die zusätzlichen Abbrems- und Beschleunigungsphasen Zeit und Energie kosten (s. Abschn. 4.1). Für diesen Fall bieten die Steuerungen die Möglichkeit der Glättung der Bahn durch *Überschleifen*. Vor dem Erreichen der Zwischenstellung wird die Bewegung vorzeitig abgebrochen und stattdessen wird die nächste anzufahrende Stellung an die Steuerung übergeben, d.h. das nächste Bahnsegment begonnen. Beim Überschleifen wird die Zwischenstellung nur näherungsweise erreicht. Die Geschwindigkeit am Zwischenpunkt wird nicht Null, eine andere Bezeichnung ist daher *Fly-by-Punkt*. Die Servokreise bzw. die Feininterpolation der Gelenke sorgen dafür, dass sich die Richtung dabei sanft ändert. Für die Auslösung des vorzeitigen Wechsels auf das nächste gibt es zwei Kriterien:

Positionsüberschleifen

Der Effektor hat sich dem Zwischenpunkt bis auf einen Abstand R genähert. Man kann sich diese Zone also als Kugel mit Radius R um den Zwischenpunkt vorstellen, die so genannte *Überschleifkugel*. Mit Hilfe der Vorwärtstransformation wird festgestellt, wann der TCP in die Überschleifkugel eintritt.

Geschwindigkeitsüberschleifen

Die Geschwindigkeit hat bei der Annäherung an den Zwischenpunkt einen festgelegten Minimalwert unterschritten.

Das Überschleifen kann auf der Ebene der Gelenkwinkel erfolgen (PTP-Überschleifen) oder im kartesischen Raum (CP-Überschleifen). Beim CP-Überschleifen wird die Richtungsänderung innerhalb der Überschleifkugel bewirkt, beim Verlassen der Kugel ist der TCP wieder exakt auf der Bahn. In Abb. 4.10 ist ein Beispiel für das Positionsüberschleifen gezeigt. Man sieht an dem Beispiel,

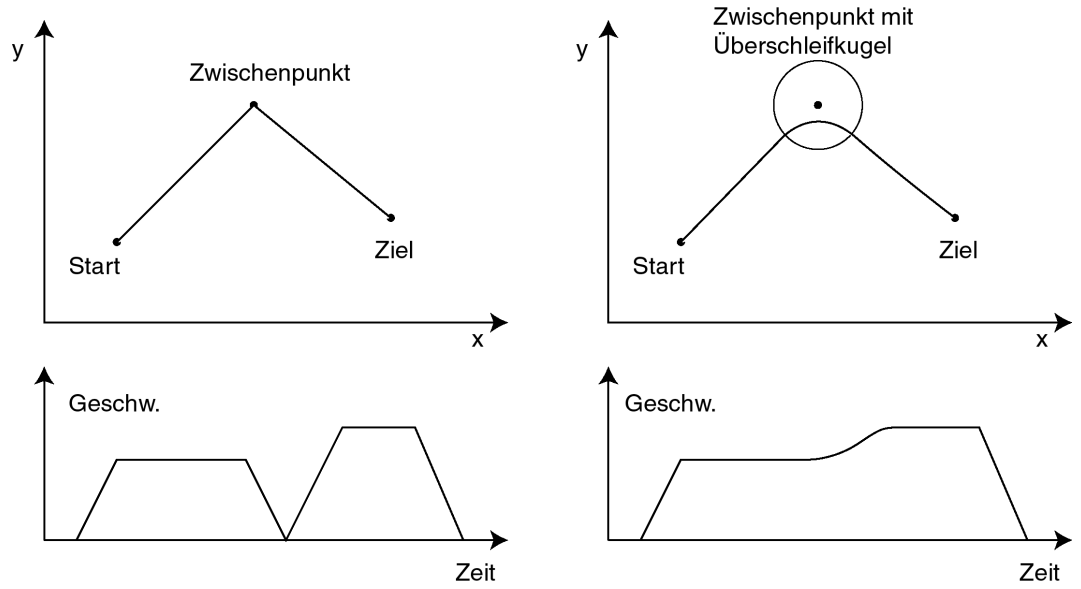


Abbildung 4.10.: Links: Bewegungsablauf mit Zwischenpunkt ohne Überschleifen, der Roboterarm kommt zum Stillstand. Rechts: Durchfahren des Zwischenpunktes mit Positionsüberschleifen, der Roboter kommt nicht zum Stillstand und führt eine flüssige Bewegung aus, der Zwischenpunkt wird aber nicht mehr exakt erreicht.

dass durch das Überschleifen die Gesamtstrecke verkürzt wird. Dies ergibt eine weitere Verkürzung der Ausführungszeit.

Wenn die Orientierung des Effektors im Zielpunkt sich von der im Startpunkt unterscheidet, erlauben viele Steuerungen, die Umorientierung schon bei Eintritt in die Überschleifkugel zu beginn (s. Abb. 4.11). Manchmal kann für die Umorientierung eine eigene Überschleifkugel festgelegt werden. Weiterhin kann ein Fly-by-Punkt genutzt werden um auf eine andere Steuerungsart umzuschalten, z.B. von CP- auf PTP-Steuerung.

Syntaxbeispiele in RAPID (ABB-Roboter)

```
MoveJ p1,vmax,z30,Zange1
```

MoveJ steht für „Move Joint“ und legt PTP-Steuerung (achseninterpoliert) für die Bewegung zu Punkt p1 der nicht erreicht wird, sondern mit einem Überschleifradius z30 flüssig passiert wird. Bei der Bewegung wird der TCP von Werkzeug Zange1 benutzt und mit einer Geschwindigkeit von vmax gefahren.

```
MoveL p2,v100,z10,Duese2
```

Mit linearer CP-Steuerung wird Punkt p2 mit einer Geschwindigkeit v100 und einem Überschleifradius z10 flüssig passiert, TCP von Werkzeug Duese2 wird benutzt.

```
MoveC KPunkt,EPunkt,v500,fine,Dorn
```

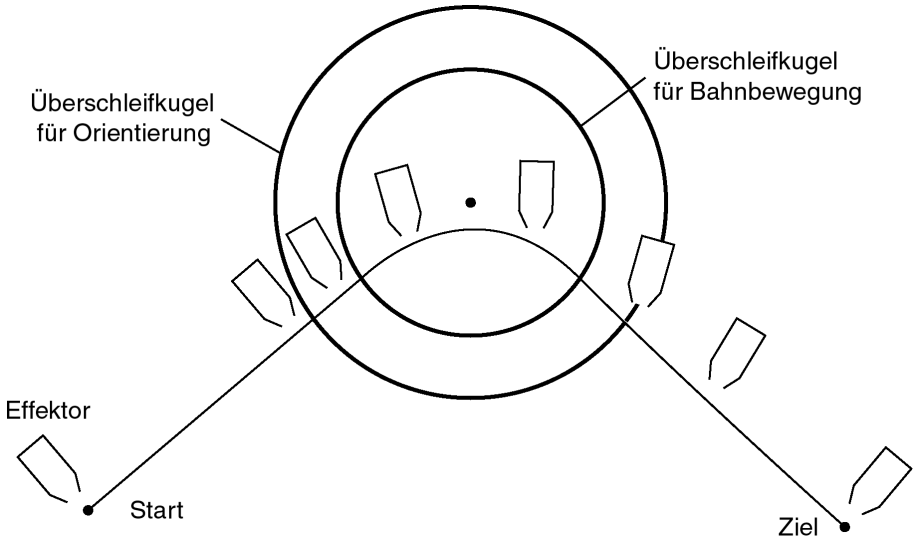


Abbildung 4.11.: Überschleifen eines Zwischenpunktes (Fly-By-Punkt): Es gibt einen Überschleifradius für die Bahnänderung (kleine Kugel) und einen zweiten für den Beginn der Umorientierung (große Kugel).

Mit zirkularer CP-Steuerung wird der Punkt EPunkt angefahren, der Zwischenpunkt KPunkt legt das Kreissegment fest. Geschwindigkeit ist v500, TCP von Werkzeug Dorn wird benutzt, der Endpunkt wird exakt angefahren (fine), kein Überschleifen, TCP kommt im Endpunkt zum Stillstand.

4.5. Synchronisation von Robotern

In vielen Anwendungen ist es notwendig, Roboter zu synchronisieren. Zum Beispiel soll ein Roboter erst mit seinem Arbeitsprogramm beginnen, wenn ein neues Werkstück auf dem Band eingefahren ist. Oder zwei Roboter stehen so nah, dass einer sich im Kollisionsbereich des anderen befindet. Dann ist es notwendig diese Roboter zu anzuhalten bzw. frei zu schalten. Das kann über GPIO-Leitungen (General Purpose Input/Output, Digitale Ein-/Ausgänge) an der Steuerung geschehen oder - heute eher üblich - durch Feldbusse. Wenn ein Feldbus eingesetzt wird, können weitere Geräte angekoppelt werden, wie z.B. ein OPC-Server. Dieser kann aus den Steuerungen (und fast allen Automatisierungsgeräten) Daten auslesen und weiter geben, an einen Datenbank-Server, ein Alarmsystem oder ein Factory-Portal.

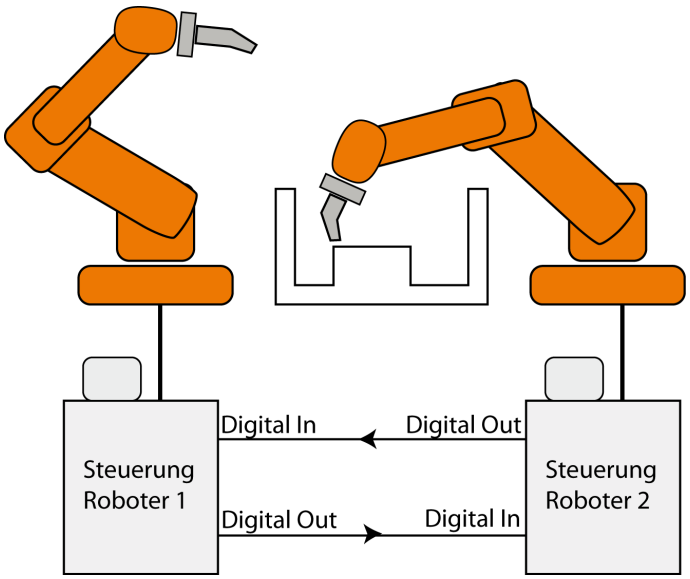


Abbildung 4.12.: Robotersteuerungen können über einfache digitale IO-leitungen kommunizieren und sich synchronisieren. Wenn einer der beiden Roboter im Kollisionsbereich des anderen arbeitet, signalisiert er das über einen Digital-Ausgang. Die andere Steuerung liest das Signal und geht in Wartezustand (HOME-Position).

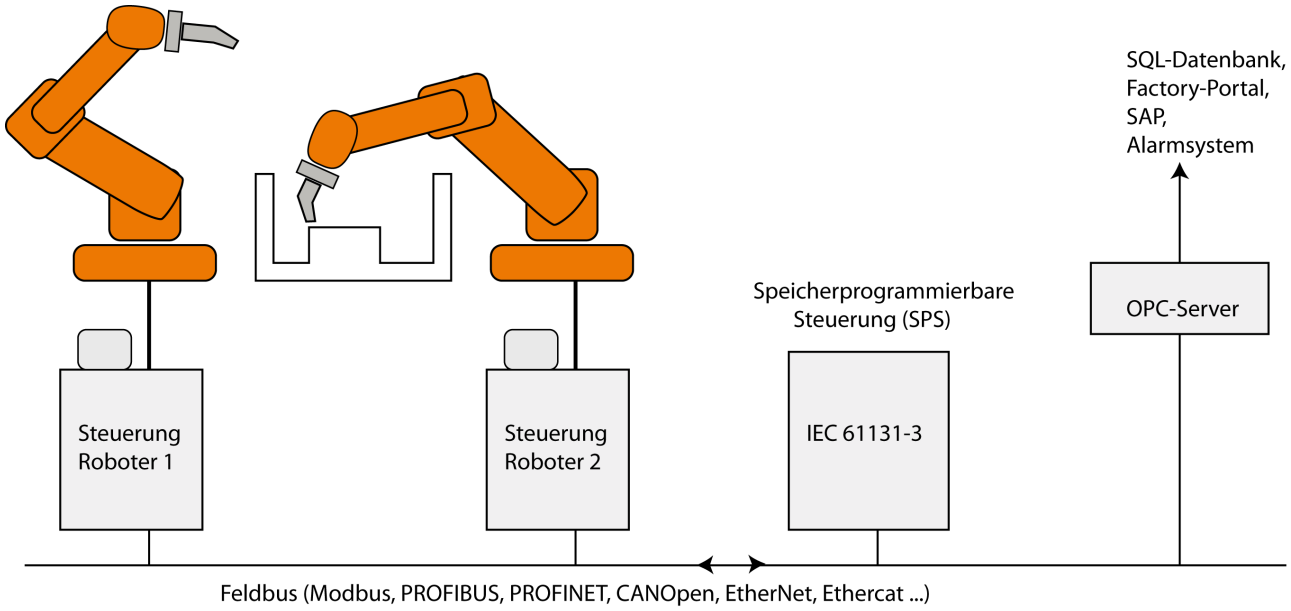


Abbildung 4.13.: Robotersteuerungen können an einen Feldbus angeschlossen werden und sich darüber synchronisieren. Häufig ist dann auch eine SPS am gleichen Feldbus angeschlossen, die mehrere Roboter steuert. Ein OPC-Server kann die Daten aus der SPS und den Robotersteuerungen auslesen.

5. Programmierung von Robotern

Die Stärke eines Roboters liegt vor allem in seiner Flexibilität. Ein Roboter kann durch Änderung seines Programmes durch den Anwender auf neue Aufgaben eingestellt werden. Es werden daher Möglichkeiten gebraucht um den Roboter einfach und sicher zu programmieren. Dazu wurden einige Methoden entwickelt.

Bei der Online-Programmierung benutzt der Programmierer den eingeschalteten Roboter und seine Steuerung als Hilfsmittel der Programmierung. Die wichtigsten Methoden sind:

- Teach-In-Programmierung
- Folgeprogrammierung
- Master-Slave-Programmierung

Bei der Offline-Programmierung dagegen arbeitet der Programmierer zunächst ohne den Roboter und kopiert das Programm erst später in die Robotersteuerung. Die Offline-Programmierung beruht auf der Verwendung von Robotersprachen.

5.1. Online-Programmierung

5.1.1. Teach-In-Programmierung

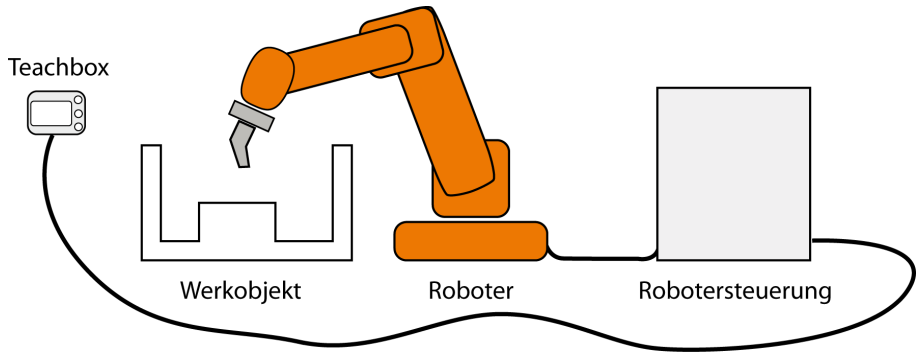


Abbildung 5.1.: Zwischen Teachbox und Robotersteuerung ist ein langes Kabel damit man sich beim einteachen zum Werkobjekt bewegen kann.

Bei der Teach-In-Programmierung benutzt der Programmierer ein spezielles Eingabegerät, die *Teach-Box* (auch *Teach-Panel, Handprogrammiergerät* oder Lehrgerät). Die Teach-Box ist Teil der Robotersteuerung mit vielen Tasten und Hebeln, mit der alle Funktionen des Roboters in der Art einer Fernbedienung direkt gesteuert werden. Oft ist auch eine so genannte *Teach-Kugel 6D-Maus* vorhanden, an der manuell durch Kräfte und Drehmomente die 6 Achsen gesteuert werden können (gewöhnungsbedürftig). Der Anwender führt nun mit der Teach-Box die zu programmierende Bewegungsfolge aus. Dabei wird in der Regel mit stark verminderter Geschwindigkeit gearbeitet. Für die Teach-Bewegungen werden verschiedene Alternativen angeboten, z. B. :

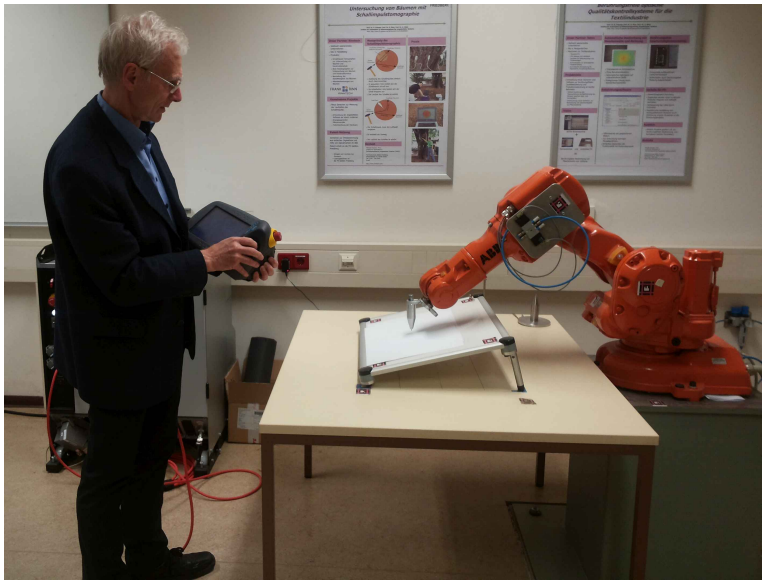


Abbildung 5.2.: Teach-In am IRB140.

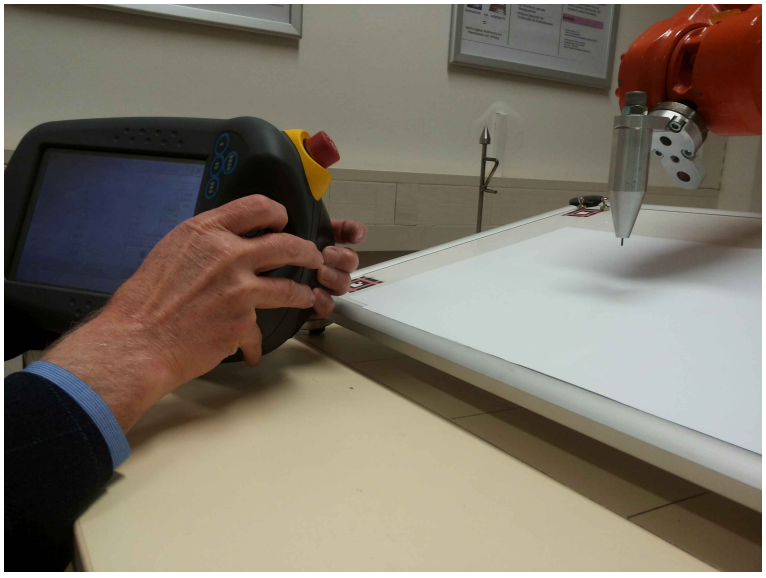


Abbildung 5.3.: Beim einteachen steht man oft nah am Roboter.

Abbildung 5.4.: Der Roboter wird behutsam mit dem Joystick bewegt.

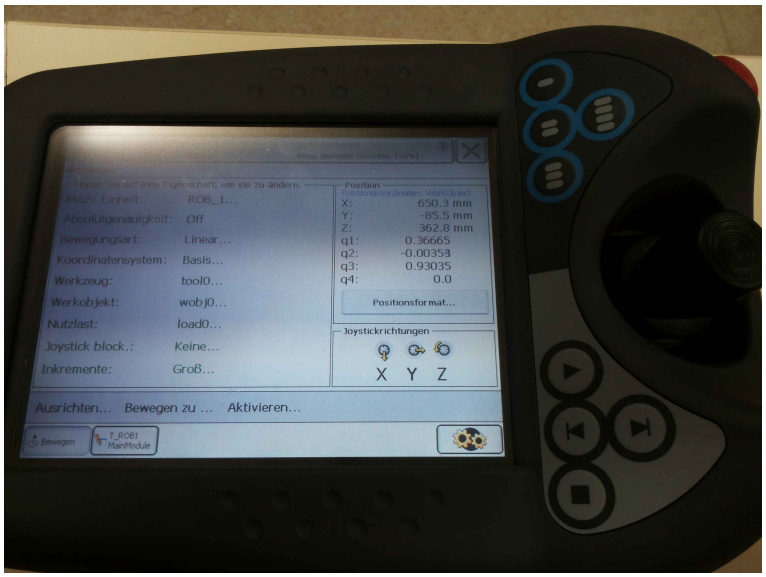


Abbildung 5.5.: Die Teachbox infomiert über die Bewegungsparameter des Roboters an.

- Verfahren nach Gelenkkoordinaten,
- Reine Translation, Tool behält Orientierung bei
- Reines Umorientieren, TCP bleibt am gleichen Ort.

Wichtig ist nun, dass eine Reihe von Zwischenstellungen durch einen speziellen Tastendruck abgespeichert wird (*Touch-Up*). Diese Zwischenstellungen werden nummeriert und es wird zu jedem Punkt vermerkt, mit welchen Bewegungsparametern dieser Punkt später angefahren werden soll. Es wird also für jeden Zwischenpunkt ein Datensatz hinterlegt, der z.B. enthält:

- Alle Gelenkkoordinaten oder Position und Orientierung des Effektors,

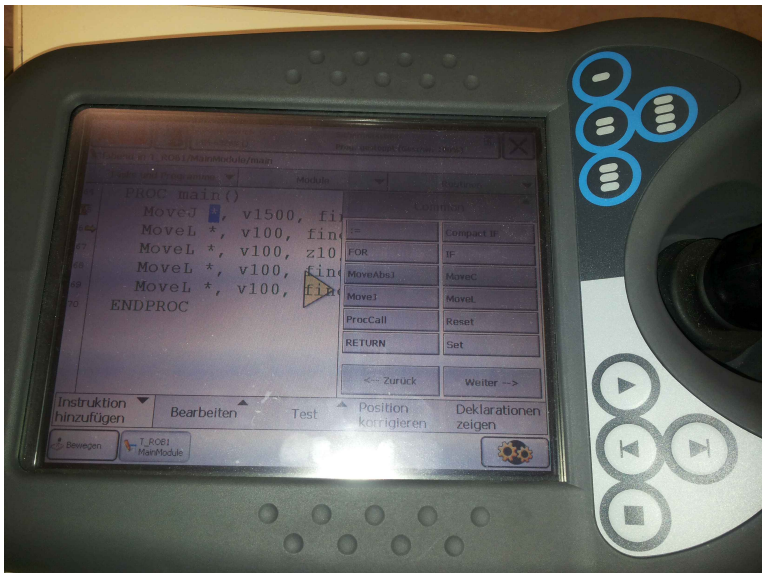


Abbildung 5.6.: Die Steuerung bietet beim einteachen den Befehlssatz des Roboters zum einfügen an.

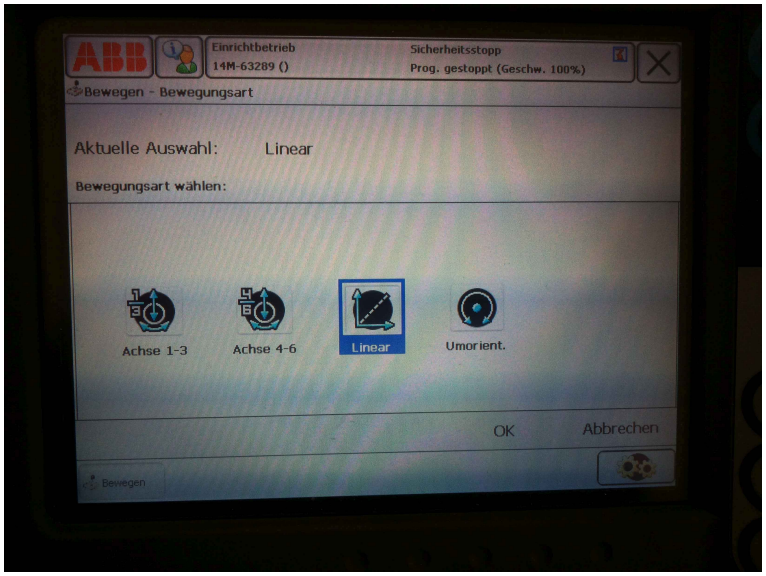


Abbildung 5.7.: Die Steuerung bietet die Bewegungen achsenweise Bewegung, Linear CP oder Umorientieren an.

- Greiferzustand,
- Bewegungssteuerung: PTP, lineare CP-Steuerung oder zirkulare CP-Steuerung,
- Anhalten oder Überschleifen, Überschleifradius
- Geschwindigkeit,
- Beschleunigung

Das „eingeteachte“ Programm kann später noch an Bildschirm oder Teach-Box nachbearbeitet werden, um z.B. Wiederholungen einzufügen. Für die Anwen-

dung wird es dann im so genannten *Automatikbetrieb* abgespielt: Der Roboter fährt nun der Reihe nach alle eingespeicherten Zwischenstellungen ab und arbeitet so die beabsichtigte Bewegungsfolge flüssig ab.

5.1.2. Force-Control-Teach-In, Master-Slave-Programmierung

Beim Force-Control-Teach-In führt der Bediener direkt den Effektor des Roboters, dessen Steuerung gleichzeitig eine Bahnaufzeichnung macht. Dazu wird am Tool ein *Kraft-Momenten-Sensor* montiert. Der Roboter wertet diesen Sensor aus und folgt der durch den Bediener gewünschten Bewegung. Typische Anwendung: Lackieren. Kritisch: Sicherheitsaspekt.

Bei der Master-Slave-Programmierung gibt es ein kleines Modell des Roboters (Masterarm, Bedienarm), an dem der Programmierer die Bewegung ausführt. Diese Bewegungen werden an die Steuerung übertragen und vom eigentlichen Roboter (Slave-Arm, Arbeitsarm) direkt ausgeführt. Die Bewegungen können bzw. müssen dabei in einem festen Maßstab skaliert werden.

5.1.3. Praktische Aspekte bei der Programmierung

5.1.3.1. Vorpunkte (Annäherungspunkte, Abrückpunkte)

Man führt das Werkzeug üblicherweise nicht in einer schnellen Bewegung bis zum Werkstück, sondern zunächst nur bis zu einem *Annäherungspunkt*. Dieser liegt in geringem Abstand senkrecht über dem Werkstück. Von dort

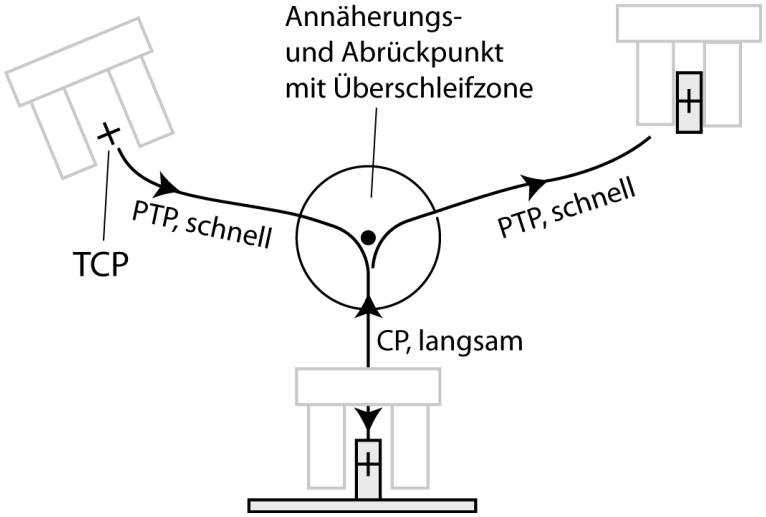


Abbildung 5.8.: Typisches Schema für einen Greifvorgang. Ein Punkt wird sowohl beim Annähern wie auch beim Entfernen genutzt. (Annäherungs- und Abrückpunkt) Dort findet ein Wechsel der Geschwindigkeit und der Bewegungsart statt.

nähert sich das Werkzeug langsam und kontrolliert dem Werkstück. Nach Beendigung des Bearbeitens oder Greifens entfernt sich das Werkzeug langsam und CP-gesteuert zunächst bis zum *Abrückpunkt* und geht erst dort wieder in eine schnellere PTP-Bewegung über. Diese Punkte heißen auch *Vorpunkte*.

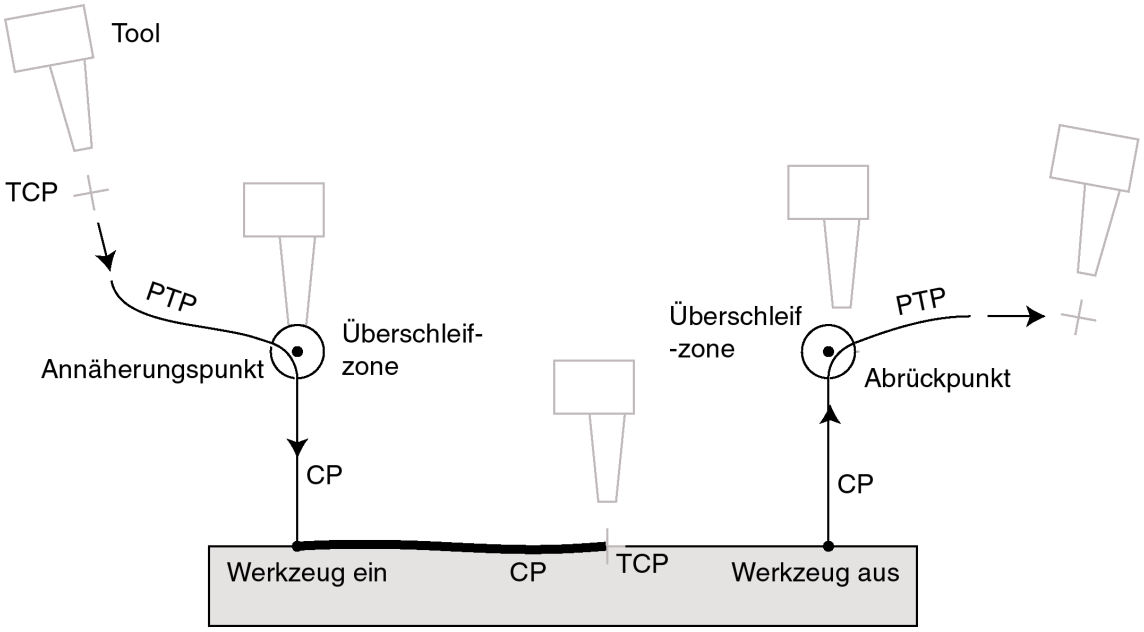


Abbildung 5.9.: Typisches Schema für einen Bearbeitungsgang an einem Werkstück. Im Annäherungs- und im Abrückpunkt findet ein Wechsel der Geschwindigkeit und der Bewegunsgart statt.

Für eine Bearbeitung eines Werkstückes (z. B. Schweißen oder Wasserstrahlschneiden) ist ein bewährtes Grundschema:

- Bewegung des Werkzeuges im freien Arbeitsraum: PTP-Steuerung bis zum Annäherungspunkt
- Im Annäherungspunkt besteht schon die richtige Orientierung des Werkzeugs
- Um den Annäherungspunkt kann eine kleine Überschleifzone liegen
- Der Annäherungspunkt wird ohne Stopp durchfahren, dabei Wechsel auf niedrigere Geschwindigkeit und auf Steuerungsart CP
- Annäherung an das Werkstück auf einer Bahn senkrecht zur Oberfläche, langsam mit CP linear
- Beginn der Bearbeitung, Bahn mit kontrollierter Geschwindigkeit CP linear oder zirkular
- Beendigung der Bearbeitung
- Entfernung vom Werkstück auf senkrechter Bahn langsam mit CP-Steuerung bis zum Abrückpunkt

- Der Abrückpunkt wird ohne Stopp durchfahren, dabei Wechsel auf höhere Geschwindigkeit und Steuerungsart PTP
- Weitere Entfernung des Werkzeuges im freien Arbeitsraum

Die nach diesem Schema entstehende Bewegung ist in Abb.5.9 dargestellt. Ähnlich wird man beim Annähern an einen zu greifenden Gegenstand vorgehen.

5.1.3.2. Genügend Teachpunkte

Bei der Teach-In-Programmierung muss genau darauf geachtet werden, welche Zwischenpunkte eingespeichert werden, und welche nicht. Der Roboter fährt alle Achsen direkt und gleichzeitig auf die eingespeicherten Werte. Nicht gespeicherte Zwischenpunkte werden nicht berücksichtigt. In Abb. 5.10 wurde beim Teach-In auf das Hindernis Rücksicht genommen, der entsprechende Punkt an der Ecke links oben aber nicht eingespeichert. Der Roboter steuert daher seine Achsen direkt auf die Achsenkoordinaten des Zielpunktes und es kommt zur Kollision.

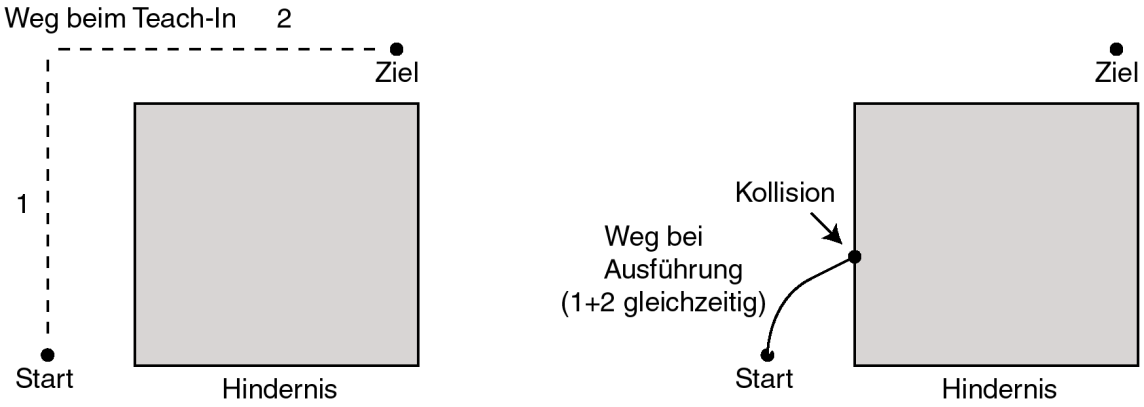


Abbildung 5.10.: Der angefahrene Zwischenpunkt links oben wurde nicht eingespeichert, der Roboter steuert den Zielpunkt direkt an.

Nach Einspeicherung des Zwischenpunktes ist das Problem beseitigt und das Hindernis wird umfahren. Im Falle einer Punkt-zu-Punkt-Steuerung verbleiben allerdings immer noch gewisse Bahnabweichungen. Um eine Kollision sicher zu vermeiden, muss der Zwischenpunkt weit genug vom Hinderniss entfernt sein.(Abb. 5.11).

5.1.3.3. Größe der Überschleifzone

Wenn beim Teach-In die Überschleifzone eines Punktes zu groß gewählt wird, kann es beim Abfahren der Bahn zur Kollision kommen, wenn ein Hindernis im Weg ist (Abb. 5.12). Das Durchfahren des Fly-By-Punktes ist sicher, wenn die Überschleifzone keine Gegenstände überlappt. (Abb. 5.13)

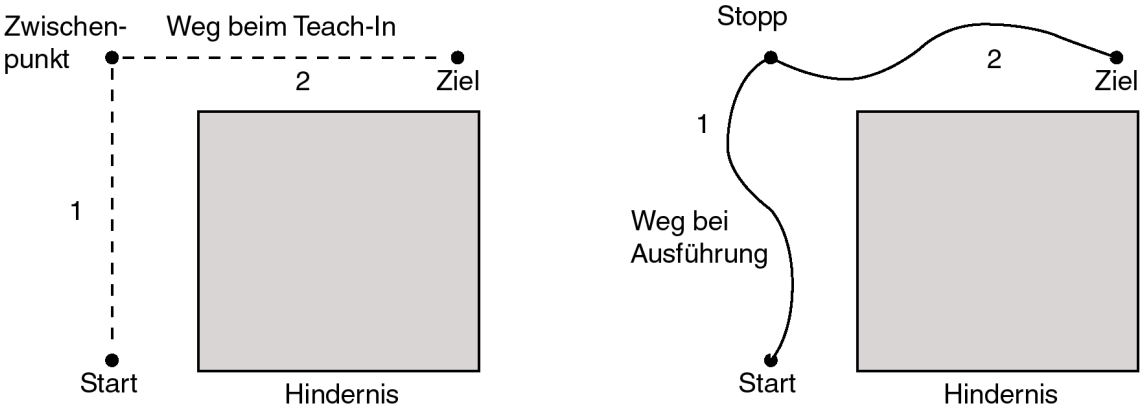


Abbildung 5.11.: Der angefahrene Zwischenpunkt links wird eingespeichert, der Roboter umfährt nun wie gewünscht das Hindernis.

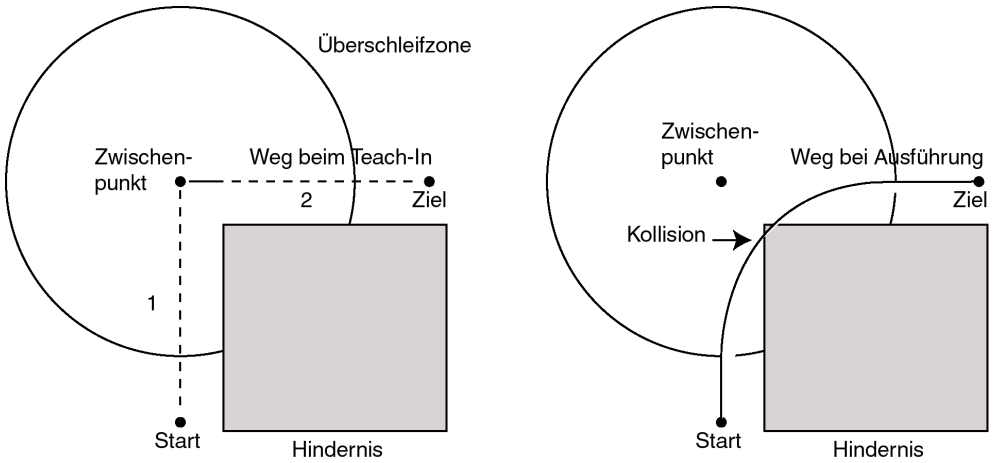


Abbildung 5.12.: Eine zu große Überschleifzone kann zur Kollision führen.

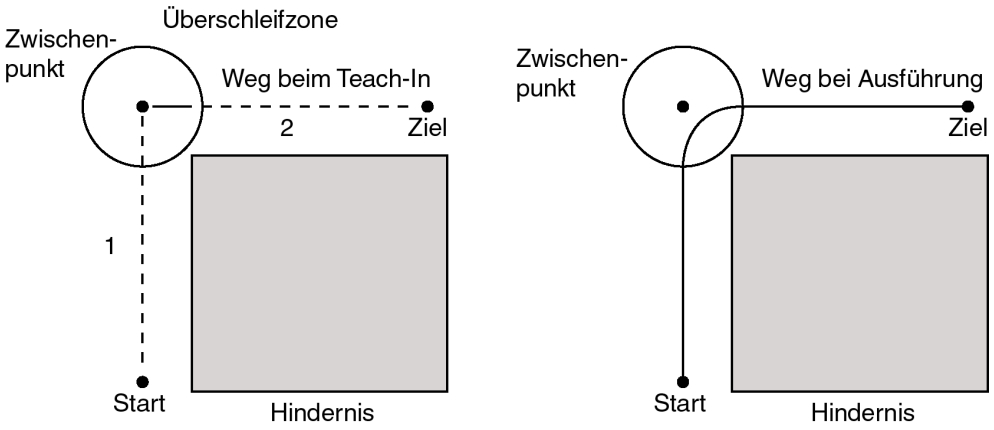
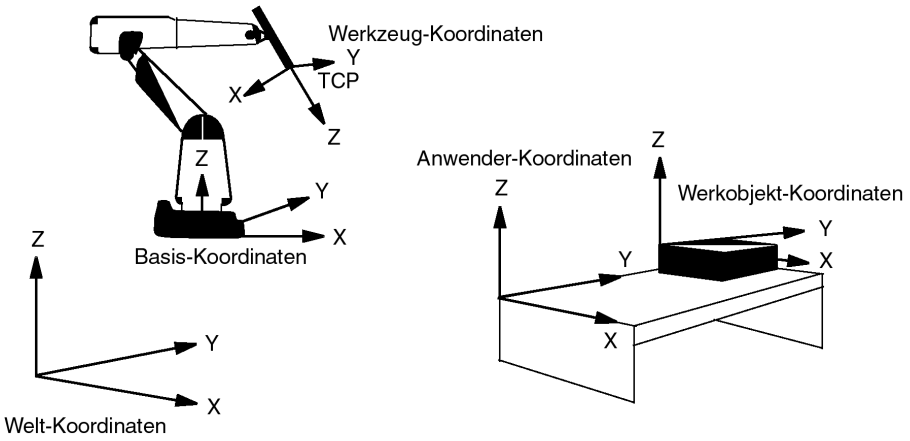


Abbildung 5.13.: Eine Überschleifzone, die nicht mit Gegenständen überlappt, kann nicht zur Kollision führen.

5.1.4. Vor- und Nachteile der Online-Programmierung

Vorteile: Programmierung ist anschaulich, findet direkt in Arbeitsumgebung statt, Kollisionen, Ungenauigkeiten und andere Störungen werden unmittel-

bar entdeckt. Arbeitsraum und Objekt müssen nicht vermessen werden. Nur geringe Programmierkenntnisse sind notwendig, Erfahrungsschatz der Anwender kann bei Programmierung einfließen.
Nachteile: Erstellung komplexer Programme schwierig, Sensorintegration schwierig, Roboterarbeitsplatz wird zur Programmierung belegt.



Die verschiedenen Koordinatensysteme des Manipulators (wenn der Manipulators das Werkzeug hält) Abbildung: ABB Robotics.

Abbildung 5.14.: Die Steuerungen von Industrierobotern unterstützen die Verwendung verschiedener Bezugssysteme. (Quelle: ABB) Daraus ergeben sich viele praktische Vorteile.

5.2. Die praktische Verwendung von Bezugssystemen

In der Roboterprogrammierung ist es oft sehr nützlich Bezugssysteme (Koordinatensysteme) für Effektoren (Werkzeuge), Werkstücke, Werkstückaufnahmen und Gegenstände der Umwelt zu vergeben. Die Verwendung mehrerer Bezugssysteme wird von den Steuerungen der üblichen Industrieroboter unterstützt. (s. Abb.5.14) Die Vorteile sind:

- Daten, die relativ zu einem Bezugssystem definiert sind, ändern sich nicht, wenn das Bezugssystem als Ganzes seine Lage ändert (interne Lage bleibt unverändert).
- Beim Teach-In kann entlang der Achsen des Bezugssytems verfahren werden. Wenn die Achsen des Bezugssystems parallel zu ausgezeichneten Richtungen liegen erleichtert das ein Teach-In.
- Die Offline-Programmierung wird erleichtert, da entlang der Achsen des Bezugssystemes einfache Koordinatentransformationen entstehen.
- rechnerische Erzeugung von Punkten erleichert
- Bezugssysteme können von mehreren Robotern verwendet werden.

5.2.1. Einteachen eines Koordinatensystems

Ein Koordinatensystem kann eingeteacht werden, indem mit einer Werkzeugspitze drei Punkte eingeteacht werden, beispielsweise x_1, x_2 und y_1. Dann wird aus x_1 und x_2 die x-Achse ermittelt. Anschließend wird eine Senkrechte zur x-Achse ermittelt, die durch den Punkt y_1 geht. Der Schnittpunkt dieser Senkrechten mit der x-Achse ist der Ursprung. Die z-Achse wird senkrecht zu den beiden anderen Achsen durch den Ursprung gelegt, so dass ein Rechtssystem entsteht.

5.2.2. Einteachen eines Werkzeugarbeitspunktes (TCP)

Bei einem neuen Werkzeug muss ein Werkzeugarbeitspunkt (TCP) eingemessen werden. Dies kann mit einer Spitze gemacht werden, an die das Werkzeug mehrmals aus verschiedenen Richtungen vorsichtig herangefahren wird. Wird auch eine z-Achse für das Werkzeug gebraucht, kann beispielsweise ein Zusatzpunkt angefahren werden. Die Verbindungslinie vom TCP zum Zusatzpunkt ist dann die z-Achse des Werkzeugs.

5.2.3. Anwendungsbeispiele zu Bezugssystemen

Definition eines Werkzeugkoordinatensystems Alle Bewegungsbefehle beziehen sich mit Bahnangabe, Geschwindigkeit und Beschleunigung auf den TCP (Werkzeugarbeitspunkt). Bewegungen beim Teach-In können in Richtung der Werkzeugachsen geschehen, auch bei abgeknickten Werkzeugen. Nach einem Werkzeugwechsel wird nur das neue Werkzeug eingemessen, das Programm wird nicht geändert.

Definition eines Werkstückkoordinatensystems Die Verwendung ist vorteilhaft, wenn sie sich die Bewegungen am Werkstück ausrichten. Das Programm bleibt unverändert, wenn die Lage des Werkstückes sich ändert.

Beispiel: Herausziehen eines Zylinders aus einer Bohrung. Zunächst legt man ein Werkstückkoordinatensystem so, dass die Längsachse der Bohrung in Richtung der z-Achse des Bezugssystems liegt. Dann gibt es mehrere praktische Möglichkeiten:

- Man teacht die Greifposition ein, schließt den Greifer, schaltet auf Werkstückkoordinaten, verfährt in z-Richtung bis der Zylinder frei ist, teacht diese Position wieder ein und wählt Bewegungsart CP-linear zwischen dem ersten und dem zweiten Punkt.
- Man richtet das Werkzeug nach den Achsen des Werkstückes aus und bewegt sich entlang der z-Achse des Werkzeuges.

- Man teacht die Greifposition ein, erzeugt dann rechnerisch einen zweiten Punkt, der in z-Richtung gegen den ersten Punkt verschoben ist, wobei man sich auf Werkstückkoordinaten bezieht.

Als Bewegungsart zwischen dem ersten und dem zweiten Punkt wählt man CP-linear.

Definition eines Koordinatensystems an der Werkstückaufnahme Dies ist ein Zwischensystem, auf das sich die Werkstückkoordinatensysteme beziehen. Wenn die Position der Werkstückaufnahme verändert wird, braucht nur deren Koordinatensystem neu eingeteacht werden, alle Werkstücke sind weiterhin relativ dazu definiert.

Bewegte Roboter Wenn ein Roboter auf einer Verfahrachse oder in einem Portal bewegt wird, ist die Transformation zwischen Basis-KS (Roboterfuß) und Weltkoordinatensystem nicht mehr fix sondern variabel. Alle anderen Koordinatensysteme sind aber fix relativ zum Weltkoordinatensystem. Wenn die Lage des Roboterfußes relativ zum Welt-KS bekannt ist, sind alle Koordinaten, die sich auf das Welt-KS beziehen, korrekt.

5.3. Offline-Programmierung

5.3.1. Allgemeines über Robotersprachen

5.3.1.1. Die Entstehung von Robotersprachen

Für die Entwicklung von Roboterprogrammen, die Sensorsignale verwenden oder die CAD-Daten enthalten, bietet sich die Offline-Programmierung an. Sie ergibt sich schon bei der Nachbearbeitung von Daten, die durch Online-Programmierung gewonnen wurden. Es gibt mehrere Möglichkeiten dazu:

- Textuelle Programmierung, Robotersprachen, ähnl. Programmiersprachen.
- Grafische Programmierung.
- Programmierung in Simulationssystemen.

Wir betrachten hier nur die textuelle Programmierung. Für die Entwicklung sind drei Wege möglich:

1. Vollständiger Neuentwurf der Sprache unter besonderer Berücksichtigung der Robotik. Bsp.: AL, VAL,
2. Ergänzung einer vorhandenen Programmiersprache, Bsp.: AUTOPASS, PASRO,

3. Erweiterung einer Automatisierungs- oder Steuersprache, Bsp.: RAPT.

Der Versuch einer Normung der Robotersprachen ist bis zum heutigen Tag gescheitert (auch IRL), so dass herstellerspezifische Sprachen die Praxis dominieren (ABB: RAPID, Unimation/Stäubli: VAL, V+, KUKA: KRL).

5.3.1.2. Grundelemente von Robotersprachen

Alle Robotersprachen stellen einen bestimmten Vorrat an Grundbefehlen zur Verfügung, mit denen sich die üblichen Aufgabenstellungen bewältigen lassen. Dazu gehören:

- Vorgabe des Zieles in Position und Orientierung, Vorgabe der Bewegungssteuerung: PTP-, Linear- oder Zirkulartsteuerung, Geschwindigkeit und Beschleunigung.
- Ansteuerung des Effektors, und evtl. weiterer Achsen.
- Ein- und Ausgabe von Sensorsignalen für Mess- und Synchronisierungsaufgaben.
- Kontrollstrukturen für Wiederholungen und Verzweigungen.
- Zuweisungen auf Variable, Auswertung von Ausdrücken, arithmetische und logische Operationen, Unterprogramme.

Beispiel In gebogenes Blech mit einem Radius von 40 mm soll durch einen Roboter mit einem Wasserstrahlschneidegerät ein Ausschnitt wie abgebildet gesägt werden. Der Abstand des Effektors zum Blech soll konstant 10 mm betragen, die Schneidegeschwindigkeit 10 mm/s, der TCP liegt im Arbeitspunkt (Schneidepunkt). Der folgende Programmvorschlag ist in Pseudo-Robotersprache

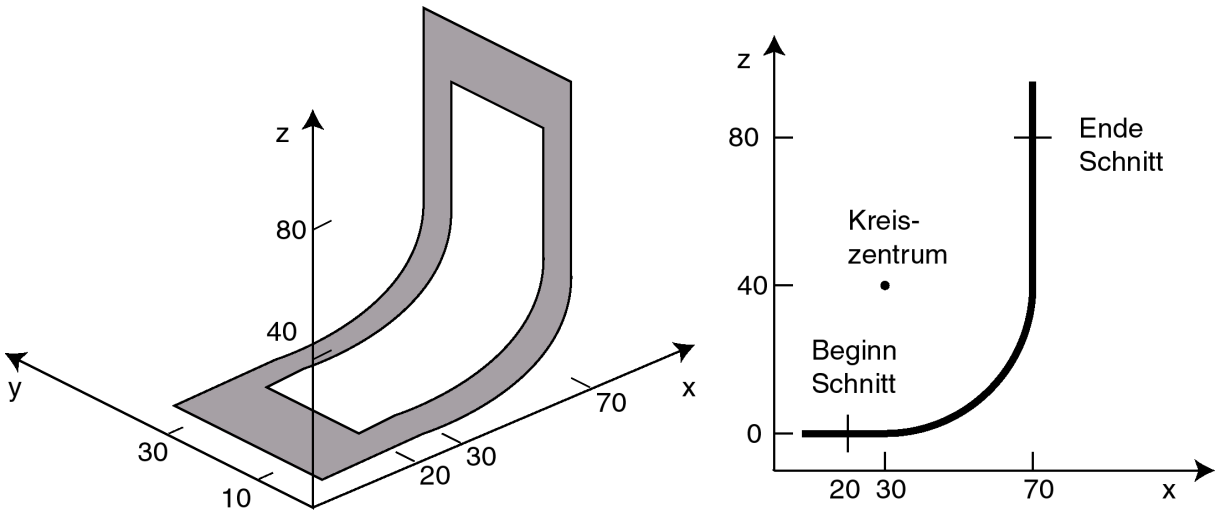


Abbildung 5.15.: In ein Blech wird mit einem Wasserstrahl ein Ausschnitt gesägt. Links: 3D-Skizze, rechts: Seitenansicht.

abgefasst. Der Gehezu-Befehl enthält die Parameter $x, y, z, \phi_x, \phi_y, \phi_z$. Die Zwischenpunkte für die Kreisbögen wurden in die Mitte des Bogens gelegt.

```
Gehezu(20, 10, 0, 0, 0, 0, PTP, UEB=NEIN)
Tool EIN
Gehezu(30,10,0,0,0, 0, LIN, UEB=NEIN, Geschw=10)
Gehezu(70,10,40,0,-90,0,ZIRK (Zwischenpunkt=58.28,10,11.72),UEB=NEIN,Geschw=10)
Gehezu(70,10,80,0,-90,0,LIN, UEB=NEIN, Geschw=10)
Gehezu(70,30,80,0,-90,0,LIN, UEB=NEIN, Geschw=10)
Gehezu(70,30,40,0,-90,0,LIN, UEB=NEIN, Geschw=10)
Gehezu(30,30,0,0,0,0,ZIRK (Zwischenpunkt=58.28,30,11.72),UEB=NEIN,Geschw=10)
Gehezu(20,30,0,0,0,0,LIN, UEB=NEIN, Geschw=10)
Gehezu(20,10,0,0,0,0,LIN, UEB=NEIN, Geschw=10)
Tool AUS
Home, PTP
```

Evtl. ist es möglich die Punkte mit Bahnüberschleifen flüssig zu durchfahren, das müsste getestet werden.

5.3.1.3. Weitergehende Konzepte von Robotersprachen

Mit den Grundelementen der Roboterpsrachen lassen sich zwar alle Programmieraufgaben lösen, manchmal entstehen jedoch etwas schwer lesbare Programme und die weitere Bearbeitung des Programmes wird zunehmend mühselig. Stellen wir uns zum Beispiel vor, in obigem Beispiel (Abb. 5.15) gäbe es ein zweites gleiches Blech, in das ein gleicher Ausschnitt gesägt werden müsste. Das zweite Blech sei gegenüber dem ersten um den Vektor $(0.75, 3.19, -5.02)$ verschoben und verdreht um 27^o um die x-Achse und um -143^o um die y-Achse. Es ist nun sehr mühsam, die entsprechende zweite Programmliste aufzustellen. Es gibt deshalb Programmiersprachen, die weitergehende Konzepte zur komfortablen und übersichtlichen Programmierung anbieten [8]. Ein Vorreiter war in dieser Hinsicht die Sprache AL, die fast alle nützlichen Konzepte verwirklichte. Leider wurde AL nie in der Praxis eingesetzt, seine Konzepte waren aber Vorbild für viele andere Roboterprachen. Im Folgenden sind einige wichtige Konzepte fortschrittlicher Robotersprachen kurz beschrieben.

Beschreibung der Roboterwelt in kartesischen Koordinaten Der Programmierer muss keine detaillierten Kenntnisse der Kinematik dieses speziellen Roboters haben, die Programme werden leicht lesbar und können leichter auf andere Roboter übertragen werden.

Framekonzept und Umweltmodell Unter Frame wird hier ein Koordinatensystem verstanden, das man sich mit einem Gegenstand fest verbunden denkt. So kann jeder Gegenstand im Arbeitsraum hinsichtlich Position und Orientierung durch seinen Frame beschrieben werden. Die relative Lage der Gegenstände zueinander wird durch Transformationen zwischen den Frames

beschrieben. Wenn Gegenstände miteinander verbunden sind, ist die Transformation zwischen ihnen konstant.

Ein Umweltmodell bietet weitere Beschreibungsmerkmale. Manche Konzepte sehen vor, dass durch eine Anweisung in der Robotersprache die Frames der Gegenstände verbunden werden (AL: AFFIX). Eine entgegengesetzt wirkende Anweisung löst die Verbindung wieder (AL.UNFIX). Der Gesamtzustand kann in einem *Framereferenzgraphen* dargestellt werden. Es kann auch unterschieden werden, ob die Verbindung zwischen den Gegenständen beidseitig fest (AL: rigidly) oder nur einseitig ist (AL: non-rigidly).

Spezielle Datentypen Neben den gängigen Datentypen für ganze Zahlen und Fließkommazahlen gibt es Datentypen für

- Vektoren
- Homogene Matrizen
- Transformationen
- Rotationen
- Translationen
- Frames

Da diese Datentypen im Zusammmenhang mit Bewegungsbefehlen eingesetzt werden können, lassen sich diese sehr viel eleganter formulieren.

Spezielle Operatoren Für die speziellen Datentypen werden sinnvollerweise auch spezielle Operatoren angeboten, mit denen man z.B. eine Vektoraddition oder eine Matrizenmultiplikation ausführen kann. Beispiele:

```
v2 = v1 + VEKT(10,20,0) //Vektor v2 = Vektor v1 + (10,20,0)
F = F * TRANS(0,0,20);  //Translation des Frames F um Vektor (0,0,20)
F = F * ROT(90,0,0)     //Rotation des Frames F um 90 Grad um x-Achse
```

Konstruktoren und Selektoren Um die genannten strukturierten Datentypen zu konstruieren und mit Werten zu belegen müssen Konstruktoren angeboten werden. So könnte z. B. ein Vektor aus drei Skalaren aufgebaut werden. Eine homogene Matrix wiederum aus drei Vektoren.

Eine Rotation kann aus drei raumfesten Drehwinkeln (ϕ_x, ϕ_y, ϕ_z) konstruiert werden. Ebenso könnte ein *Konstruktor* eine Rotationsmatrix aus drei Euler-Winkeln erzeugen, oder aus einem Vektor (Drehachse) und einem Drehwinkel. Aus einer Rotationsmatrix und einem Translationsvektor kann man dann eine Transformation erzeugen, die beides beinhaltet.

Umgekehrt wird es vielleicht einmal notwendig, aus einer Transformationsmatrix den Rotationsanteil herauszuziehen, dafür wird ein *Selektor* bereitgestellt.

Ein anderer Selektor liefert den Translationsanteil der Transformationsmatrix. Letzlich stehen dann allen Konstruktoren entsprechende Selektoren gegenüber.

Anrück- und Abrückpunkte In vielen Fällen ist es notwendig, bei der Annäherung an ein Ziel einen ganz bestimmten Weg exakt einzuhalten. Beispiele dafür sind das Einführen eines Bolzens in eine Bohrung und das Absenken des Greifers über ein Objekt. Moderne Konzepte stellen Sprachkontrukte zur Verfügung, die solche Anrück- und Abrückpunkte selbständig berechnen. das kann z.B. aus der Orientierung der Hand und einer Entfernungsangabe oder einem Vektor geschehen (AL: APPROACH und DEPART).

Konfigurationsparameter Wenn ein Roboter eine Stellung mit verschiedenen Sätzen von Gelenkwinkeln einnehmen kann, muss der Programmierer entscheiden, welcher davon benutzt werden soll. Beispiele:

- RIGHTY oder LEFTY
- ABOVE oder BELOW
- FLIP oder NOFLIP

Kommen wir nun noch kurz auf obiges Beispiel zurück. Man könnte jeweils an die Endpunkte der Bahnsegmente einen Frame legen, der Position und Orientierung des Effektors beschreibt (Abb. 5.16). Alle Abstände, Punkte und

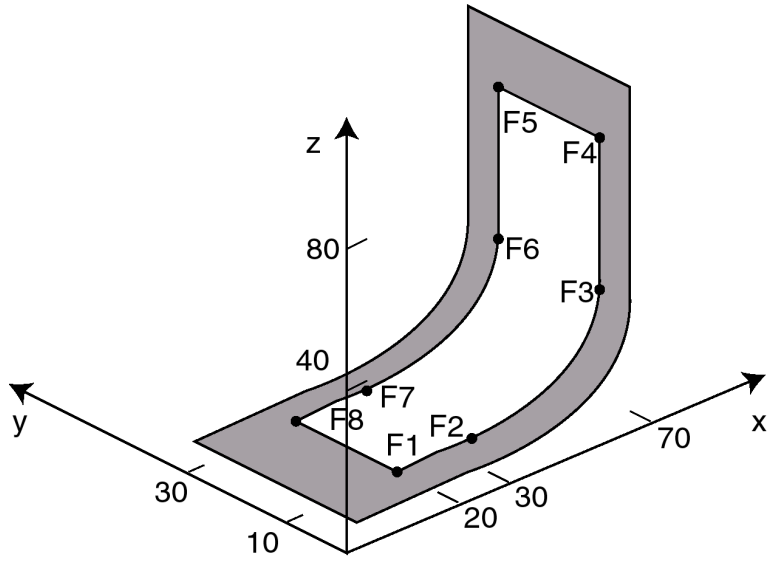


Abbildung 5.16.: Für die gleiche Schneideaufgabe werden Frames (F1 bis F8) in die Endpunkte der Bansegmente gelegt.

Zwischenpunkte werden zunächst auf Variablen abgelegt. Aus diesen Variablen werden dann die Frames (Position und Orientierung) erzeugt, die vom TCP angefahren werden. Die Zwischenpunkte sind Vektoren und werden in die Mitte des Kreisbogens gelegt.

```
int Schneideabstand=0;      // da TCP im Schneidepunkt
VEKT Schnittanfang=(20,10,0);
int Unterelaenge = 10;
int Oberelaenge = 40;
int Breite = 20;
int Radius = 40;
VEKT Kreiszentrum1=(30,10,40);
VEKT Zwischenpunkt1 = Kreiszentrum1 + Radius*(SQRT(0.5), 0 , -SQRT(0.5));
VEKT Zwischenpunkt2 = Zwischenpunkt1 + (0, Breite, 0);
F1 = TRANS(Schnittanfang) * TRANS(0, 0, Schneideabstand) * ROT(0, 0, 0);
```

Alle folgenden Frames werden nun aus dem ersten Frame erzeugt:

```
F2 = F1 * TRANS(Unterelaenge, 0, 0);
F3 = F2 * TRANS(Radius, 0, Radius) * ROT(0,-90,0)
F4 = F3 * TRANS(0,0,Oberelaenge);
```

Die Frames $F5 \ldots F8$ liegen nun parallel zu den Frames $F1 \ldots F4$ und können einfach durch eine Translation aus den ersten vier erzeugt werden:

```
F5 = F4 * TRANS(0, Breite, 0);
F6 = F3 * TRANS(0, Breite, 0);
F7 = F2 * TRANS(0, Breite, 0);
F8 = F1 * TRANS(0, Breite, 0);
```

Nun sind alle Endpunkte von Bahnsegmenten durch Frames beschrieben und können durch Bewegungsbefehle angefahren werden.

```
GEHEZU(F1, PTP, UEB=NEIN)
GEHEZU(F2, LIN, UEB=NEIN, Geschw=10)
GEHEZU(F3, ZIRK (Zwischenpunkt1), UEB=NEIN, Geschw=10)
GEHEZU(F4, LIN, UEB=NEIN, Geschw=10)
GEHEZU(F5, LIN, UEB=NEIN, Geschw=10)
GEHEZU(F6, LIN, UEB=NEIN, Geschw=10)
GEHEZU(F7, ZIRK (Zwischenpunkt2), UEB=NEIN, Geschw=10)
GEHEZU(F8, LIN, UEB=NEIN, Geschw=10)
GEHEZU(F1, LIN, UEB=NEIN, Geschw=10)
```

Um nun an einem zweiten Blech den gleichen Ausschnitt zu erzeugen, müsste man nur alle acht Frames transformieren:

```
T = ROT(27, -143, 0) * TRANS(0.75, 3.19, -5.02);
F1 = F1 * T
F2 = F2 * T
```

```
F3 = F3 * T
F4 = F4 * T
F5 = F5 * T
F6 = F6 * T
F7 = F7 * T
F8 = F8 * T
```

Wenn die Frames in einem Array organisiert sind, lässt sich das sogar in einer kleinen Schleife durchführen. Anschließend müssen wieder die gleichen Bewegungsbefehle aufgerufen werden, wie oben, praktischerweise in Form eines Unterprogrammes. Man sieht, dass die Lösung nun sehr einfach und flexibel geworden ist (man stelle sich testweise vor, einer der Parameter, z.B. die Breite oder der Radius würden sich ändern: Die Veränderungen im Programm wären minimal). Außerdem ist sie besser lesbar, so dass weniger Fehler entstehen.

6. Kinematik serieller Roboter

6.1. Beschreibung einer Roboterstellung

6.1.1. Koordinatensysteme und Vektoren

Wir benutzen kartesische Koordinatensysteme mit drei senkrecht zueinander stehenden Koordinatenachsen. Dabei gilt die Rechtsschraubenregel:

Dreht man die x-Achse auf dem kürzesten Weg zur y-Achse hin so ergibt sich ein Drehsinn, der eine Schraube mit Rechtsgewinde in Richtung der positiven z-Achse bewegen würde.

In der Robotik wird mit zwei Arten von Vektoren gearbeitet: *Freie Vektoren* lassen sich im Raum beliebig verschieben. Beispiele für freie Vektoren sind Geschwindigkeit und Beschleunigung. *Ortsvektoren* sind relative Positionsangaben und gehen immer vom Ursprung des Koordinatensystems aus, auf das sie sich beziehen.

Beispiele:
1) Geschwindigkeit, freier Vektor: (T = Transponierung, Spiegelung um die Hauptdiagonale)

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = (v_x, v_y, v_z)^T$$

2) Ortsvektor, bezogen auf ein Koordinatensystem:

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = (x, y, z)^T$$

6.1.2. Transformationen

Vektoren werden unter bestimmten Umständen transformiert, die möglichen Transformationen sind Skalierung, Addition, Subtraktion und Rotation. (Das Vektorprodukt wird hier nicht behandelt.)

Skalierung Alle Komponenten des Vektors werden mit dem gleichen Skalierungsfaktor multipliziert.

$$\vec{u} = s \cdot \vec{v} = \begin{pmatrix} s \cdot v_x \\ s \cdot v_y \\ s \cdot v_z \end{pmatrix} \tag{6.1}$$

Beispiel:
Der Effektor eines Roboters bewegt sich linear mit dem Geschwindigkeitsvektor $\vec{v} = (12, 5, -2)^T$. Nach Beschleunigung auf doppelte Geschwindigkeit ist der Geschwindigkeitsvektor

$$\vec{u} = 2 \cdot \vec{v} = \begin{pmatrix} 2 \cdot v_x \\ 2 \cdot v_y \\ 2 \cdot v_z \end{pmatrix} = \begin{pmatrix} 2 \cdot 12 \\ 2 \cdot 5 \\ 2 \cdot -2 \end{pmatrix} = \begin{pmatrix} 24 \\ 10 \\ -4 \end{pmatrix}$$

Addition/Subtraktion Diese Transformation stellt entweder die Addition freier Vektoren oder die Verschiebung (Translation) von freien Vektoren dar. Zwei Vektoren werden komponentenweise addiert/subtrahiert.

$$\vec{v} = \vec{u} + \vec{w} = \begin{pmatrix} u_x + w_x \\ u_y + w_y \\ u_z + w_z \end{pmatrix} \tag{6.2}$$

Beispiel:
Der Effektor eines Roboters bewegt sich mit dem Geschwindigkeitsvektor $\vec{u} = (5, 3, -2)^T$. Gleichzeitig wird der ganze Roboter auf einer Schiene mit dem Geschwindigkeitsvektor $\vec{w} = (6, 0, 0)^T$ verfahren. Die resultierende Gesamtgeschwindigkeit des Effektors ist

$$\vec{v} = \vec{u} + \vec{w} = \begin{pmatrix} 5 + 6 \\ 3 + 0 \\ -2 + 0 \end{pmatrix} = \begin{pmatrix} 11 \\ 3 \\ -2 \end{pmatrix}$$

Rotation Rotationen werden durch Matrizenmultiplikationen beschrieben.

$$\vec{v} = T \cdot \vec{u}$$

Die Multiplikation zweier Matrizen A, B kann ausgeführt werden, *wenn die Zeilenanzahl von B mit der Spaltenanzahl von A übereinstimmt*. Die Produktmatrix $C = AB$ wird gebildet nach der Rechenregel

$$c_{mn} = \sum_{k=1}^{N} a_{mk} b_{kn} \tag{6.3}$$

Das bedeutet, jedes Element c_{mn} der Matrix C ist das Skalarprodukt von Zeile m der Matrix A und Spalte n der Matrix B. Beispiel:

$$A = \begin{pmatrix} -1 & 5 \\ 0 & 6 \\ 1 & 7 \end{pmatrix} \quad , \quad B = \begin{pmatrix} 1 & 3 \\ 2 & -1 \end{pmatrix}$$

$$C = AB = \begin{pmatrix} -1 \cdot 1 + 5 \cdot 2 & -1 \cdot 3 + 5 \cdot -1 \\ 0 \cdot 1 + 6 \cdot 2 & 0 \cdot 3 + 6 \cdot -1 \\ 1 \cdot 1 + 7 \cdot 2 & 1 \cdot 3 + 7 \cdot -1 \end{pmatrix} = \begin{pmatrix} 9 & -8 \\ 12 & -6 \\ 15 & -4 \end{pmatrix}$$

Beispiel:
Der TCP eines Roboters befindet sich am Punkt $\vec{u} = (7, 2, 5)^T$ bezogen auf ein Koordinatensystem, das sich im Fußpunkt des Roboters befindet. Der Roboter dreht sich nun um 90^o um die z-Achse. Diese Rotation wird durch folgende Transformationsmatrix beschrieben:

$$T = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.4}$$

Die neuen Koordinaten des TCP ergeben sich damit zu:

$$\vec{v} = T \cdot \vec{u} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 2 \\ 5 \end{pmatrix} = \begin{pmatrix} -2 \\ 7 \\ 5 \end{pmatrix}$$

Problematik bei diesen Transformationen: Es gibt keinen einheitlichen Rechenformalismus.

6.1.3. Homogene Koordinaten und homogene Matrizen

Es wird mit 4er-Vektoren und 4 x 4-Matrizen gearbeitet. Die Vektoren haben nun die Form:

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = (x, y, z, 1)^T \tag{6.5}$$

Bei den Matrizen ist die unterste (vierte) Zeile immer $(0, 0, 0, 1)$. Die oben genannten Transformationen lassen sich wie folgt darstellen:

Skalierungen um den Faktor s:

$$Skal(s) = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.6}$$

Vektoradditionen bzw. Translationen (Verschiebungen) um den Vektor $(x, y, z)^T$:

$$Trans(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.7}$$

Rotation um die x-Achse um den Winkel θ:

$$Rot(x,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.8}$$

Rotation um die y-Achse um den Winkel θ:

$$Rot(y,\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.9}$$

Rotation um die z-Achse um den Winkel θ:

$$Rot(z,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.10}$$

Drehrichtungen: Eine Drehung mit positivem Drehwinkel um eine Achse liegt dann vor, wenn die Achsenrichtung dem Daumen und die Drehrichtung den Fingern der rechten Hand entspricht (Rechte-Hand-Regel, Abb. 6.1). Das bedeutet im Einzelnen:

- Eine positive Drehung um die x-Achse dreht die y-Achse auf die z-Achse hin,
- eine positive Drehung um die y-Achse dreht die z-Achse auf die x-Achse hin,
- eine positive Drehung um die z-Achse dreht die x-Achse auf die y-Achse hin.

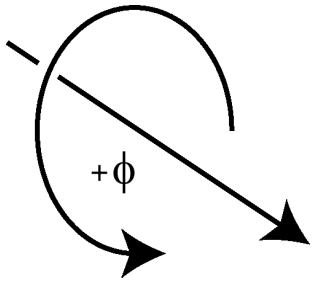


Abbildung 6.1.: Rechte-Hand-Regel: Die Drehrichtung entspricht den Fingern, die Achsenrichtung dem Daumen der rechten Hand.

Homogene Matrizen erlauben also eine formal einheitliche Darstellung der Transformationen Skalierung, Translation, Rotation durch Matrizenmultiplikation.

Beispiel 1 Der Raumpunkt $P_1 = (1, 5, 0, 1)^T$ wird verschoben um den Vektor $(5, 7, -4, 1)^T$. Die Transformationsmatrix ist

$$T = Trans(5, 7, -4) = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Der Raumpunkt erhält die neue Position

$$P_2 = TP_1 = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 5 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ 12 \\ -4 \\ 1 \end{pmatrix}$$

Beispiel 2 Der Raumpunkt $P_1 = (7, 2, 5, 1)^T$ soll um 90^o um die z-Achse gedreht werden. Die Transformationsmatrix ist

$$T = Rot(z, 90) = \begin{pmatrix} \cos 90 & -\sin 90 & 0 & 0 \\ \sin 90 & \cos 90 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Der Raumpunkt erhält die neue Position

$$P_2 = TP_1 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 7 \\ 2 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ 7 \\ 5 \\ 1 \end{pmatrix}$$

Anschaulich: Der z-Wert blieb unverändert, x- und y-Wert haben sich durch eine Drehung im Gegenuhrzeigersinn geändert.

6.1.4. Aufeinanderfolgende Transformationen

Bei aufeinanderfolgenden Transformationen wird der transformierte Punkt erneut einer Transformation unterworfen:

$$\begin{aligned} P_2 &= T_1 P_1 \\ P_3 &= T_2 P_2 = T_2 T_1 P_1 \end{aligned} \tag{6.11}$$

Das Matrizenprodukt $T_2 T_1$ stellt die Summenwirkung der beiden Transformationen dar, wobei T_1 zuerst ausgeführt werden muss.

$$T_2 T_1 = T_{ges} \tag{6.12}$$

6.1.5. Rechenregeln

Matrizenprodukte sind assoziativ:

$$T_2(T_1P_1) = (T_2T_1)P_1 = T_{ges}P_1 \tag{6.13}$$

Matrizenprodukte sind nicht kommutativ (vertauschbar):

$$T_2T_1 \neq T_1T_2 \tag{6.14}$$

Beispiel Der Raumpunkt $P_1 = (5, 6, 7, 1)^T$ soll durch folgende aufeinanderfolgenden Transformationen in einen Punkt transformiert werden, den wir P_3 nennen:

1. Eine Rotation um 90^o um die x-Achse,
2. eine Translation um den Vektor $(1, 2, 3, 1)^T$

Die Transformationsmatrix ist

$$T = Trans(1,2,3) \cdot Rot(x,90) = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Der Raumpunkt erhält die neue Position

$$P_3 = TP_1 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 7 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ -5 \\ 9 \\ 1 \end{pmatrix}$$

Alternativ kann die Transformation des Punktes in zwei Schritten durchgeführt werden. Im ersten Schritt wird die Rotation durchgeführt und wir erreichen den Zwischenpunkt P_2:

$$P_2 = Rot(x,90)P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 7 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ -7 \\ 6 \\ 1 \end{pmatrix}$$

In einem zweiten Schritt wird mit P_2 die Translation durchgeführt:

$$P_3 = Trans(1,2,3)P_2 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ -7 \\ 6 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ -5 \\ 9 \\ 1 \end{pmatrix}$$

6.1.6. Weitere Eigenschaften homogener Matrizen

Um die zu beschreibenden Eigenschaften klar formulieren zu können, benennen wir die Komponenten einer beliebigen homogenen Transformationsmatrix wie folgt:

$$T = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.15}$$

1) Wenn die Transformation aus mehreren Rotationen und einer abschließenden Translation besteht, stellt $\vec{p}$ den Translationsanteil dar und die Vektoren $\vec{n}, \vec{o}, \vec{a}$ den Rotationsanteil dar.

2) Die Vektoren $\vec{n}, \vec{o}, \vec{a}$ sind orthogonal:

$$\vec{n} \times \vec{o} = \vec{a}$$
$$\vec{a} \times \vec{n} = \vec{o}$$
$$\vec{o} \times \vec{a} = \vec{n}$$

3) Die Vektoren $\vec{n}, \vec{o}, \vec{a}$ geben die Orientierung des betreffenden Koordinatensystems an: $\vec{n}$ ist der Normalenvektor, $\vec{o}$ der Orientierungsvektor und $\vec{a}$ der Annäherungsvektor.

4) Die inverse Matrix T^{-1} wird wie folgt berechnet:

$$T^{-1} = \begin{pmatrix} n_x & n_y & n_z & -\vec{p} \cdot \vec{n} \\ o_x & o_y & o_z & -\vec{p} \cdot \vec{o} \\ a_x & a_y & a_z & -\vec{p} \cdot \vec{a} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.16}$$

wobei $\vec{p}\vec{n}$ das Skalarprodukt der Vektoren $\vec{p}$ und $\vec{n}$ ist.

6.1.7. Beschreibung der Orientierung

6.1.7.1. Drehung um raumfeste Achsen

Die Orientierung eines Körpers enthält drei Freiheitsgrade. Sie kann mathematisch auf verschiedene Arten formuliert werden. Wir betrachten hier zunächst die Beschreibung der Orientierung durch drei aufeinanderfolgende Drehungen um die Achsen eines raumfesten Koordinatensystems. Diese Beschreibeung der Orientierung ist beispielsweise in der Seefahrt üblich, wo man je nach Rotationsachse vom Rollen, Beugen oder Gieren des Schiffes spricht (Abb. 6.2). In Bezug auf die Handachsen spricht man oft von Drehen, Beugen und Neigen, auch DBN-System (Drehung, Beugung, Neigung) (Abb. 6.3):

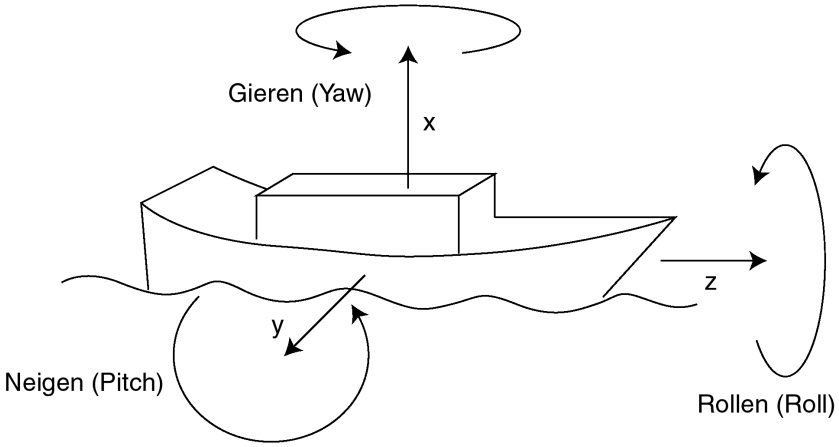


Abbildung 6.2.: Rollen, Neigen und Gieren beim Schiff.

1. Drehung um die x-Achse um den Winkel ϕ_x (Neigung),
2. Drehung um die y-Achse um den Winkel ϕ_y (Beugung),
3. Drehung um die z-Achse um den Winkel ϕ_z (Drehung).

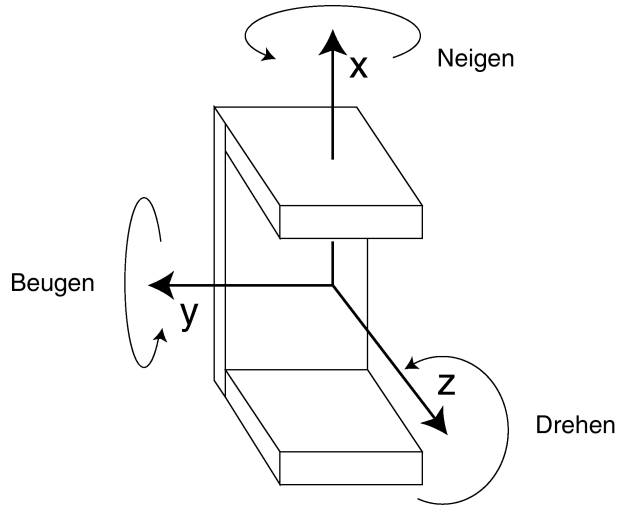


Abbildung 6.3.: Drehung, Beugung und Neigung an einer Roboterhand.

Die zugehörige Transformationsmatrix erhält man durch aufmultiplizieren der Rotationsmatrizen (Gl. 6.8 – Gl. 6.10):

$$\begin{aligned} T_O &= Rot(z,\phi_z)Rot(y,\phi_y)Rot(x,\phi_x) \\ &= \begin{pmatrix} c_z c_y & c_z s_y s_x - s_z c_x & c_z s_y c_x + s_z s_x & 0 \\ s_z c_y & s_z s_y s_x + c_z c_x & s_z s_y c_x - c_z s_x & 0 \\ -s_y & c_y s_x & c_y c_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{6.17}$$

Wobei die Bedeutung der Abkürzungen wie folgt ist: $s_x = \sin\phi_x, c_x = \cos\phi_x, s_y = \sin\phi_y \ldots$. Wenn nun die Transformationsmatrix gegeben ist, kann durch Ver-

gleich mit Gl. 6.17 der Orientierungsanteil bestimmt werden und durch die drei Drehwinkel ϕ_x, ϕ_y, ϕ_z ausgedrückt werden. Da hierbei die Winkel gesucht werden, die der Transformation zu Grunde liegen, handelt es sich um ein *inverses Problem*. Wenn man die Elemente von T_O wiederum wie in Gl. 6.15 bezeichnet, ergibt sich durch elementweises Gleichsetzen:

3. Zeile, 1. Spalte: $\sin\phi_y = -n_z \quad \phi_y = \arcsin(-n_z)$

3. Zeile, 2. Spalte: $\cos\phi_y \sin\phi_x = o_z \quad \phi_x = \arcsin(\frac{o_z}{\cos\phi_y})$

1. Zeile, 1. Spalte: $\cos\phi_z \cos\phi_y = n_x \quad \phi_z = \arccos(\frac{n_x}{\cos\phi_y})$

Diese Bestimmungsgleichungen führen aus mehreren Gründen bisweilen zu Problemen:

- arcsin und arccos haben in manchen Bereichen eine unendliche Steigung, dies führt zu Bestimmungs- und Genauigkeitsproblemen.
- Bei $\phi_y = 90^o$ ergibt sich eine Division durch Null.
- arcsin und arccos sind mehrdeutig.

Man kann aber aus Gl. 6.17 weitere Beziehungen ableiten, die eine u.U. bessere Bestimmung der Orientierungswinkel gestatten [6]:

$$\begin{aligned} \phi_z &= \arctan(\frac{n_y}{n_x}) \\ \phi_y &= \arctan(\frac{-n_z}{n_x\cos\phi_z + n_y\sin\phi_z}) \\ \phi_x &= \arctan(\frac{a_x\sin\phi_z - a_y\cos\phi_z}{o_y\cos\phi_z - o_x\sin\phi_z}) \end{aligned} \tag{6.18}$$

Beispiel Die Hand eines Roboters wird aus dem raumfesten Koordinatensystem erzeugt durch die Transformation ${}^0T_H$. Wie ist die Orientierung der Hand?

$${}^0T_H = \begin{pmatrix} 0 & -1 & 0 & 6 \\ 0.8660 & 0 & 0.5000 & 7 \\ -0.5000 & 0 & 0.8660 & 8 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Um die Orientierung zu berechnen, wird der Translationsanteil $(6,7,8,1)^T$ außer acht gelassen und die Rotationswinkel nach Gl. 6.18 bestimmt. Es ergibt sich:

$$\begin{aligned} \phi_z &= \arctan(\frac{n_y}{n_x}) = \arctan(\infty) = 90^o \\ \phi_y &= \arctan(\frac{-n_z}{n_x\cos\phi_z + n_y\sin\phi_z}) = \arctan(\frac{0.5}{0.8660}) = 30^o \\ \phi_x &= \arctan(\frac{a_x\sin\phi_z - a_y\cos\phi_z}{o_y\cos\phi_z - o_x\sin\phi_z}) = \arctan(0) = 0^o \end{aligned}$$

6.1.7.2. Euler-Winkel

Ein in der Praxis sehr übliche Beschreibung sind die Euler-Winkel. Dabei stellt man sich vor, dass das verdrehte Koordinatensystem durch drei aufeinanderfolgende Drehungen aus einem Bezugssystem entsteht. Die erste Drehung erfolgt um eine Achse des Bezugssystems, die zweite und dritte Drehung um Achsen der schon verdrehten Zwischensysteme. Die Orientierung wird durch die drei Drehwinkel beschrieben, die auch Euler-Winkel heißen. [2]

Z-Y-Z-Euler-Winkel Die erste Drehung erfolgt um die Z-Achse des Bezugssystems und erzeugt das Koordinatensystem K'. Die zweite Drehung erfolgt um die Y-Achse von K' (die y'-Achse) und erzeugt das Zwischensystem K''. Die dritte Drehung erfolgt um die Z-Achse von K'' (die z''-Achse) und erzeugt das Zielkoordinatensystem.

Z-Y-X-Euler-Winkel Die erste Drehung erfolgt um die Z-Achse des Bezugssystems und erzeugt das Koordinatensystem K'. Die zweite Drehung erfolgt um die Y-Achse von K' (die y'-Achse) und erzeugt das Zwischensystem K''. Die dritte Drehung erfolgt um die X-Achse von K'' (die x''-Achse) und erzeugt das Zielkoordinatensystem.

6.1.7.3. Quaternionen

Auch mit Quaternionen[1] kann man eine Orientierung beschreiben und manche Roboterhersteuerung (z.B. ABB-Steuerungen) machen davon Gebrauch. Quaternionen sind Zahlen, die aus vier Komponenten q_1, q_2, q_3, q_4 bestehen und folgende Normierungsbedingung erfüllen:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{6.19}$$

Die Komponenten der Quaternionen können aus der Transformationsmatrix 6.15 berechnet werden. [10] Abschließend sei erwähnt, dass natürlich alle Beschreibungen für die Orientierung ineinander umgerechnet werden können.

6.1.7.4. Beispiele zur Darstellung der Orientierung von Körpern

[1] Die Quaternionen gehen auf den irischen Mathematiker Hamilton zurück.

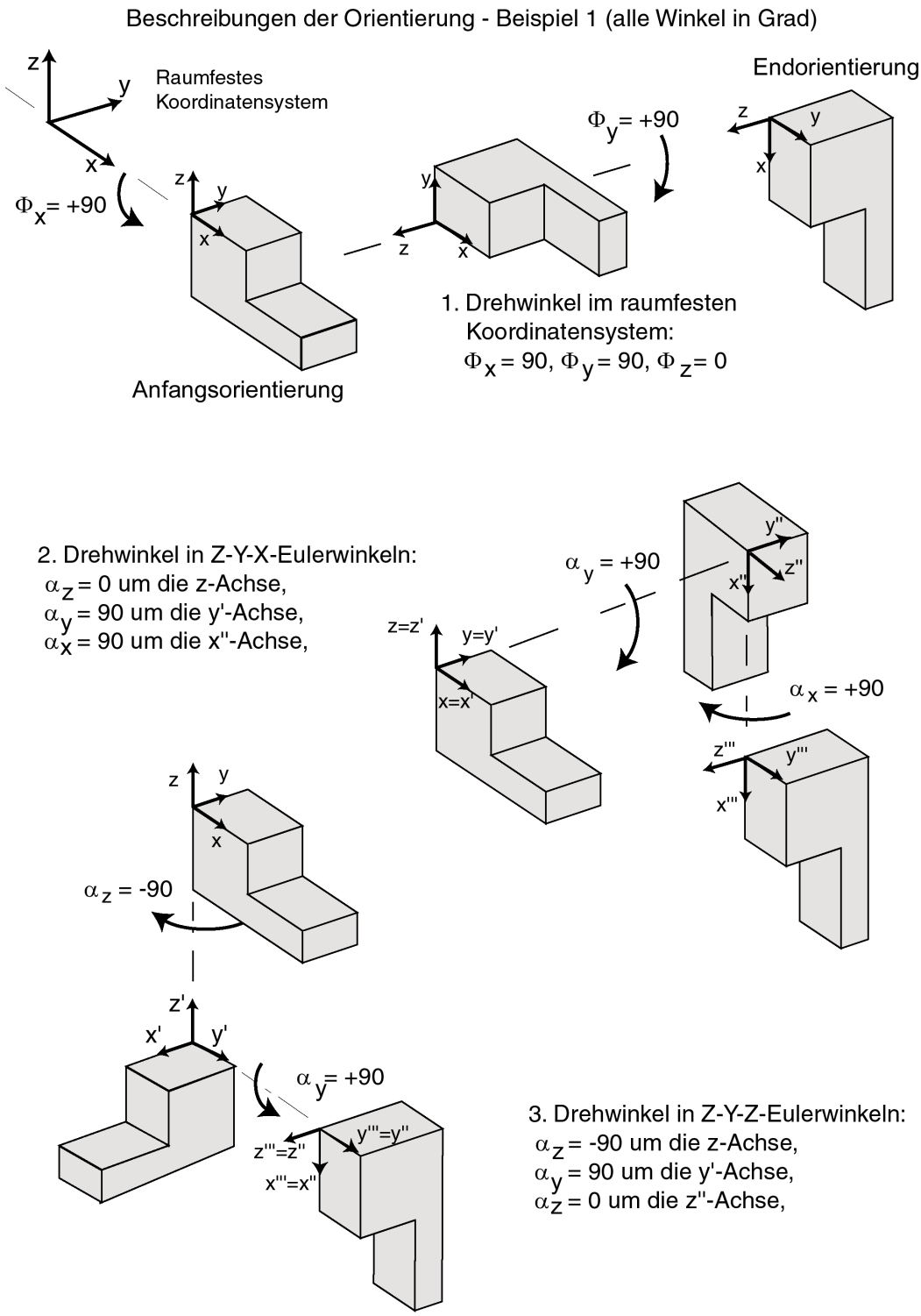


Abbildung 6.4.: Beispiel 1

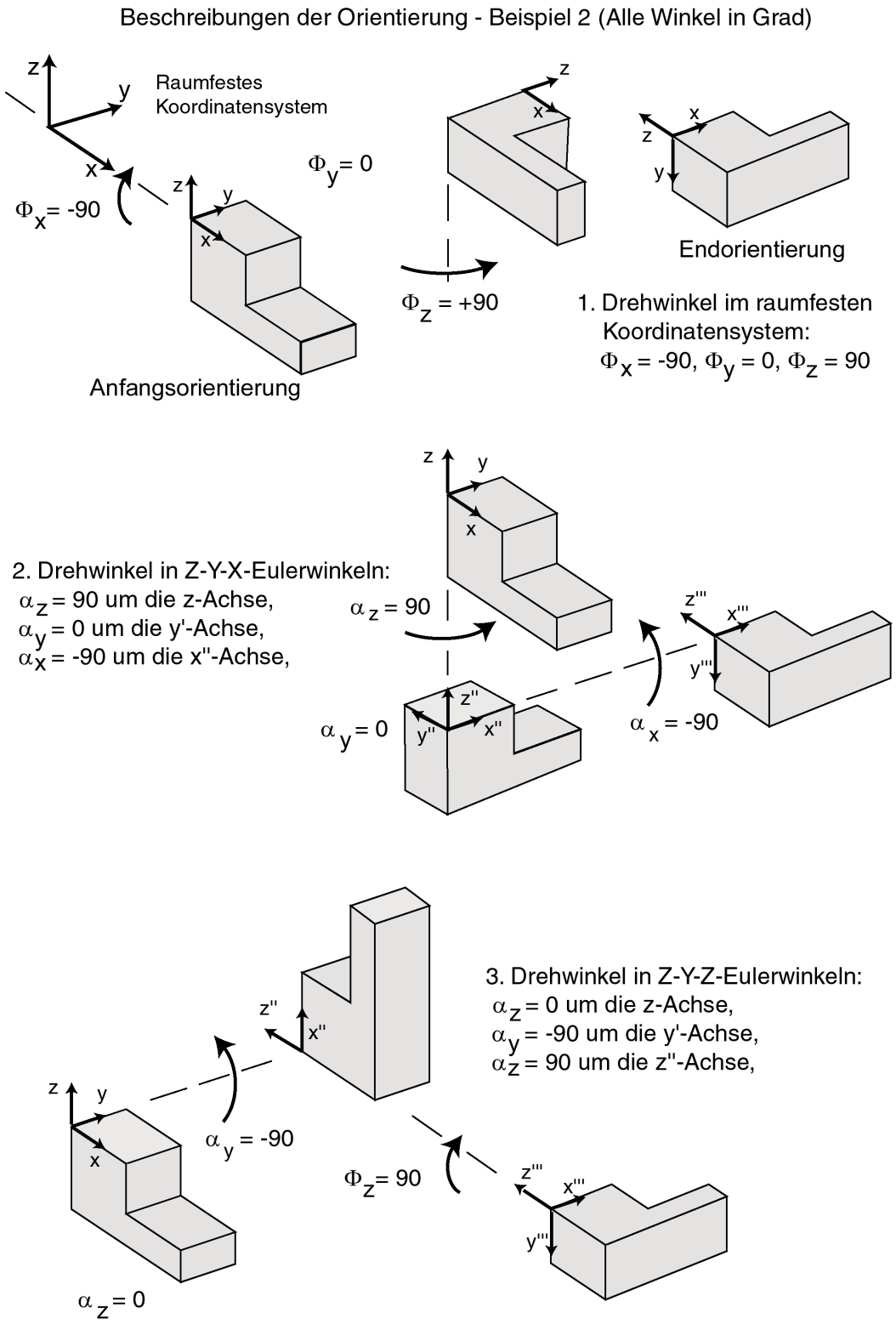


Abbildung 6.5.: Beispiel 2

6.1.8. Bezugssysteme

6.1.8.1. Verwendung von Bezugssystemen

Unter Bezugssystemen versteht man die Verwendung mehrerer Koordinatensysteme bei der Beschreibung von Roboterteilen und Werkstücken. Jedes Bezugssystem ist einem Körper zugeordnet, das können Armteile, Werkzeuge, Werkstücke oder Hindernisse sein. Durch die Benutzung von Bezugssystemen lassen sich viele Probleme einfacher formulieren, weil einerseits Zeitabhängigkeiten beseitigt werden können und andererseits problemangepasste Koordinatenachsen gelegt werden können. Eine besondere Anwendung der Bezugssysteme werden wir bei der kinematischen Kette kennenlernen, die Grundlage für die Vorwärts- und Rückwärtstransformation ist. Dabei wird jedes Armteil durch ein Koordinatensystem dargestellt, das fest mit dem Armteil verbunden ist und es bezüglich Lage und Orientierung eindeutig beschreibt. Einige Beispiele für Bezugssysteme:

- Die Längsachse eines Werkzeuges steht schief im Raum, verwendet man Werkzeugkoordinaten, so stimmt sie mit deren z-Achse überein.
- Die Position eines Werkstückes im Greifer verändert sich während der Bewegung, relativ zum TCP ist sie konstant.
- Ein Pfad auf einem Werkstück verläuft schräg im Raum, es lassen sich aber Werkstückkoordinaten einführen, bei denen der Pfad z.B. mit der x-Achse übereinstimmt und nicht mehr von der Lage des Werkstückes abhängt.
- Die Lage von Werkstückdetails hängt von der Lage des Werkstücks ab und bewegt sich mit diesem; relativ zu einem Werkstückkoordinatensystem ist sie konstant.
- Die Lage mehrerer Werkstücke auf einer Werkstückaufnahme ist meistens verdreht gegen die Weltkoordinaten; nach Einführung eines eigenen Bezugssystems für die Werkstückaufnahme ergeben sich einfache und zeitkonstante Relationen.
- Die Lage eines Armteiles im Raum hängt von vielen Gelenkwinkeln ab; relativ zu dem vorherigen Armteil hängt sie nur noch von einem Gelenkwinkel ab.

Die Verwendung von Bezugssystemen ist sehr charakteristisch für die Robotik. Die Robotersteuerungen und die Robotersprachen sind darauf eingerichtet, mit unterschiedlichen Bezugssystemen zu arbeiten. Koordinatensysteme heißen in diesem Zusammmenhang auch *frames*. In der Roboterprogrammierung ist die Verwendung folgender Koordinatensysteme üblich:

Weltkoordinaten Fest mit der Welt (Fußboden verbunden)

Basiskoordinaten (base frame) Fest mit dem Sockel des Roboters verbunden, relativ zu den Weltkoordinaten definiert, oft mit diesen identisch.

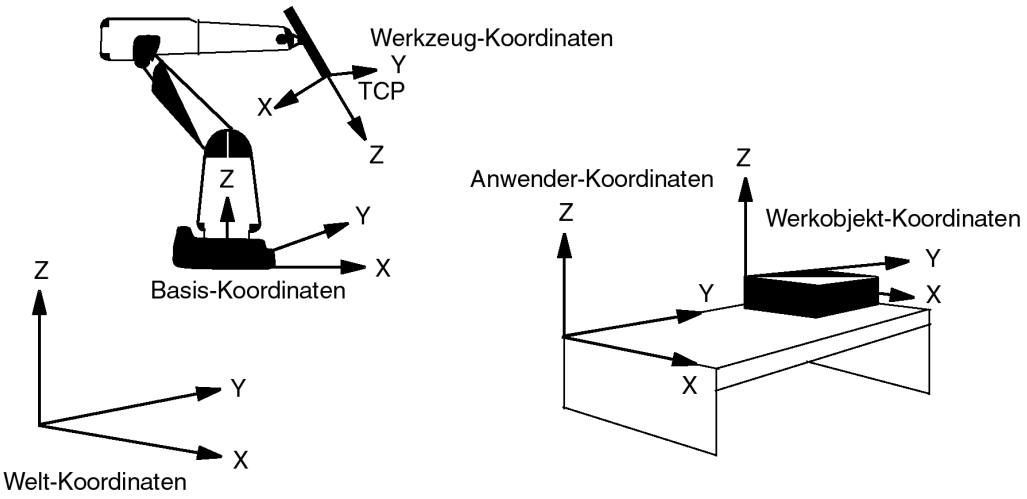
Anwenderkoordinaten (user frame) Mit einer Aufnahmevorrichtung für Werkstücke verbunden, oft relativ zu Weltkoordinaten definiert.

Werkstückkoordinaten (object frame) Mit einem Werkstück verbunden, oft relativ zu Anwenderkoordinaten definiert.

Handflanschkoordinaten (tool0) Mit dem Handflansch verbunden, mitbewegt und über die kinematisch Kette der Gelenke relativ zu den Basiskoordinaten festgelegt.

Werkzeugkoordinaten (tool frame) Relativ zu den Handflanschkoordinaten definiert.

Die untenstehende Abbildung 6.6 stammt aus dem Handbuch eines Industrieroboters und zeigt die typische Verwendung der Koordinatensysteme in der Praxis.



Die verschiedenen Koordinatensysteme des Manipulators (wenn der Manipulators das Werkzeug hält) Abbildung: ABB Robotics.

Abbildung 6.6.: Übliche Koordinatensysteme und ihr Bezug zueinander in einer industriellen Roboterzelle. (ABB)

Die rechnerische Transformation eines Koordinatensystems erfolgt am einfachsten dadurch, dass der Urpsrung und die Einheitsvektoren der Transformation unterworfen werden, die transformierten Punkte spannen dann das neue Koordinatensystem auf.

Beispiel Die Hand eines Roboters wird durch ein Koordinatensystem K_H repräsentiert, das aus dem Fußkoordinatensystem K_0 hervorgeht durch

1. Rotation um -90^o um die x-Achse,

2. Translation um $P_1 = (-3, 10, 0)^T$

Die Transformationsmatrix ist

$$\begin{aligned} {}^0T_H &= Trans(-3,10,0) \cdot Rot(x,-90) \\ &= \begin{pmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -3 \\ 0 & 0 & 1 & 10 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Diese Transformation kann nun auf den Ursprung des Koordinatensystems und die Endpunkte der Einheitsvektoren angewandt werden: Die Transformation des Ursprungs ergibt den Ursprung des neuen Koordinatensystems:

$${}^0T_H \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -3 \\ 10 \\ 0 \\ 1 \end{pmatrix}$$

Ebenso wird mit dem Endpunkt des Einheitsvektors e_x verfahren:

$${}^0T_H \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ 10 \\ 0 \\ 1 \end{pmatrix}$$

Der Endpunkt des Einheitsvektors e_y wird transformiert nach:

$${}^0T_H \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -3 \\ 10 \\ -1 \\ 1 \end{pmatrix}$$

Der Endpunkt des Einheitsvektors e_z schließlich wird transformiert nach:

$${}^0T_H \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -3 \\ 11 \\ 0 \\ 1 \end{pmatrix}$$

Der transformierte Ursprung wird nun durch Geraden mit den transformierten Endpunkten verbunden und man erhält so die Achsen x', y', z' des neuen, transformierten Koordinatensystems H (s. Abb.6.7).

Verwendet man mehrere Koordinatensysteme um die Lage von Körpern zu beschreiben, spricht man auch von *Bezugssystemen*. Bezugssysteme gehen durch Transformationen aus anderen Koordinatensystemen hervor. Bei Verwendung verschiedener Bezugssysteme wird durch einen links hochgestellten Index angezeigt, auf welches Bezugssystem sich eine Koordinatenangabe bezieht, z.B.

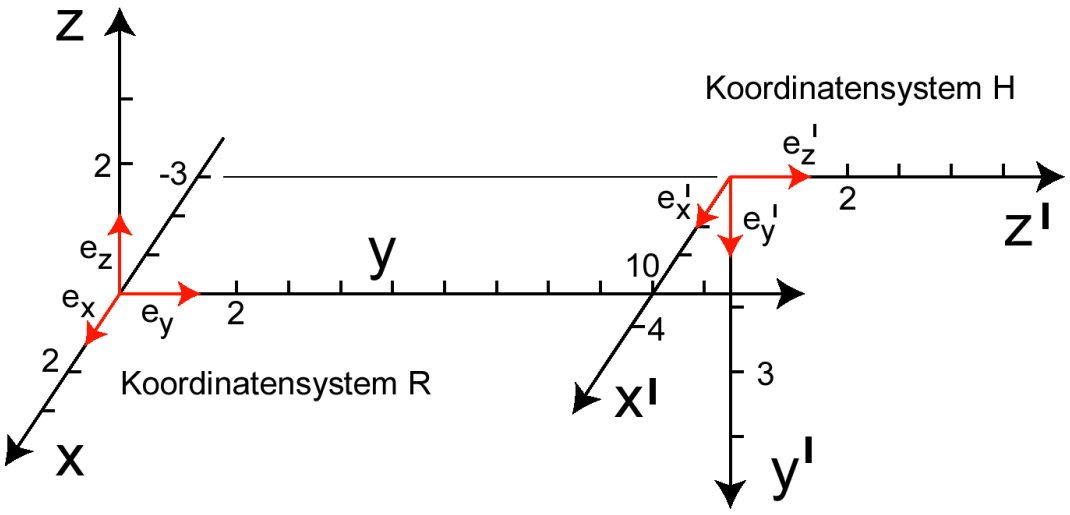


Abbildung 6.7.: Koordinatensystem H wird durch die Transformation im obigen Beispiel aus Koordinatensystem R abgeleitet.

${}^{A}P$ der Punkt P in Koordinaten des Bezugssystems A,
${}^{B}P$ der Punkt P in Koordinaten des Bezugssystems B.

Wenn eine Transformation ${}^{A}T_{B}$ das Koordinatensystem A in Koordinatensystem B überführt, dann transformiert ${}^{A}T_{B}$ auch Punkte von der Darstellung in B-Koordinaten in die Darstellung in A-Koordinaten.

$$K_B = {}^{A}T_{B} K_A \tag{6.20}$$
$${}^{A}P = {}^{A}T_{B}\, {}^{B}P \tag{6.21}$$

Die inverse Matrix $({}^{A}T_{B})^{-1}$ wirkt genau umgekehrt: Sie transformiert Koordinatensystem B in Koordinatensystem A und Punkte von A-Koordinaten in B-Koordinaten.

$$K_A = ({}^{A}T_{B})^{-1} K_B \tag{6.22}$$
$${}^{B}P = ({}^{A}T_{B})^{-1}\, {}^{A}P \tag{6.23}$$

Aus der letzten Gleichung ist die Transformation der Koordinaten ersichtlich. Die inverse Matrix kann daher auch geschrieben werden als:

$$({}^{A}T_{B})^{-1} = {}^{B}T_{A} \tag{6.24}$$

Dazu ein einfaches eindimensionales Beispiel. Ein eindimensionales Koordinatensystem K_B werde aus dem Koordinatensystem K_A erzeugt durch Translation um den Betrag $+4$ in x-Richtung. Dies ist in Abbildung 6.8 dargestellt (nur x-Achsen gezeichnet): Die Transformationsmatrix ist

$${}^{A}T_{B} = \begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

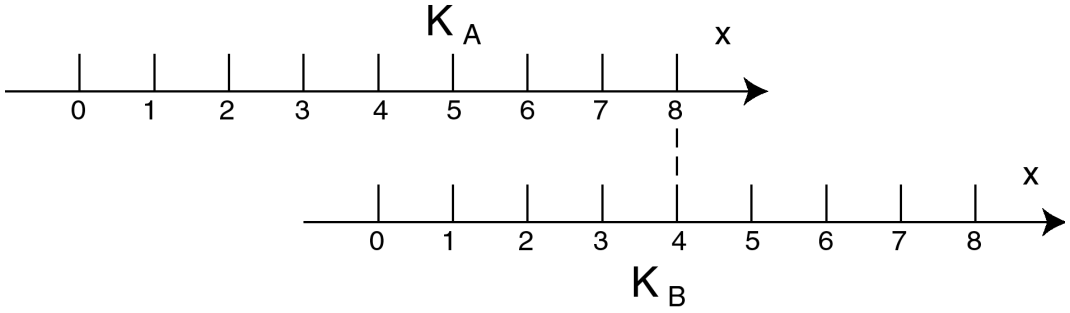


Abbildung 6.8.: Ein Koordinatensystem K_B wird aus Koordinatensystem K_A erzeugt. Die Transformation ist eine Translation um 4 Einheiten in x-Richtung. (nur x-Achsen gezeichnet)

Die gleiche Matrix transformiert von B-Koordinaten auf A-Koordinaten, so wird z.B. aus dem Punkt ${}^{B}P = {}^{B}(4,0,0,1)$ in B-Koordinaten der Punkt ${}^{A}P = {}^{A}(8,0,0,1)$ in A-Koordinaten. Dieser Punkt ist in der obigen Abbildung gekennzeichnet.

$$ {}^{A}T_{B}\,{}^{B}P = \begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 0 \\ 0 \\ 1 \end{pmatrix} = {}^{A}P $$

Die inverse Matrix dazu ist

$$ ({}^{A}T_{B})^{-1} = {}^{B}T_{A} = \begin{pmatrix} 1 & 0 & 0 & -4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Sie transformiert A- zu B-Koordinaten. So wird z.B. aus dem Punkt ${}^{A}(8,0,0,1)$ in A-Koordinaten der Punkt ${}^{B}(4,0,0,1)$ in B-Koordinaten.

6.1.8.2. Drei und mehr Bezugssysteme

Bei Verwendung von drei und mehr Bezugssystemen können alle Bezugssysteme durch Transformationsmatrizen zueinander in Bezug gesetzt werden. Auf diese Weise sind auch Ketten- und Zirkularbezüge möglich. Bei der formalen Beschreibung entstehen Gleichungen, die Matrizenprodukte enthalten. Durch Multiplikation mit inversen Matrizen kann nach einzelnen Transformationsmatrizen aufgelöst werden. Das folgende Beispiel möge dies verdeutlichen [6]:

Beispiel Der Robotereffektor enthält einen Bohrer und bohrt ein Loch in ein Objekt. Es gibt folgende Koordinatensysteme:

1. Basiskoordinatensystem (Koordinatensystem des Roboterfußes) K_B

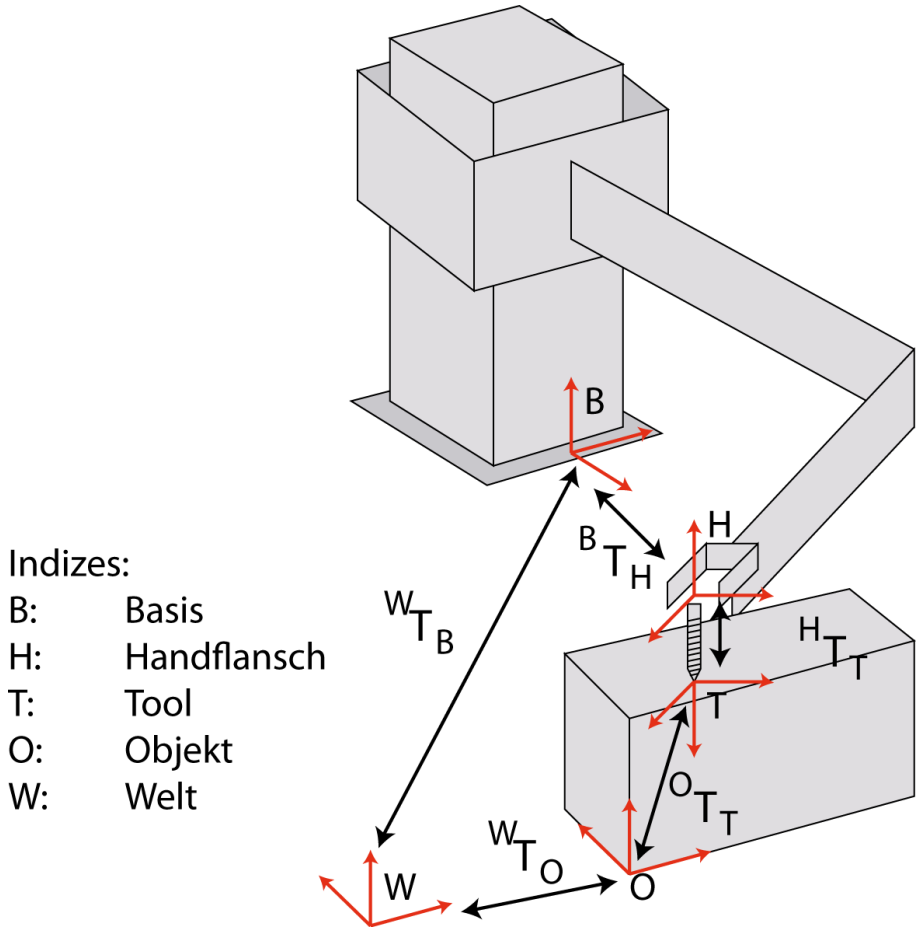


Abbildung 6.9.: Verwendung mehrere Bezugssysteme am Beispiel eines Roboters, der ein Loch in ein Objekt bohrt.

2. Handflanschkoordinatensystem K_H
3. Toolkoordinatensystem (Werkzeugkoordinaten) K_T
4. Objektkoordinatensystem K_O
5. raumfestes Weltkoordinatensystem K_W

Die Transformation vom TCP (Tool-KS) über Handflansch und Fuß des Roboters zu Weltkoordinaten ist

$$^{W}T_{T} = ^{W}T_{B}\ ^{B}T_{H}\ ^{H}T_{T}$$

Die Lage des Bohrloches (=TCP) ist aber auch relativ zum Werkstück fixiert und dieses wiederum hat festen Bezug zum Weltkoordinatensystem. Also gilt auch:

$$^{W}T_{T} = ^{W}T_{O}\ ^{O}T_{T}$$

Es gibt also zwei Transformationen von der Bohrerspitze zum Weltkoordinatensystem. Wenn der Bohrer die korrekte Bohrposition einnimmt, sind beide Transformationen gleich. Die Gleichsetzung ergibt:

$$^{W}T_{O}\ ^{O}T_{T} = ^{W}T_{B}\ ^{B}T_{H}\ ^{H}T_{T}$$

Um nun die Stellung des Roboterarmes zu bestimmen, muss nach ${}^{B}T_{H}$ aufgelöst werden. Dazu wird zunächst von links mit $({}^{W}T_{B})^{-1}$ multipliziert und dann von rechts mit $({}^{H}T_{T})^{-1}$. Es ergibt sich:

$$\begin{aligned} ({}^{W}T_{B})^{-1}\,{}^{W}T_{O}\,{}^{O}T_{T} &= {}^{B}T_{H}\,{}^{H}T_{T} \\ ({}^{W}T_{B})^{-1}\,{}^{W}T_{O}\,{}^{O}T_{T}\,({}^{H}T_{T})^{-1} &= {}^{B}T_{H} \end{aligned}$$

Diese Transformation ${}^{B}T_{H}$ beschreibt die notwendige Stellung des Roboters. Sie wird nun durch die Robotersteuerung der Rückwärtstransformation unterzogen um die Gelenkwinkel zu bestimmen, anschließend kann die Position (aus einer geeigneten Näherungsposition) angefahren werden.

In Kap.5 finden sich weitere Einzelheiten über die praktische Verwendung von Bezugssystemen.

Beispiel In einer Roboterzelle misst ein Kamerasystem einen Punkt P am Werkstück, der sich in der Nähe des Effektors befindet. Im Kamerakoordinatensystem sind die Koordinaten des Punktes ${}^{K}P = (2, 0, 3, 1)^{T}$. Das Kamerakoordinatensystem geht aus den Basiskoordinaten hervor durch eine Rotation um 90^{o} um die y-Achse und eine anschließende Translation um $(0, 50, 10, 1)^{T}$. Die Handkoordinaten gehen aus den Basiskoordinaten hervor durch eine Rotation um -90^{o} um die x-Achse und eine anschließende Translation um $(10, 40, 8, 1)^{T}$.

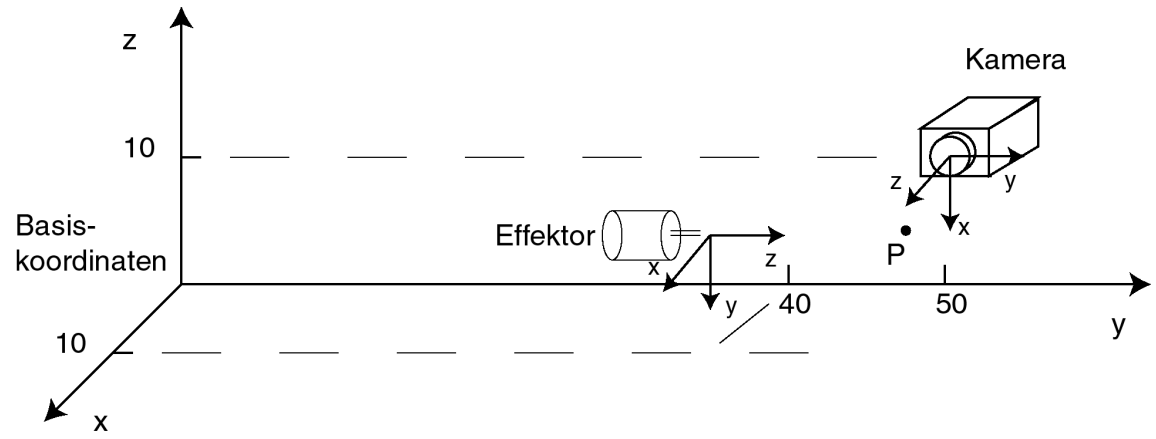


Abbildung 6.10.: Roboterzelle mit Kamerasystem

Bestimmen Sie
a) den Punkt P in Basiskoordinaten,
b) den Punkt P in Effektorkoordinaten.

Lösung
Die Transformationsmatrizen sind:

$${}^{B}T_{K} = Trans(0, 50, 10) \cdot Rot(y, 90^{o}) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 50 \\ -1 & 0 & 0 & 10 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$ {}^{B}T_{E} \;=\; Trans(10,40,8)\cdot Rot(x,-90^{o}) = \begin{pmatrix} 1 & 0 & 0 & 10 \\ 0 & 0 & 1 & 40 \\ 0 & -1 & 0 & 8 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Teil a)
Der Punkt ${}^{K}P$ ist in Kamerakoordinaten bekannt und kann über die Transformation ${}^{B}T_{K}$ direkt in Basiskoordinaten umgerechnet werden:

$$ {}^{B}P = {}^{B}T_{K}\,{}^{K}P = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 50 \\ -1 & 0 & 0 & 10 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 50 \\ 8 \\ 1 \end{pmatrix} $$

Teil b)
Wenn der Punkt P in Effektorkoordinaten formuliert ist kann ${}^{E}P$ ebenso über die Effektor-Basis-Transformation in Basiskoordinaten umgerechnet werden. Gleichsetzen ergibt:

$$ {}^{B}P = {}^{B}T_{K}\,{}^{K}P = {}^{B}T_{E}\,{}^{E}P $$

Diese Gleichung kann nach dem gesuchten ${}^{E}P$ aufgelöst werden:

$$ {}^{E}P = ({}^{B}T_{E})^{-1}\,{}^{B}T_{K}\,{}^{K}P = ({}^{B}T_{E})^{-1}\,{}^{B}P $$

Die inverse Matrix von ${}^{B}T_{E}$ bestimmen wir nach Gl. 6.16:

$$ ({}^{B}T_{E})^{-1} = \begin{pmatrix} 1 & 0 & 0 & -10 \\ 0 & 0 & -1 & 8 \\ 0 & 1 & 0 & -40 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Aufmultiplizieren von ${}^{B}P$ ergibt das Ergebnis:

$$ {}^{E}P = \begin{pmatrix} 1 & 0 & 0 & -10 \\ 0 & 0 & -1 & 8 \\ 0 & 1 & 0 & -40 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 50 \\ 8 \\ 1 \end{pmatrix} = \begin{pmatrix} -7 \\ 0 \\ 10 \\ 1 \end{pmatrix} $$

Beispiel

In einer Fertigungsstraße steht ein Lackierroboter hinter einer Autokarosserie. Die Bezüge zwischen Weltkoordinaten, Basiskoordinaten, Effektorkoordinaten und Objektkoordinaten sind wie folgt gegeben:

- Das Basiskoordinatensystem geht aus den Weltkoordinaten hervor durch eine Translation um $(20,30,0)^{T}$
- Das Objektkoordinatensystem geht aus den Weltkoordinaten hervor durch eine Rotation um 180^{o} um die z-Achse und eine anschließende Translation um $(350,0,130)^{T}$.

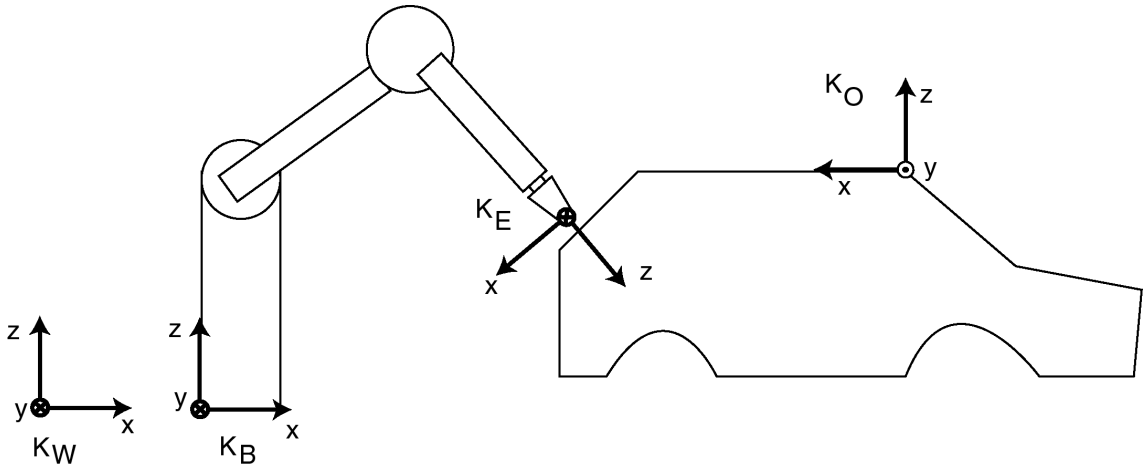


Abbildung 6.11.: Roboterzelle mit Effektor und Werkobjekt

- Das Effektorkoordinatensystem geht aus den Basiskoordinaten hervor durch eine Rotation um 135^o um die y-Achse und eine anschließende Translation um $(150, 0, 125)^T$.
- Ein Punkt P ist in Objektkoordinaten gegeben durch ${}^oP = (200, 25, -10)^T$

Bestimmen Sie die Koordinaten des Punktes P in Weltkoordinaten, Basiskoordinaten und Effektorkoordinaten!

Ergebnisse:

$$ {}^WP = \begin{pmatrix} 150 \\ -25 \\ 120 \\ 1 \end{pmatrix} \quad {}^BP = \begin{pmatrix} 130 \\ -55 \\ 120 \\ 1 \end{pmatrix} \quad {}^EP = \begin{pmatrix} 17.67 \\ -55 \\ -10.6 \\ 1 \end{pmatrix} $$

6.2. Allgemeines zu den kinematischen Transformation

Ein sehr wichtiges Problem der Robotik ist das Umrechnen zwischen den kartesischen Koordinaten und den Gelenkkoordinaten. Die kartesischen Koordinaten sind x,y,z sowie die drei Orientierungswinkel, gemessen in einem zur Roboterzelle gehörenden Koordinatensystem. Beim Entwurf des Bearbeitungsvorganges wird man immer in kartesischen Koordinaten arbeiten und denken. Oft werden direkt Maße einbezogen, z.B. Konstruktionsdaten, da bleiben nur kartesischen Koordinaten. Die kartesischen Koordinaten sind auch immer die bevorzugten Koordinaten für den Menschen beim Programmiervorgang.

Anders sieht es aus der Ebene der Robotersteuerung aus: Der Roboter erreicht sein Ziel nur deshalb, weil jedes Gelenk exakt auf den richtigen Winkel bzw. die richtige Länge gefahren wird. Die Robotersteuerung auf der maschinennahen Ebene muss bei einer Bewegung den Servomotoren des Roboters korrekte Zielangaben machen und diese müssen im internen Koordinatensystem der

Servomotoren formuliert sein. Dieses sind die Gelenkkoordinaten und auf der unteren Steuerungsebene können nur sie benutzt werden. Die Gelenkkoordinaten heißen auch Gelenkwinkel, Maschinenkoordinaten oder Achskoordinaten.

Die Robotersprachen enthalten sowohl Bewegungsbefehle die kartesiche Koordinaten benutzen (Normalfall) als auch Bewegungsbefehle, die Gelenkkoordinaten benutzen (seltener genutzt).

Die Umrechnung zwischen kartesischen und Gelenkkoordinaten heißen *kinematische Transformationen*. Wann ist nun eine kinematische Transformation erforderlich? Wenn die Steuerung einen kartesisch programmierten Punkt anfährt, muss sie von den kartesischen Koordinaten auf Gelenkkoordinaten transformieren (umrechnen). Bei einer CP-Bewegung muss das in sehr kurzen Abständen geschehen, um die Bahn kontinuierlich unter Kontrolle zu halten. Möchte man dagegen für einen vorgegebenen Satz von Gelenkwinkeln nachrechnen, wo der TCP sich befinden wird, wenn diese Gelenkkoordinaten angefahren sind braucht man eine Transformation von Gelenkkoordinaten auf kartesische Koordinaten.

Zur Unterscheidung benutzt man folgende Namensregelung:

- Die Umrechnung von Gelenkkoordinaten auf kartesische Koordinaten heißt *kinematische Vorwärtstransformation*.
- Die Umrechnung von kartesischen Koordinaten auf Gelenkkoordinaten heißt *kinematische Rückwärtstransformation* oder auch *inverse kinematische Transformation*.

Bei der mathematischen Ausführung der Transformationen nutzt man die Tatsache, dass es sich hier um *serielle Geometrien* handelt. Das bedeutet, dass an jedem Armteil über ein Gelenk das nächste Armteil befestigt ist. Die Armteile bilden also eine Kette, man spricht von der *kinematischen Kette* wie in Abb.6.12 gezeigt ist. Die kinematische Kette beginnt beim Sockel (Fuß) des Roboters, in den man sich ein erstes Koordinatensystem gelegt denkt: $K_0 = K_R$ ist das raumfeste Fußkoordinatensystem (R=Root=Wurzel); es bewegt sich nicht, wenn der Roboter sich bewegt. Man wird z.B. die Lage des Werkstücks relativ zum raumfesten Koordinatensystem K_0 beschreiben. (siehe auch Abschnitt 6.1.8) Die kinematische Kette führt bis zum Handflansch, der durch das Handflansch-Koordinatensystem $K_N = K_H$ beschrieben wird. Wenn ein Effektor montiert ist, gibt es eine weitere Transformation zum Effektorkoordinatensystem K_E, dessen Ursprung der TCP ist. Hier endet die kinematische Kette. [2]

Die Vorwärts- und die Rückwärtstransformation an der kinematischen Kette können auf zwei Arten durchgeführt werden:

[2] Im praktischen Betrieb kann man an der Robotersteuerung einstellen, ob nur bis zum Handflansch gerechnet werden soll (bei ABB: tool0) oder bis zum TCP eines eingemessenen Werkzeugs.

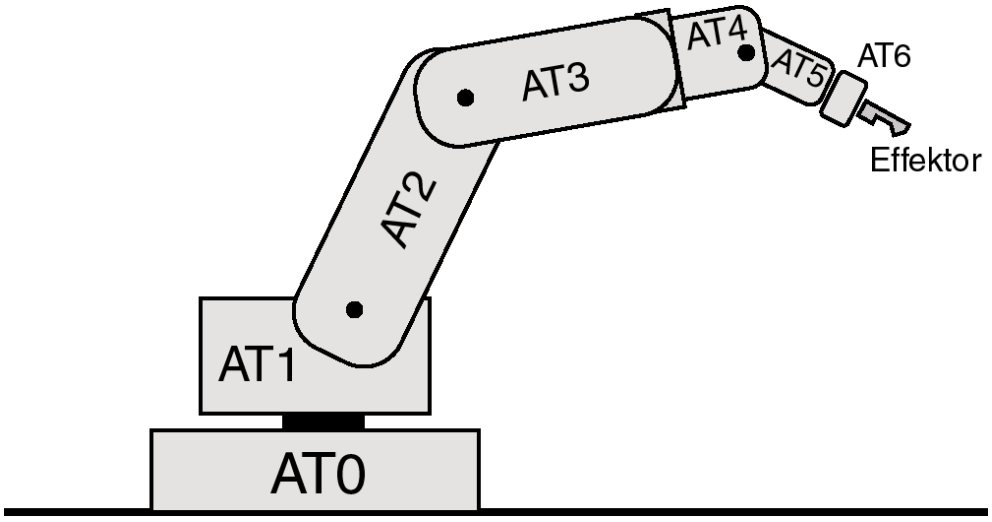


Abbildung 6.12.: Die Armteile eines Roboters bilden eine kinematische Kette, in diesem Beispiel 6 Armteile (N=6). AT=Armteil

Elementargeometrische Rückwärtstransformation (Vektor-Loop-Methode)

Man betrachtet die beteiligten Winkel und Hebelarme und versucht mit den elementaren Mitteln der Geometrie (meistens über Dreiecke) darzustellen, wie die kartesischen Koordinaten des TCP sich aus den Gelenkkoordinaten (Maschinenkoordinaten) des Roboters ergeben. Wenn die Zahl der Achsen klein oder die Geometrie des Roboters einfach ist, kann recht gut so gearbeitet werden. Die Vorgehensweise kann als Bildung einer Kette von Vektoren aufgefasst werden, die vom Fuß bis zum Effektor reicht. Man spricht deshalb auch von der *Vektor-Loop-Methode*.

Matrizenmethode

Eine etwas allgemeinere Methode für die kinematischen Transformationen ist die Matrizenmethode, die auch für kompliziertere Geometrien funktioniert. Dabei legt man in jedes Armteil des Roboters ein eigenes Koordinatensystem und beschreibt mit Matrizen jeweils den Übergang von einem Armteil zum nächsten. Diese Matrizen enthalten sowohl die konstruktiven Parameter (Armlängen und Achsenlage) sowie die Gelenkkoordinate des Gelenks, das zu diesem Armteil gehört. Vollzieht man dann nacheinander alle Transformationen vom ersten bis zum letzten Armteil und zum TCP, so hat man unter Einbeziehung der Gelenkkoordinaten die gesamte kinematische Transformation vom Roboterfuß zum TCP vollzogen. Für die Matrizenmethode gibt es einen Vereinheitlichungsvorschlag von Denavit und Hartenberg, der in A genauer besprochen ist.

Die kinematische Kette muss nicht immer bis zum Handflansch oder Effektor geführt werden. es ist auch möglich, die Kette zu verkürzen und damit Position und Orientierung von Zwischengelenken zu berechnen.

6.3. Die kinematische Vorwärtstransformation

Bei N Gelenken nennen wir die Gelenkkoordinaten $\theta_1 \dots \theta_N$ gegeben, wobei dies je nach Bauart des Gelenks ein Winkel oder ein Weg ist. Position und Orientierung des Effektors (Hand) sind $x, y, z, \phi_x, \phi_y, \phi_z$ und werden berechnet. Die kinematische Vorwärtstransformation entspricht also folgender Abbildung:

$$\begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ . \\ . \\ \theta_N \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ \phi_x \\ \phi_y \\ \phi_z \end{pmatrix} \tag{6.25}$$

Wir werden ein einfaches Beispiel elementargeometrisch durchrechnen und ein etwas komplexeres mit der Matrizenmethode.

6.3.1. Kinematische Vorwärtstransformation elementargeometrisch am TR-Roboter

An unserem TR-Roboter (siehe) gibt es nur zwei Gelenkkoordinaten: Die Verschiebung des Schlittens um den Betrag d aus der Nullstellung heraus und das hochschwenken um den Winkel θ. Zwei Achsen reichen natürlich nicht aus um einen dreidimensionalen Arbeitsraum zu erreichen und auch nicht zur freien Orientierung. Im Überblick

Gelenkkordinaten:	d, θ
Kartesische Koordinaten	x, y, z (immer 0) ϕ_x (immer 0), ϕ_y (immer 0), ϕ_z (nicht frei wählbar)

Man kann hier die kartesischen Achsen frei wählen, benennt auch die Gelenkkoordinaten und betrachtet dann die geometrischen Zusammenhänge. (Abb.6.13)

Man erhält sehr einfach die Abhängigkeit der kartesischen von den Gelenkkoordianten :

$$x = d + L\cos\theta \tag{6.26}$$
$$y = L\sin\theta \tag{6.27}$$
$$\phi_z = \theta \tag{6.28}$$

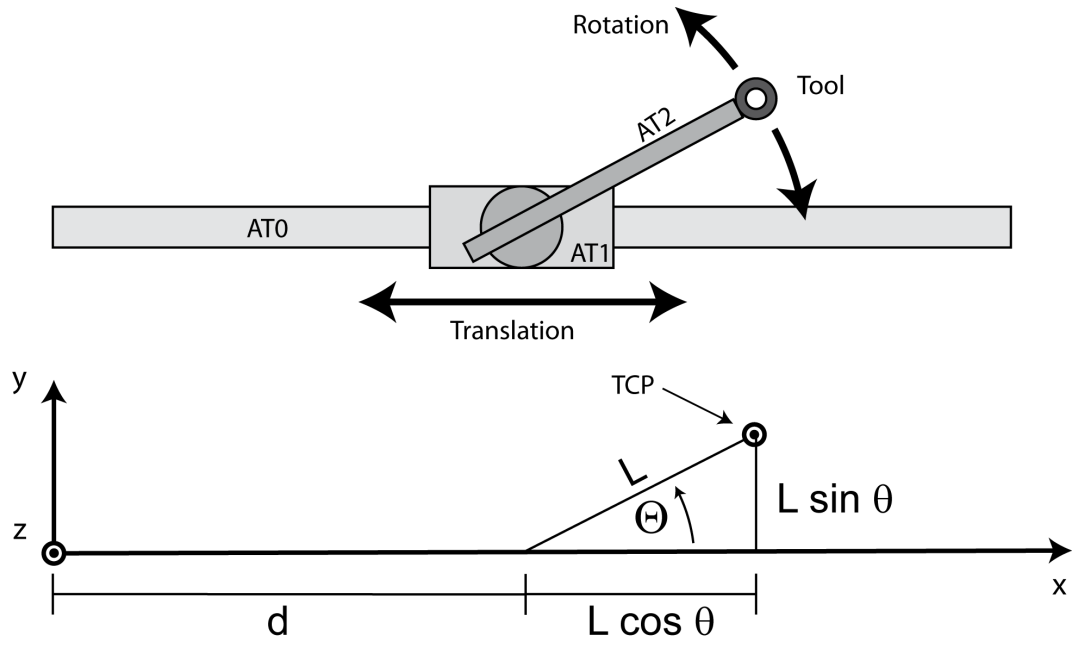


Abbildung 6.13.: Am TR-Roboter können die kartesischen Koordinaten einfach aus den Gelenkkordinaten bestimmt werden.

6.3.2. Kinematische Vorwärtstransformation mit der Matrizenmethode

Mit der Matrizenmethode lässt sich die Vorwärtstransformation einfach nach Schema durchführen. Diese Transformationsmatrizen sollten gemäß den Konventionen von Denavit und Hartenberg fomuliert sein. Diese Konventionen führen zu einer Vereinheitlichung und sind in Anhang A ausführlich behandelt.

Jedes Armteil enthält ein Koordinatensystem, das man sich fest mit dem Armteil verbunden denkt und das die Stellung des Armteiles genau beschreibt. Jeder Übergang auf das nächste Armteil wird durch eine Transformationsmatrix dargestellt. Diese Transformationsmatrix beschreibt die Stellung eines Armteiles in Bezug auf das Armteil, an dem es montiert ist. (Vorgänger). Für einen N-achsigen Roboter sind die Koordinatensysteme der Armteile K_1, K_2 usw. Die kinematische Kette ist daher in der formalen Darstellung ein mehrfaches Matrizenprodukt. Da es sich jetzt um Transformationsmatrizen mit besonderer Bedeutung handelt, verwenden wir hier ein als Symbol A statt T. Außerdem kürzen wir wie folgt ab:

$$^{n-1}A_n = A_n$$

Dann gilt für die kinematische Kette:

$$\begin{aligned} K_1 &= A_1\, K_0 \\ K_2 &= A_2\, K_1 \\ &\vdots \\ K_n &= A_n\, K_{n-1} \quad \text{mit} \quad n \leq N \end{aligned} \tag{6.29}$$

Wenn man die K_i schrittweise ersetzt kann man daraus ableiten:

$$\begin{aligned} K_1 &= A_1\, K_0 \\ K_2 &= A_2 A_1\, K_0 \\ K_3 &= A_3 A_2 A_1\, K_0 \\ &\vdots \\ K_n &= A_n \cdots A_3 A_2 A_1\, K_0 \quad \text{mit} \quad n \leq N \end{aligned} \tag{6.30}$$

Für die Transformation von Punktkoordinaten gilt:

$$\begin{aligned} {}^0P &= A_1\, {}^1P \\ {}^1P &= A_2\, {}^2P \\ {}^2P &= A_3\, {}^3P \\ &\vdots \\ {}^{n-1}P &= A_n\, {}^nP \quad \text{mit} \quad n \leq N \end{aligned} \tag{6.31}$$

Woraus wiederum schrittweise abgeleitet werden kann:

$$\begin{aligned} {}^0P &= A_1\, {}^1P \\ {}^0P &= A_1 A_2\, {}^2P \\ {}^0P &= A_1 A_2 A_3\, {}^3P \\ &\vdots \\ {}^0P &= A_1 A_2 A_3 \cdots A_n\, {}^nP \quad \text{mit} \quad n \leq N \end{aligned} \tag{6.32}$$

Das Matrizenprodukt $A_1 A_2 A_3 \cdots A_N$ ist also die Transformation vom Koordinatensystem K_N auf die raumfesten Fußkoordinaten K_0 und wird hier als ${}^0T_N$ bezeichnet.

$${}^0P = A_1 A_2 A_3 \cdots A_N\, {}^NP = {}^0\,T_N\, {}^NP \tag{6.33}$$

Die kinematischen Transformationen mit der Matrizenmethode erfolgen also in folgenden Schritten:

1. Identifizieren der Armteile (N Stück)
2. Ein Koordinatensystem in jedes Armteil legen, am besten entsprechend den Denavit-Hartenberg-Konventionen.
3. Die Transformationen zwischen den Koordinatensystemen bestimmen, ebenfalls am besten entsprechend den Denavit-Hartenberg-Konventionen.
4. Die so bestimmten Transformationen als Matrix formulieren
5. Alle Matrizen aufmultiplizieren, um die kinematische Kette als eine Gesamt-Transformationsmatrix ${}^0T_N$ zu erhalten
6. An dieser Gesamt-Transformationsmatrix entweder die kartesischen Koordinaten aus den Maschinenkoordinaten bestimmen (Vorwärtstransformation) oder die Maschinenkoordinaten aus den kartesischen Koordinaten bestimmen (Rückwärtstransformation)

6.3.3. Die Vorwärtstransformation am TR-Roboters mit Matrizen

Als Beispiel wollen wir wieder unseren TR-Roboter heranziehen den wir schon früher als Beispielroboter verwendet haben (siehe Seiten **??**, 28, 30 und 80).

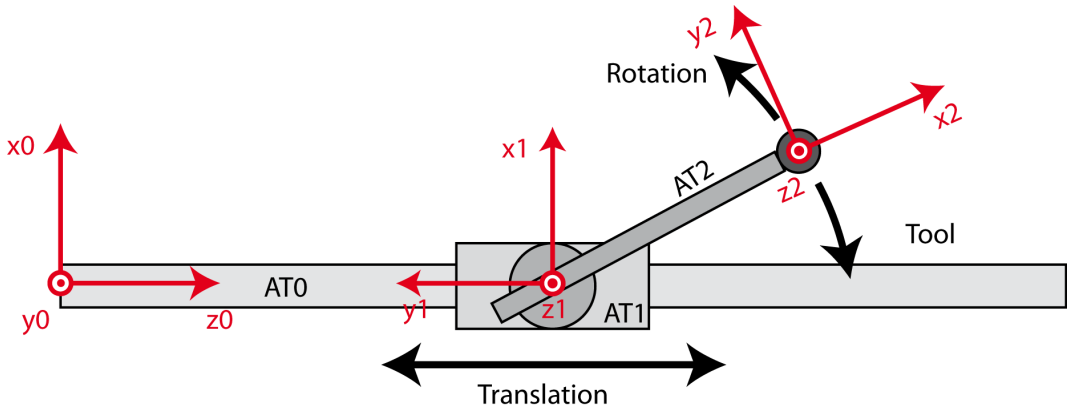


Abbildung 6.14.: Die zwei Achsen des TR-Roboters mit Koordinatensystemen gemäß Denavit-Hartenberg-Konventionen.

Zunächst werden wir in jedes Armteil ein Koordinatensystem gemäß Denavit-Hartenberg (DH) legen. Das Koordinatensystem K_0 ist mit dem Fuß (Sockel) des Roboters verbunden, seine z-Achse muss in der Gleitachse von Armteil 1 (Schlitten) liegen. Koordinatensystem K_1 liegt in Armteil 1 (Schlitten), seine z-Achse muss die Drehachse von Armteil 2 sein. Bei Koordinatensystem K_3 gibt es mehr Freiheit, es wurde so gelegt, dass es einfach aus K_1 abzuleiten ist. In Abb. 6.14 sind beiden Achsen unseres TR-Roboters und die gemäß DH-Konvention in die Armteile gelegten Koordinatensysteme gezeigt. Es ergeben sich folgende DH-Parameter:

Achse	Matrix	θ_i	d_i	a_i	α_i
1	A_1	0	d (variabel)	0	$-\pi/2$
2	A_2	$\theta - \pi/2$	0	L	0

Aus diesen DH-Parametern bildet man nun durch Einsetzen in Gleichung A.1 die gehörigen Denavit-Hartenberg-Transformationsmatrix A_1 und A_2. Dabei beschreibt A_1 die Transformation von K_0 zu K_1 und A_2 die Transformation von K_1 zu K_2. Diese Matrizen drücken also die relative Lage der Koordinatensysteme und damit der Armteile aus. In den Matrizen stecken sowohl konstruktive Parameter (L) wie auch Gelenkkoordinaten (d, θ).

Das Aufmultiplizieren von A_1 und A_2 ergibt eine Transformation, die die gesamte kinematische Kette mit $N = 2$ beschreibt. Wir verwenden die Abkürzungen $s2 = \sin\theta_2, c2 = \cos\theta_2$:

$$^0T_N = A_1 A_2$$

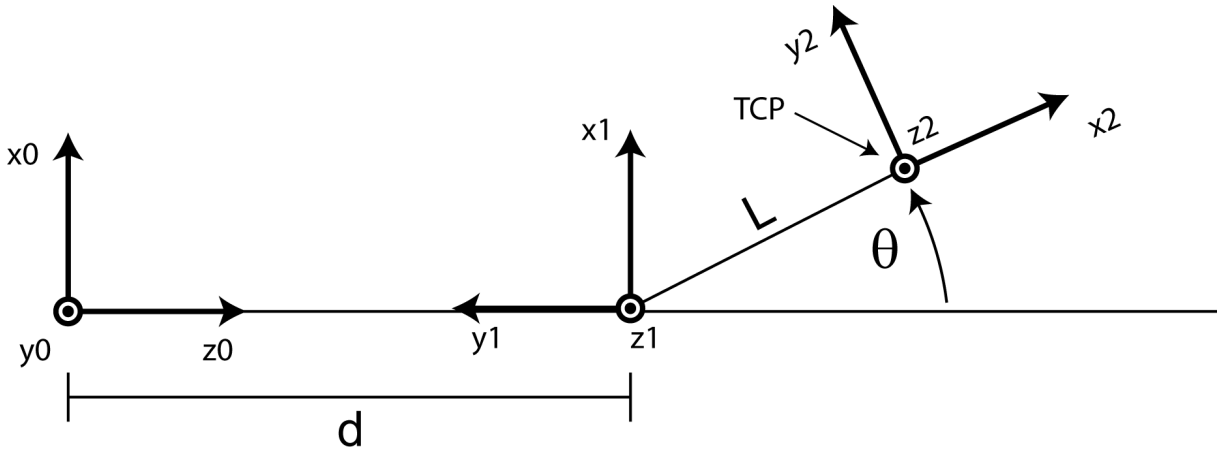


Abbildung 6.15.: Der TR-Roboter, hier sind nur noch die Koordinatensysteme der kinematischen Kette gezeichnet.

$$
\begin{aligned}
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \sin\theta & \cos\theta & 0 & L\sin\theta \\ -\cos\theta & \sin\theta & 0 & -L\cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \sin\theta & \cos\theta & 0 & L\sin\theta \\ 0 & 0 & 1 & 0 \\ \cos\theta & -\sin\theta & 0 & d + L\cos\theta \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{6.34}
$$

Aus dieser Transformationsmatrix wird nun der Translationsanteil und der Rotationsanteil bestimmt. Die Position lässt sich als Translationsanteil aus der rechten Spalte entnehmen:

$$
\begin{aligned}
x &= p_x = L\sin\theta \\
y &= P_y = 0 \\
z &= p_z = d + L\cos\theta
\end{aligned} \tag{6.35}
$$

Die Orientierung ergibt sich aus den Gleichungen 6.18

$$
\begin{aligned}
\phi_z &= \arctan(\frac{n_y}{n_x}) = 0 \\
\phi_y &= \arctan(\frac{-n_z}{n_x\cos\phi_z + n_y\sin\phi_z}) = \theta - \pi/2 \\
\phi_x &= \arctan(\frac{a_x\sin\phi_z - a_y\cos\phi_z}{o_y\cos\phi_z - o_x\sin\phi_z}) = -\pi/2
\end{aligned}
$$

Wir erhalten also die gleichen Ergebnisse, wie bei der elementargeometrischen Vorwärtstransformation, nur sind hier die Koordinaten wegen der DH-Konventionen anders gewählt.

6.3.4. Die Vorwärtstransformation an den Grundachsen des SCARA-Roboters mit Matrizen

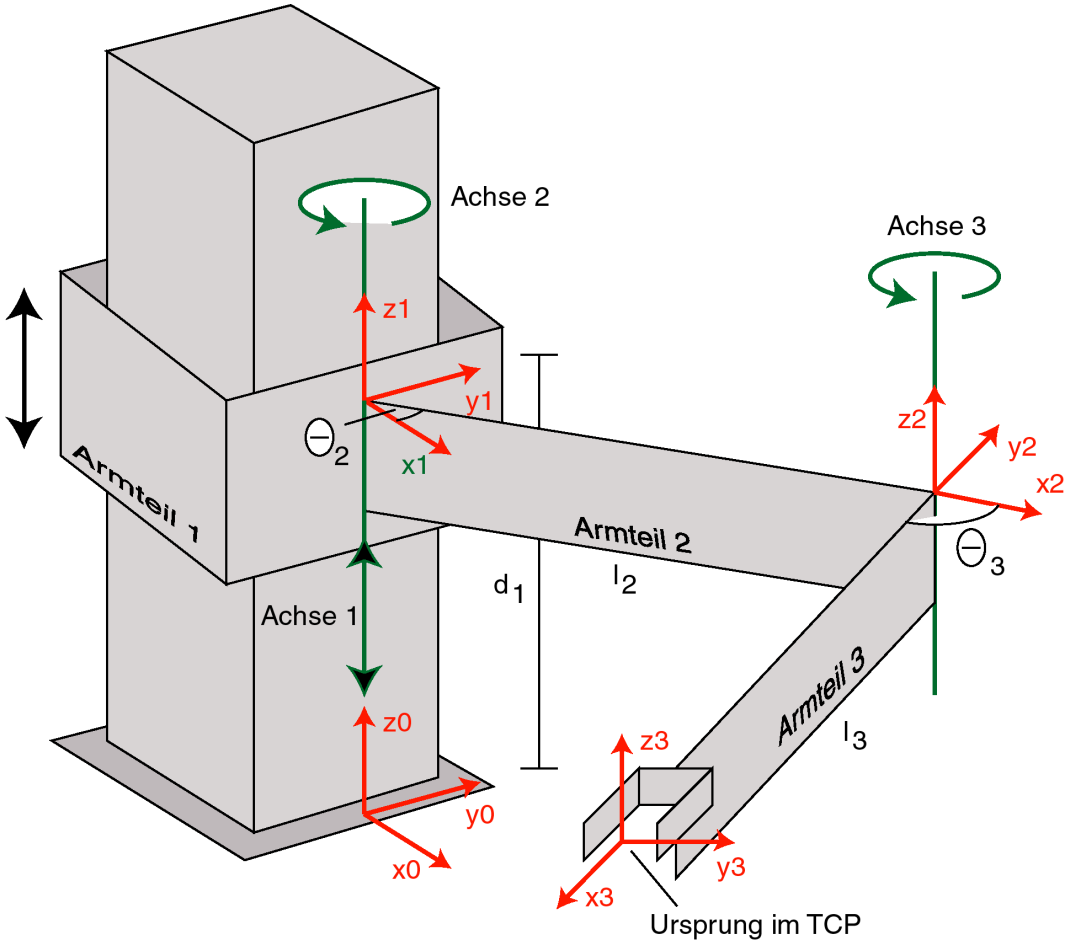


Abbildung 6.16.: Die drei Grundachsen eines SCARA-Roboters mit Koordinatensystemen gemäß Denavit-Hartenberg-Konventionen.

In Abb. 6.16 sind die drei Grundachsen eines SCARA-Roboters und die gemäß DH-Konvention gelegten Koordinatensysteme gezeigt. Die Ausrichtung der x_0, y_0-Achsen ist frei und wurde so gewählt, dass sich zu K_1 möglichst wenig Änderungen ergeben. Die Achsen z_0 und z_1 wurden in einer Linie angeordnet um die Transformationsgleichungen einfach zu halten. Die übrigen Achsen wurden entsprechend den oben beschriebenen DH-Konventionen gelegt. Es ergeben sich folgende DH-Parameter:

Achse	Matrix	α_i	d_i	a_i	θ_i
1	A_1	$\alpha_1 = 0$	d_1 (variabel)	$a_1 = 0$	$\theta_1 = 0$
2	A_2	$\alpha_2 = 0$	$d_2 = 0$	$a_2 = l_2$	θ_2 (variabel)
3	A_3	$\alpha_3 = 0$	$d_3 = 0$	$a_3 = l_3$	θ_3 (variabel)

Als Transformationsmatrix ergibt sich mit $N = 3$ für die gesamte kinematische Kette mit den Abkürzungen $s2 = \sin\theta_2, c2 = \cos\theta_2, s3 = \sin\theta_3, c3 = \cos\theta_3$:

$$
\begin{aligned}
{}^0T_N &= A_1 A_2 A_3 \\
&= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_2 & -s_2 & 0 & c_2 l_2 \\ s_2 & c_2 & 0 & s_2 l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_3 & -s_3 & 0 & c_3 l_3 \\ s_3 & c_3 & 0 & s_3 l_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} c_2 c_3 - s_2 s_3 & -c_2 s_3 - s_2 c_3 & 0 & c_2 c_3 l_3 - s_2 s_3 l_3 + c_2 l_2 \\ s_2 c_3 + c_2 s_3 & -s_2 s_3 - c_2 c_3 & 0 & s_2 c_3 l_3 + c_2 s_3 l_3 + s_2 l_2 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{6.36}
$$

Da die Aufgabe der Vorwärtstransformation ja die Bestimmung von Position $\vec{p}$ und Orientierung $\vec{\phi}$ der Hand ist, muss nun aus dieser Transformationsmatrix der Translationsanteil und der Rotationsanteil bestimmt werden. Die Position lässt sich als Translationsanteil aus der rechten Spalte entnehmen:

$$
\begin{aligned}
p_x &= c_2 c_3 l_3 - s_2 s_3 l_3 + c_2 l_2 \\
p_y &= s_2 c_3 l_3 + c_2 s_3 l_3 + s_2 l_2 \\
p_z &= d_1
\end{aligned} \tag{6.37}
$$

Um die Orientierung der Hand zu bestimmen muss man aus der Transformationsmatrix den Rotationsanteil benutzen und die Drehwinkel gemäß Gl. 6.18 bestimmen. Es ergibt sich:

$$
\begin{aligned}
\phi_x &= 0 \\
\phi_y &= 0 \\
\phi_z &= \theta_2 + \theta_3
\end{aligned} \tag{6.38}
$$

Bei der Bestimmung von ϕ_z kann Gebrauch von den folgenden Formeln für die Sinus- und Kosinuswerte der Summe bzw. Differenz von Winkeln gemacht werden:

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta \tag{6.39}$$

$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta \tag{6.40}$$

Der Wert von ϕ_z stimmt mit der Anschauung überein wie man sieht, wenn man den SCARA-Arm in x-y-Projektion darstellt (Abb. 6.17).

Zahlenbeispiel Es sei

$$\left.\begin{aligned} l_2 &= 1 \\ l_3 &= 1 \end{aligned}\right\} \text{konstruktive Parameter}$$

$$\left.\begin{aligned} \theta_2 &= 45^o \\ \theta_3 &= 45^o \\ d_1 &= 5 \end{aligned}\right\} \text{Gelenkstellung, variable Parameter}$$

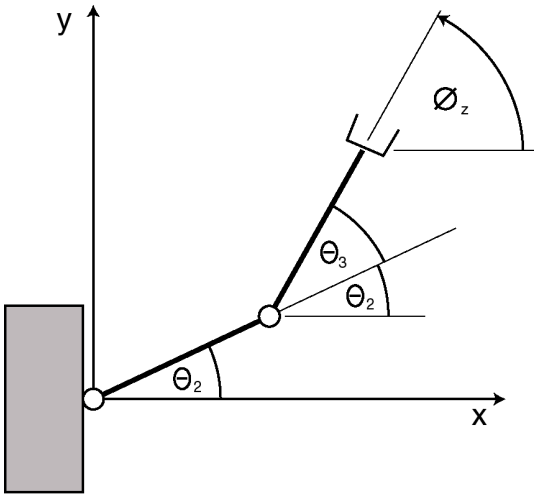


Abbildung 6.17.: Der Roboter mit SCARA-Grundachsen in der x-y-Projektion.

Man bestimme
a) Position und Orientierung der Hand,
b) die Position des Ellenbogens.

a) Bei der Berechnung der Position mit Gln. 6.37 ist $s2 = c2 = s3 = c3 = \sqrt{1/2}$. Damit ergibt sich:

$$p_x = \sqrt{1/2} \approx 0.707 \quad , \quad p_y = 1 + \sqrt{1/2} \approx 1.707 \quad , \quad p_z = 5$$

Die Orientierung ergibt sich mit Gln. 6.38 zu:

$$\phi_x = 0 \quad , \quad \phi_y = 0 \quad , \quad \phi_z = 90^o$$

b) Die Position des Ellenbogens ergibt sich aus Gl. 6.32 mit $n = 2$.

$$\begin{aligned} {}^0P_E &= A_1 A_2 \, {}^2P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_2 & -s_2 & 0 & c_2 l_2 \\ s_2 & c_2 & 0 & s_2 l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} {}^2P \\ &= \begin{pmatrix} c_2 & -s_2 & 0 & c_2 l_2 \\ s_2 & c_2 & 0 & s_2 l_2 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} {}^2P \end{aligned} \tag{6.41}$$

In diese Transformationsmatrix werden die oben gegebenen Werte eingesetzt. Um die Position des Ellenbogens zu erhalten transformiert man nun einfach den Ursprung von K_2 in raumfeste Koordinaten (K_0).

$$P_E = \begin{pmatrix} \sqrt{1/2} & -\sqrt{1/2} & 0 & \sqrt{1/2} \\ \sqrt{1/2} & \sqrt{1/2} & 0 & \sqrt{1/2} \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \sqrt{1/2} \\ \sqrt{1/2} \\ 5 \\ 1 \end{pmatrix}$$

Es ist also $p_{Ex} = p_{Ey} = \sqrt{1/2} \approx 0.707$ und $p_{Ez} = 5$. Die x-y-Projektion der Stellung des Armes ist in Abb. 6.18 gezeigt.

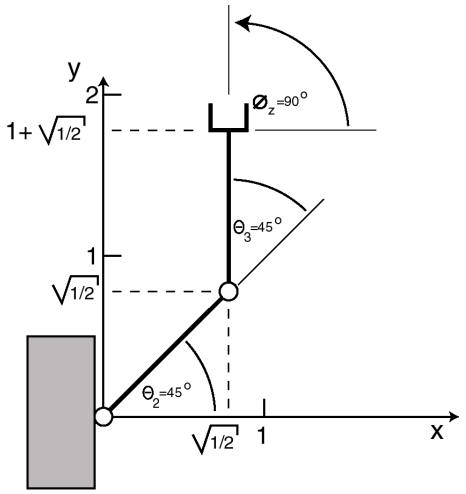


Abbildung 6.18.: Die Stellung des im Beispiel berechneten SCARA-Roboters.

6.4. Die kinematische Rückwärtstransformation

6.4.1. Allgemeines

Die Rückwärtstransformation berechnet aus vorgegebenen kartesischen Koordinaten und Orientierung des Handflansches (oder Effektors) die Gelenkkoordinaten des Roboters. Bei N Gelenken sind also die Position und Orientierung des Effektors (Hand) $x, y, z, \phi_x, \phi_y, \phi_z$ gegeben und die Gelenkwinkel/-linearkoordinaten $\theta_1 \dots \theta_N$ werden berechnet.

$$\begin{pmatrix} x \\ y \\ z \\ \phi_x \\ \phi_y \\ \phi_z \end{pmatrix} \rightarrow \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ . \\ . \\ \theta_N \end{pmatrix} \tag{6.42}$$

Für die Rückwärtstransformation müssen entweder elementargeometrisch alle Winkel und Strecken aus den geometrischen Zusammenhängen entnommen werden. Das wird extrem schwer, wenn es sich um 5 oder 6 Rotationsgelenke handelt. In diesem Fall wechselt man auf die Matrizenmethode und formuliert die kinematische Kette als Matrizenprodukt:

$$^0T_N = A_1 A_2 A_3 \cdots A_N$$

Die Auflösung dieser Gleichung nach den Gelenkkoordinaten ist ein inverses Probleme und macht häufig erhebliche Probleme: Durch jedes Rotationsgelenk entstehen auf der rechten Seite Terme, die gemischte Produkte von Sinus-

und Cosinusfunktionen enthalten, mit den gesuchten Gelenkwinkeln als Argumente. Die Auflösung nach diesen Argumenten ist schwierig. Oft muss mit inversen DH-Matrizen multipliziert werden, um weiterzukommen. Selbst bei dem einfachen Beispiel mit nur drei Achsen in Abschn. 6.4.4 ist die Auflösung nach den Gelenkwinkeln nicht einfach. Ein erstes Beispiel für diese Probleme haben wir bei der Bestimmung der Orientierung kennengelernt (s. auch Gl. 6.18), die ja ebenfalls ein inverses Problem darstellt.

Mehrdeutigkeiten

Es existieren oft mehrere Lösungen, von denen nicht alle einer realen Armstellung entsprechen. In manchen Fällen entspricht das Auftreten mehrere Lösungen aber auch der geometrischen Realität. Zum Beispiel kann ein SCARA-Roboter einen Raumpunkt immer in zwei Konfigurationen erreichen, nämlich mit dem Ellenbogen nach links und nach rechts gedreht. Auch unser TR-Roboter weist Mehrdeutigkeit auf, in Abb.6.19 ist gezeigt, wie der Roboter eine Sollposition in zwei Konfigurationen anfahren kann.

Singularitäten

Singularitäten sind spezielle Situationen, in denen die mathematische Ermittlung der Gelenkwinkel mit vorgegebener Geschwindigkeit des TCP nicht möglich ist. Sie werden im Abschnitt 6.5 besprochen.

Wir geben hier zwei Beispiele an. Zunächst soll elementargeometrisch die Rückwärtstransformation für den TR-Roboter bestimmt werden. Im zweiten Beispiel wird die Rückwärtstransformation für den SCARA durchgeführt.

6.4.2. Elementargeometrische Rückwärtstransformation für den TR-Roboter

Aus Abschnitt 6.3.1 wissen wir schon:

$$\begin{aligned} y &= L\sin\theta \\ x &= d + L\cos\theta \end{aligned}$$

Daraus ergibt sich:

$$\theta = \arcsin y/L \tag{6.43}$$

$$d = x - L\cos\theta \tag{6.44}$$

Man erkennt selbst an diesem einfachen Beispiel, dass die Rückwärtstransformation mehr Probleme bereiten kann als die Vorwärtstransformation:

1. Die Bestimmung von θ ist mehrdeutig.
2. d kann erst berechnet werden, wenn θ schon berechnet ist.

Ein Beispiel: Der Roboter habe eine Armlänge von L, der TCP soll den Punkt (0, L/2) anfahren. Aus den Gleichungen 6.43,6.44 ergibt sich:

$$\theta = \arcsin(1/2),\ d = -L\cos\theta$$

Man erhält zwei Lösungen:
1. Lösung: $\theta = 30^o,\ d = -\frac{\sqrt{3}}{2}L = -0.866L$
2. Lösung: $\theta = 150^o,\ d = \frac{\sqrt{3}}{2}L = 0.866L$

Die beiden Lösungen sind in Abb.6.19 für eine Armlänge von L=10 dargestellt.

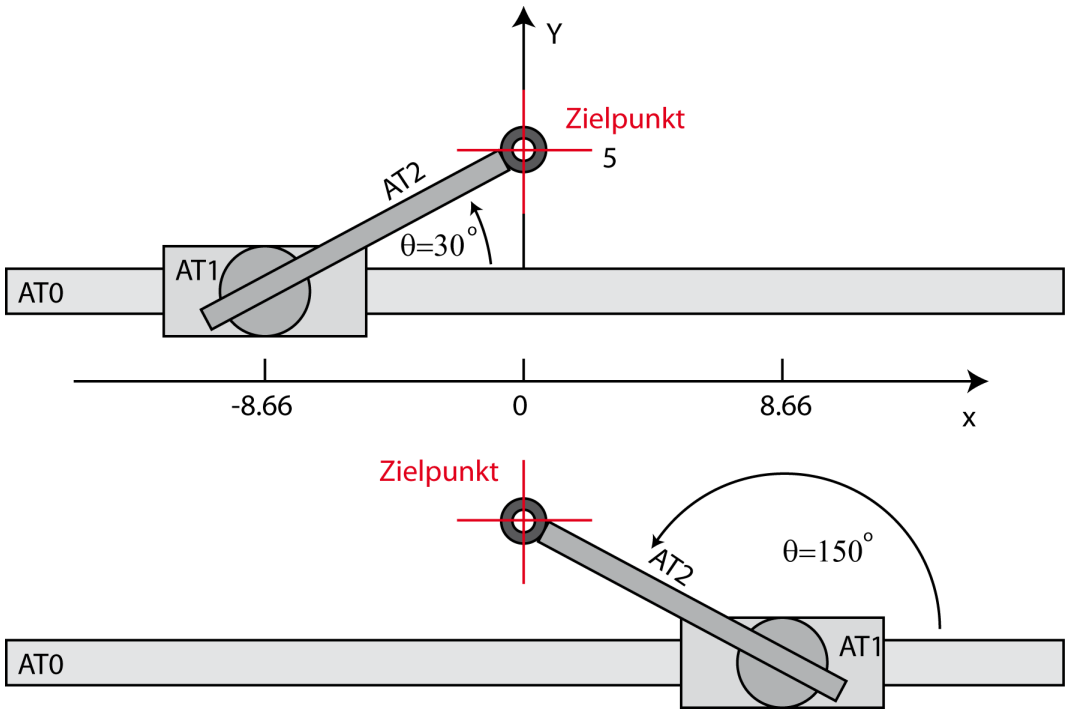


Abbildung 6.19.: Der zweiachsige TR-Roboter kann viele Punkte in zwei verschiedenen Konfigurationen anfahren. Hier der Punkt (0, 5) bei einer Armlänge von 10

Man bezeichnet diese beiden unterschiedlichen Möglichkeiten, den gleichen Raumpunkt zu erreichen, als *Konfigurationen*. In der Praxis ist es oft so, dass nur eine dieser Konfigurationen genutzt werden kann. Ein unerwünschter Wechsel in eine andere Konfiguration kann zu einem Crash führen, weshalb die Robotersteuerungen oft die Konfigurationen überwachen.

In Abschnitt 4.3 hatten wir eine Bewegung des TR-Roboters betrachtet, die vom Punkt (20,2) auf gerader Bahn zum Punkt (0,10) führte. Dabei hatte es den Anschein, als ob gegen Ende der Bewegung der Schlitten, das heißt die

d-Maschinenachse, ein kleines Stück zurück fahren muss. Dies ist in Abb.4.4 erkennbar. Das wollen wir nun genauer untersuchen. Wir geben dazu vor:

$$x(t) = x_s + (x_e - x_s)t \text{ für } 0 \leq t \leq 1 \quad (6.45)$$

$$y(t) = y_s + (y_e - y_s)t \text{ für } 0 \leq t \leq 1 \quad (6.46)$$

Damit ergibt sich eine gerade Bahn vom Startpunkt (x_s, y_s) zum Endpunkt (x_e, y_e). In der grafischen Darstellung sieht man, dass die kartesischen Koordinaten sich linear mit der Zeit ändern. (Abb.6.20) Welche Werte nehmen die

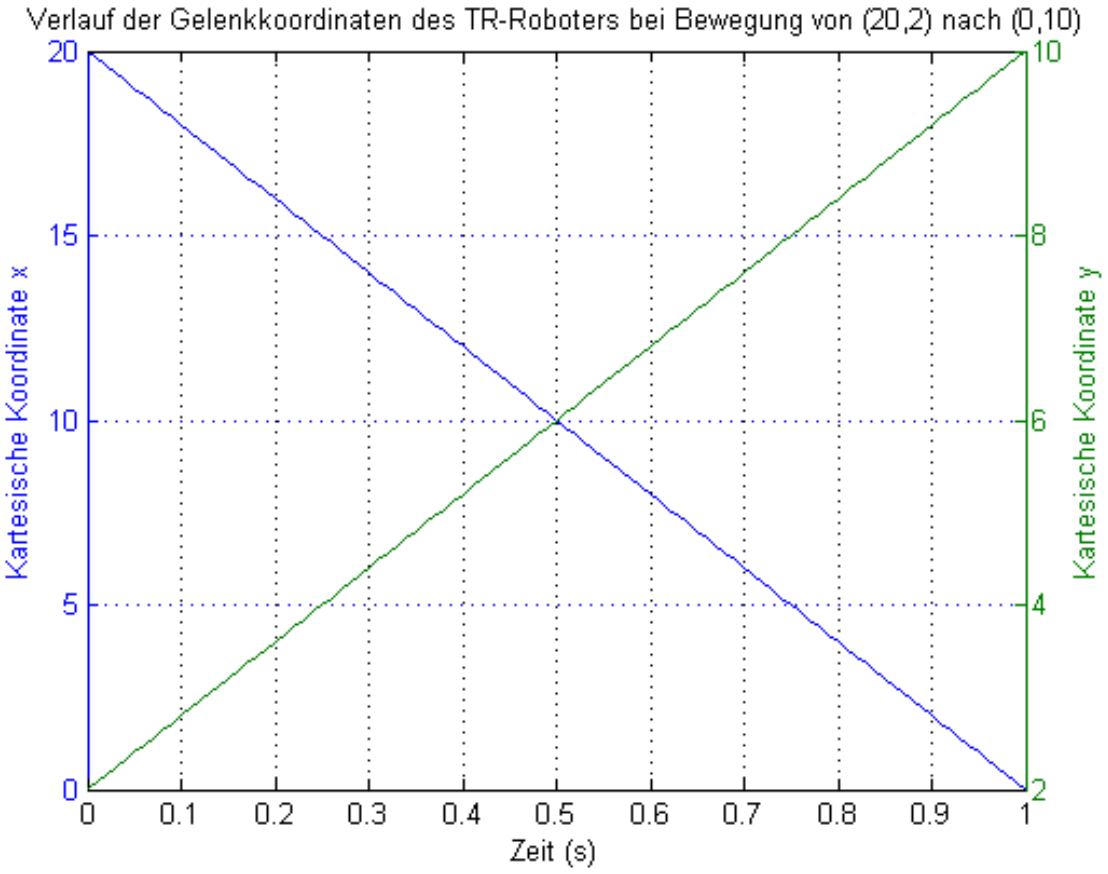


Abbildung 6.20.: Die kartesischen Koordinaten des TR-Roboters bei einer Bewegung von (20,2) nach (0,10).

Maschinenkoordinaten während der Bewegung an? Das kann man berechnen, indem man $x(t)$ und $y(t)$ in die Gleichungen der kinematischen Rückwärtstransformation einsetzt:

$$\theta(t) = \arcsin y(t)/L \quad (6.47)$$

$$d(t) = x(t) - L\cos\theta(t) \quad (6.48)$$

Das Ergebnis ist in Abb.6.21 dargestellt. Man sieht, dass tatsächlich die d-Achse in der letzten Phase rückwärts läuft. Dies ist nur ein Beispiel, und zwar eines, das an einer sehr einfachen Geometrie abgeleitet wurde. Bei einem 6-achsigen Roboter ergeben sich sehr komplexe Verlaufskurven für die Maschinenachsen, wenn im CP-Betrieb gefahren wird. Der interessanteste Fall ist

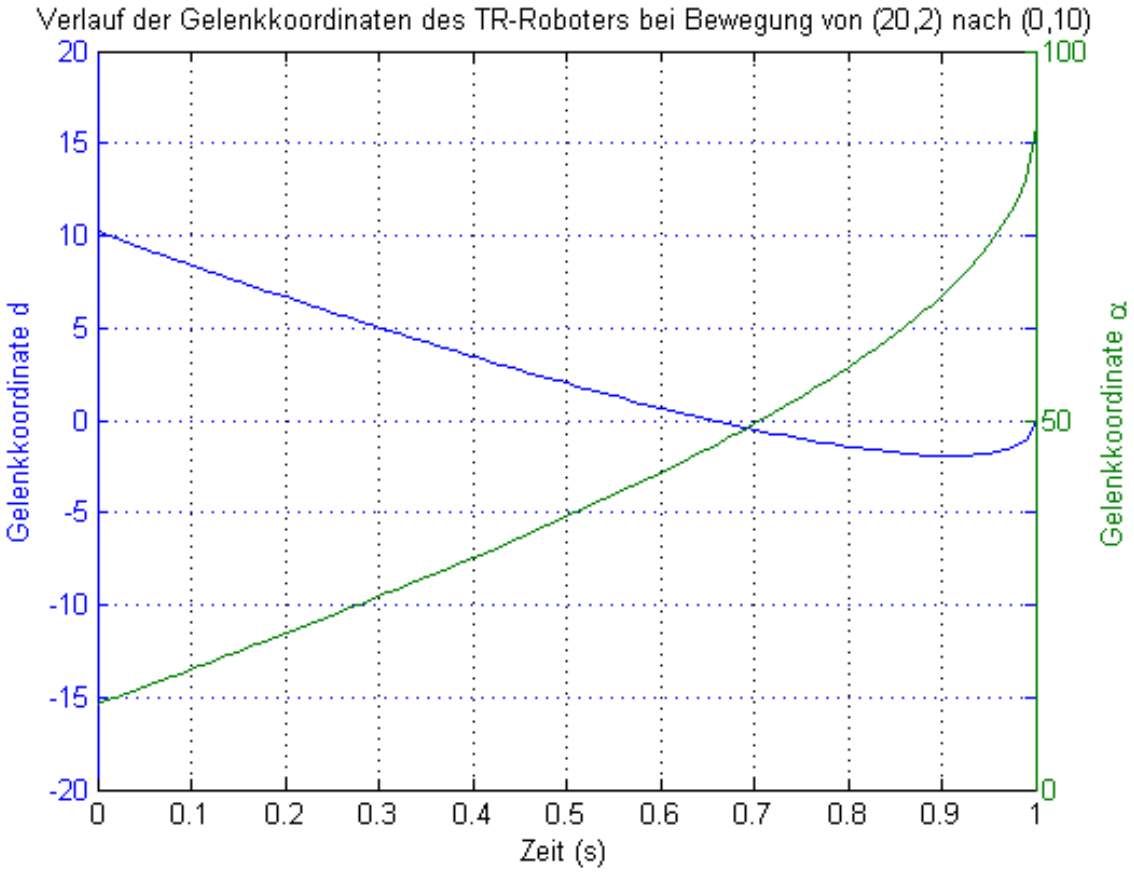


Abbildung 6.21.: Die Maschinenachsen des TR-Roboters während der Bewegung von (20,2) nach (0,10).

das Erreichen einer Singularität, worauf in Abschnitt 6.5 näher eingegangen wird.

6.4.3. Rückwärtstransformation für den TR-Roboter nach der Matrizenmethode

Die Rückwärtstransformation nach der Matrizenmethode läuft so ab wie die Vorwärtstransformation nach der Matrizenmethode, bis die Gesamttransformationsmatrix ${}^0T_N$ vorliegt. Dann versucht man Beziehungen zu gewinnen, mit denen man die Gelenkwinkel aus den kartesischen Koordinaten bestimmen kann. In diesem Fall geht man z. B. von den Gleichungen 6.35 aus und erhält:

$$\begin{aligned} L &= \frac{x}{\sin\theta} \\ d &= z - L\cos\theta \end{aligned}$$

Es haben sich also die gleichen Beziehungen wie bei der elementargeometrischen Methode ergeben, wenn man die veränderte Achsenlage berücksichtigt.

6.4.4. Die Rückwärtstransformation am Beispiel der Grundachsen des SCARA-Roboters

Wir wollen die Rückwärtstransformation am Beispiel der Grundachsen des SCARA-Roboters betrachten und gehen von Gl. 6.36 aus, wobei wir die die Elemente der Matrix wieder $\vec{n}, \vec{o}, \vec{a}, \vec{p}$ benennen ($s_2 = \sin\theta_2$ usw.):

$$
\begin{aligned}
{}^0T_N &= \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} c_2c_3 - s_2s_3 & -c_2s_3 - s_2c_3 & 0 & c_2c_3l_3 - s_2s_3l_3 + c_2l_2 \\ s_2c_3 - c_2s_3 & -s_2s_3 - c_2c_3 & 0 & s_2c_3l_3 - c_2s_3l_3 + s_2l_2 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
$$

In dieser Gleichung ist also ${}^0T_N$ d.h. $\vec{n}, \vec{o}, \vec{a}, \vec{p}$ aus der Aufgabenstellung heraus bekannt und θ_2, θ_3 sowie d_1 sind gesucht. Da keine Handachsen vorhanden sind, ist die durch ${}^0T_N$ repräsentierte Stellung des Armes nicht frei wählbar, sondern muss eine real mögliche Stellung sein. Da die Matrizen elementweise gleichgesetzt werden können, entspricht diese Gleichung 16 einfachen, skalaren Gleichungen. Wir setzen zunächst die Elemente in der 3. Zeile, 4. Spalte gleich:

$$d_1 = p_z$$

Danach werden die Elemente in der 1. und 2. Zeile der letzten Spalte gleichgesetzt:

$$p_x = (c_2c_3 - s_2s_3)l_3 + c_2l_2 \tag{6.49}$$
$$p_y = (s_2c_3 + c_2s_3)l_3 + s_2l_2 \tag{6.50}$$

Nach einigen Umformungen ergibt sich

$$p_x^2 + p_y^2 = l_2^2 + l_3^2 + 2l_2l_3c_3$$

Diese Gleichung kann nach c_3 aufgelöst werden:

$$c_3 = \frac{p_x^2 + p_y^2 - l_2^2 - l_3^2}{2l_2l_3}$$

Wenn nun $c_3 = \cos\theta_3$ brechnet ist, können wir daraus θ_3 bestimmen. Statt direkt $\arccos$ zu benutzen verwenden wir die im praktischen Gebrauch bessere $\arctan$-Funktion:

$$\theta_3 = \arctan\left(\frac{\sqrt{1-c_3^2}}{c_3}\right)$$

Da nun θ_3 berechnet ist, können bei weiteren Berechnungen s_3 und c_3 als bekannt voraus gesetzt werden. Wir können uns Gl. 6.49 und 6.50 noch einmal vornehmen. Dort sind jetzt nur noch s_2 und c_2 unbekannt. Wir können z.B. c_2 eliminieren und nach s_2 auflösen (Übungsaufgabe!). Es ergibt sich:

$$s_2 = \frac{p_y(c_3 l_3 + l_2) - p_x s_3 l_3}{2c_3 l_2 l_3 + l_2^2 + l_3^2}$$

Daraus ergibt sich wiederum:

$$\theta_2 = \arctan\left(\frac{s_2}{\sqrt{1-s_2^2}}\right)$$

Zahlenbeispiel Es sei $l_2 = l_3 = 1$ und

$${}^0T_N = \begin{pmatrix} 1 & 0 & 0 & \sqrt{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ergebnis der Rückwärtstransformation: $\theta_3 = \pm 90^o, \theta_2 = \mp 45^o$ Das Ergebnis spiegelt die beiden tatsächlich möglichen Stellungen wider (Abb. 6.22).

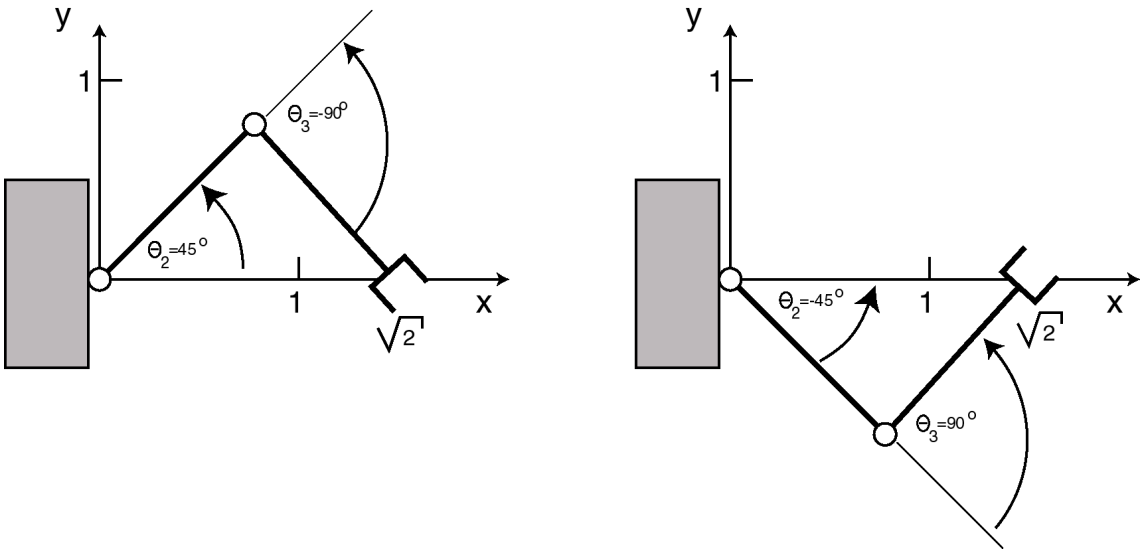


Abbildung 6.22.: Die beiden Stellungen, die sich im Zahlenbeispiel aus der Rückwärtstransformation ergeben haben.

Die Rückwärtstransformation mit 6 Achsen ist in der Regel noch sehr viel komplizierter.

6.5. Geschwindigkeiten, Singularitäten und statische Kräfte

6.5.1. Die Berechnung der Translationsgeschwindigkeit des TCP

Alle bisherigen Betrachtungen waren statisch, das heißt es wurde nur eine unveränderliche Stellung des Roboters betrachtet. Im praktischen Betrieb wird mehr verlangt. Wie im Abschnitt über Bahnsteuerung schon ausgeführt wurde, muss eine Robotersteuerung auch die Geschwindigkeit des TCP steuern. Praktisch alle Industrieroboter besitzen in den Bewegungsbefehlen einen Parameter für die Geschwindigkeit. Außerdem wird zur Realisierung von sanften ruckfreien Bewegungen ja ein Geschwindigkeitsprofil eingehalten. Für die CP-Bewegungen muss außerdem auch die Rotationsgeschwindigkeit des Tools gesteuert werden. Wie hängt nun die Geschwindigkeit des TCP von den Achskoordinaten ab?

Die Geschwindigkeit des TCP wird beschrieben durch einen dreidimensionalen Geschwindigkeitsvektor

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{6.51}$$

Jede Geschwindigkeitskomponente ist die Ableitung der zugehörigen Raumkoordinate nach der Zeit. Ableitungen nach der Zeit kennzeichnet man auch durch einen Punkt:

$$v_x = \frac{dx}{dt} = \dot{x} \tag{6.52}$$

$$v_y = \frac{dy}{dt} = \dot{y}$$

$$v_z = \frac{dz}{dt} = \dot{z} \tag{6.53}$$

Den Geschwindigkeitsvektor der Translation kann man also ausdrücken als

$$\vec{v} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \tag{6.54}$$

Ebenso kann man die Rotationsgeschwindigkeit des Tools durch die Ableitung der Orientierungswinkel beschreiben:

$$\vec{v_\phi} = \begin{pmatrix} \dot{\phi_x} \\ \dot{\phi_y} \\ \dot{\phi_z} \end{pmatrix} \tag{6.55}$$

Die verallgemeinerte Geschwindigkeit enthält sowohl den Translations- als auch den Rotationsanteil, sie kann einheitlich durch einen Vektor mit 6 Elementen beschrieben werden:

$$\vec{v_v} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{pmatrix} \tag{6.56}$$

Wir müssen nun den Zusammenhang dieser Geschwindigkeit mit den Gelenkkoordinaten herleiten. Die kartesischen Koordinaten hängen von den Gelenkwinkeln ab und diese Abhängigkeit wird durch die kinematische Vorwärtstransformation beschrieben (Kap. 6.3). Ein Beispiel dafür sind die Gleichungen 6.36. (Dort wurde statt x, y, z für die kartesischen Koordinaten p_x, p_y und p_z verwendet.)

Nehmen wir als Beispiel die erste Komponente dieses Geschwindigkeitsvektors $\dot{x} = \frac{dx}{dt}$. Die Koordinate x ist also eine Funktion der N Gelenkwinkel θ_i:

$$x = f_1(\theta_1, \theta_2 \cdots \theta_N) \tag{6.57}$$

Die Kettenregel gibt an, wie die Ableitung einer Größe zu berechnen ist, die selbst von mehreren anderen Größen abhängt. Hier ist es die Größe f_1, die von N Gelenkwinkeln abhängt und mit der Kettenregel erhalten wir:

$$\dot{x} = \frac{dx}{dt} = \frac{df_1}{dt} = \frac{\partial f_1}{\partial \theta_1}\frac{d\theta_1}{dt} + \frac{\partial f_1}{\partial \theta_2}\frac{d\theta_2}{dt} + \cdots + \frac{\partial f_1}{\partial \theta_N}\frac{d\theta_N}{dt} \tag{6.58}$$

Die Geschwindigkeit in x-Richtung ergibt sich also aus N Beiträgen, von jeder Achse ein Beitrag. Zum Beispiel ist

$$\frac{\partial f_1}{\partial \theta_2}\frac{d\theta_2}{dt}$$

der Beitrag zur Bewegung in x-Richtung, der von Achse 2 herrührt. Dabei ist

$$\frac{\partial f_1}{\partial \theta_2} \tag{6.59}$$

die *partielle Ableitung* von f_1 nach dem Gelenkwinkel θ_2. Sie beschreibt also, wie stark sich f_1 mit der Variablen θ_2 ändert, wenn alle anderen Variablen festgehalten werden. Diese partielle Ableitung drückt also aus, wie stark die kartesische Koordinate x, beschrieben durch die Funktion f_1 gerade von dieser

Gelenkkoordinate abhängt. Diese partielle Ableitung ist stark von der momentanen Roboterstellung abhängig. Dagegen ist

$$\frac{d\theta_2}{dt} \tag{6.60}$$

die Änderungsgeschwindigkeit der Gelenkkoordinate θ_2. Bei einem Lineargelenk ist das eine konventionelle Geschwindigkeit, die man z.B. in mm/s messen könnte. Bei einem Rotationsgelenk ist das eine Dreh- oder Winkelgeschwindigkeit. Um konkrete Beispiele zu rechnen muss diese Winkelgeschwindigkeit im Bogenmaß angegeben werden, also rad/s.

Wir können Gleichung 6.58 symbolisch mit dt multiplizieren und erhalten dann

$$dx = \frac{\partial f_1}{\partial \theta_1} d\theta_1 + \frac{\partial f_1}{\partial \theta_2} d\theta_2 + \cdots + \frac{\partial f_1}{\partial \theta_N} d\theta_N \tag{6.61}$$

Gleichungen dieser Art werden auch als totales Differential bezeichnet. Hier ist mit dx ein kleines Wegstück gemeint und mit $d\theta_i$ kleine Winkeländerungen. „Klein“ bedeutet infinitesimal klein, d.h. die Veränderungen sind so klein, dass sich die Stellung des Roboters nicht wesentlich ändert. Dadurch kann man die partiellen Ableitungen auf diesem kleinen Wegstück als konstant annehmen. Gleichung 6.61 beschreibt, wie sich ein kleines Wegstück besteht aus den Beiträgen der N Gelenke zusammensetzt. Man sieht, dass der Roboter die Bewegung der 6 Motoren ganz unterschiedlich in Bewegung umsetzt, er ist ein „6-dimensionales Getriebe“.

Die Koordinate y wird entsprechend durch eine Funktion f_2 beschrieben:

$$y = f_2(\theta_1, \theta_2 \cdots \theta_N)$$

und z durch f_3:

$$z = f_3(\theta_1, \theta_2 \cdots \theta_N)$$

Die Geschwindigkeitskomponenten von y und z sind:

$$\dot{y} = \frac{dy}{dt} = \frac{df_2}{dt} = \frac{\partial f_2}{\partial \theta_1}\frac{d\theta_1}{dt} + \frac{\partial f_2}{\partial \theta_2}\frac{d\theta_2}{dt} + \cdots + \frac{\partial f_2}{\partial \theta_N}\frac{d\theta_N}{dt} \tag{6.62}$$

$$\dot{z} = \frac{dz}{dt} = \frac{df_3}{dt} = \frac{\partial f_3}{\partial \theta_1}\frac{d\theta_1}{dt} + \frac{\partial f_3}{\partial \theta_2}\frac{d\theta_2}{dt} + \cdots + \frac{\partial f_3}{\partial \theta_N}\frac{d\theta_N}{dt} \tag{6.63}$$

Man kann nun formal die Gleichungen 6.58, 6.62 und 6.63 in eine Matrixgleichung zusammenfassen mit der die Translationsgeschwindigkeit v_T berechnet wird:

$$\vec{v}_T = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots & \frac{\partial f_1}{\partial \theta_N} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \cdots & \frac{\partial f_2}{\partial \theta_N} \\ \frac{\partial f_3}{\partial \theta_1} & \frac{\partial f_3}{\partial \theta_2} & \cdots & \frac{\partial f_3}{\partial \theta_N} \end{pmatrix} \begin{pmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \vdots \\ \frac{d\theta_N}{dt} \end{pmatrix} = J_T \dot{\vec{\theta}} \tag{6.64}$$

Dabei ist

$$\dot{\vec{\theta}} = \begin{pmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \vdots \\ \frac{d\theta_N}{dt} \end{pmatrix}$$

ein Vektor, der die Änderungsgeschwindigkeiten der N Gelenke enthält. Die Matrix J_T enthält die partiellen Ableitungen der kartesischen Koordinaten x, y, z nach den Gelenkkoordinaten θ_i.

6.5.2. Rotationsgeschwindigkeit und Jacobi-Matrix

Die gleichen Überlegungen gelten für die Orientierung des Tools bzw. der Roboterhand. Die drei Orientierungswinkel sind in ähnlicher Weise von den Gelenkwinkeln abhängig:

$$\phi_x = f_4(\theta_1, \theta_2 \cdots \theta_N) \tag{6.65}$$
$$\phi_y = f_5(\theta_1, \theta_2 \cdots \theta_N) \tag{6.66}$$
$$\phi_z = f_6(\theta_1, \theta_2 \cdots \theta_N) \tag{6.67}$$

Für die Rotationsgeschwindigkeiten der Hand ergibt sich auf die gleiche Art:

$$\dot{\vec{v}}_R = \begin{pmatrix} \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{pmatrix} = \begin{pmatrix} \frac{\partial f_4}{\partial \theta_1} & \frac{\partial f_4}{\partial \theta_2} & \cdots & \frac{\partial f_4}{\partial \theta_N} \\ \frac{\partial f_5}{\partial \theta_1} & \frac{\partial f_5}{\partial \theta_2} & \cdots & \frac{\partial f_5}{\partial \theta_N} \\ \frac{\partial f_6}{\partial \theta_1} & \frac{\partial f_6}{\partial \theta_2} & \cdots & \frac{\partial f_6}{\partial \theta_N} \end{pmatrix} \begin{pmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \vdots \\ \frac{d\theta_N}{dt} \end{pmatrix} = J_R \dot{\vec{\theta}} \tag{6.68}$$

Wenn man die Gleichungen 6.64 und 6.68 zusammenfasst erhält man einen Ausdruck für die verallgemeinerte Geschwindigkeit $\vec{v}_v$ in der die Translations- und die Rotationsgeschwindigkeit enthalten sind. Die entstehende Matrix ist die *Jacobi-Matrix* J_A:

$$\vec{v}_v = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots & \frac{\partial f_1}{\partial \theta_N} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \cdots & \frac{\partial f_2}{\partial \theta_N} \\ \frac{\partial f_3}{\partial \theta_1} & \frac{\partial f_3}{\partial \theta_2} & \cdots & \frac{\partial f_3}{\partial \theta_N} \\ \frac{\partial f_4}{\partial \theta_1} & \frac{\partial f_4}{\partial \theta_2} & \cdots & \frac{\partial f_4}{\partial \theta_N} \\ \frac{\partial f_5}{\partial \theta_1} & \frac{\partial f_5}{\partial \theta_2} & \cdots & \frac{\partial f_5}{\partial \theta_N} \\ \frac{\partial f_6}{\partial \theta_1} & \frac{\partial f_6}{\partial \theta_2} & \cdots & \frac{\partial f_6}{\partial \theta_N} \end{pmatrix} \begin{pmatrix} \frac{d\theta_1}{dt} \\ \frac{d\theta_2}{dt} \\ \vdots \\ \frac{d\theta_N}{dt} \end{pmatrix} = J_A \dot{\vec{\theta}} \tag{6.69}$$

Die Jacobi-Matrix ist eine der Schlüsselgrößen in der Robotik. Sie erlaubt nicht nur die Berechnung der Geschwindigkeit sondern auch die der wirkenden Kräfte. Außerdem spielt sie die zentrale Rolle bei der Bestimmung von Singularitäten.

Die Jacobi-Matrix wird von der Robotersteuerung ständig genutzt, um die gewünschte Sollgeschwindigkeit einzustellen. (Dazu zählen auch die Geschwindigkeiten während der Geschwindigkeitsprofile beim Beschleunigen und Abbremsen.) Bei CP-Bewegungen unterliegt ja auch die Rotationsgeschwindigkeit einer exakten Steuerung. Zur Realisierung einer Sollgeschwindigkeit, wird daher in Gleichung 6.69 die linke Seite $\vec{v}_v$ als Sollgeschwindigkeit vorgegeben. Dadurch entsteht ein lineares Gleichungssystem, in dem der Vektor mit den Gelenkgeschwindigkeiten $\dot{\vec{\theta}}$ auf der rechten Seite bestimmt wird:

$$\vec{v}_{Soll} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{pmatrix} = J_A \dot{\vec{\theta}} \tag{6.70}$$

Beispiel: Bestimmung der Gelenkgeschwindigkeiten am TR-Roboter

Die Vorwärtstransformation ergab für unseren zweiachsigen TR-Roboter:

$$x = L \sin\theta \tag{6.71}$$
$$z = a + L \cos\theta \tag{6.72}$$

Dabei sind die Gelenkkoordinaten $\theta_1 = a, \theta_2 = \theta$ und die funktionalen Abhängigkeiten $x = f_1(a, \theta) = L \sin\theta$ sowie $z = f_2(a, \theta) = a + L \cos\theta$ Die Jacobi-Matrix ist:

$$J_A = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial L \sin\theta}{\partial a} & \frac{\partial L \sin\theta}{\partial \theta} \\ \frac{\partial (a + L\cos\theta)}{\partial a} & \frac{\partial (a + L\cos\theta)}{\partial \theta} \end{pmatrix} = \begin{pmatrix} 0 & L\cos\theta \\ 1 & -L\sin\theta \end{pmatrix} \tag{6.73}$$

6.5.3. Singularitäten

Wir betrachten noch einmal das Gleichungssystem 6.70. Ein solches Gleichungssystem ist nicht unter allen Umständen lösbar. Die Koeffizienten der Jacobi-Matrix hängen ja stark von der momentanen Position ab und ändern sich während der Bewegung ständig. Wenn nun in einer bestimmten Positon zwei der Gleichungen linear abhängig sind (eine ist ein Vielfaches einer anderen), dann enthält diese Matrix nicht mehr genügend Information, um alle

Unbekannten (die Gelenkgeschwindigkeiten) zu bestimmen. Man sagt auch, die Matrix hat einen Rangverlust erlitten.

Man kann das formal daran feststellen, dass die Determinante der Jacobi-Matrix Null wird: $det(J_A) = 0$ Eine Matrix deren Determinante Null ist heißt auch *singulär*. Eine Roboterstellung in der die Jacobi-Matrix singulär ist, heißt deshalb *Singularität*. Da das Gleichungssystem 6.70 in der Singularität nicht mehr lösbar ist, stellt diese Position ein Problem für die Steuerung dar.

Die Determinante ist z.B. dann Null, wenn eine ganze Zeile der Jacobi-Matrix Null wird. Anschaulich bedeutet das, die zugehörige Komponente des kartesischen Geschwindigkeitsvektors $\vec{v}_v$ zwangsweise Null wird und hier keine Sollgeschwindigkeit mehr eingehalten werden kann. Nehmen wir z.B. an, dass in Gleichung 6.70 die zweite Zeile der Jacobi-Matrix Null wird. Das wäre gleichbedeutend damit, dass in Gl.6.62 alle partiellen Ableitungen gleich Null sind. Dann wäre immer $v_y = \dot{y} = 0$, auch wenn alle Gelenkachsen mit Höchstgeschwindigkeit arbeiten. Offenbar ist hier ein Freiheitsgrad verloren gegangen, der TCP kann nicht mehr in y-Richtung bewegt werden. Wenn man die Singularität noch nicht ganz erreicht hat, aber sich schon genähert hat, ist die zweite Zeile der Jacobi-Matrix noch nicht Null, aber die Beiträge der partiellen Ableitungen sind schon sehr klein. Das bedeutet: Es werden hohe Gelenkgeschwindigkeiten gebraucht, um weiterhin eine gewünschte Sollgeschwindigkeit in y-Richtung zu fahren. Das lässt sich tatsächlich beobachten: Bei Annäherung an eine Singularität, laufen die Gelenkachsen auffällig schnell. Nähert man sich dann weiter an, ist die Sollgeschwindigkeit in einer Richtung nicht mehr zu halten, weil die Gelenkgeschwindigkeiten begrenzt sind. Dann bricht die Steuerung evtl. die Bewegung mit einer Fehlermeldung ab. Dazu muss man also nicht die Singularität exakt anfahren, sondern ihr nur nah genug kommen. (siehe Abb. 6.23)

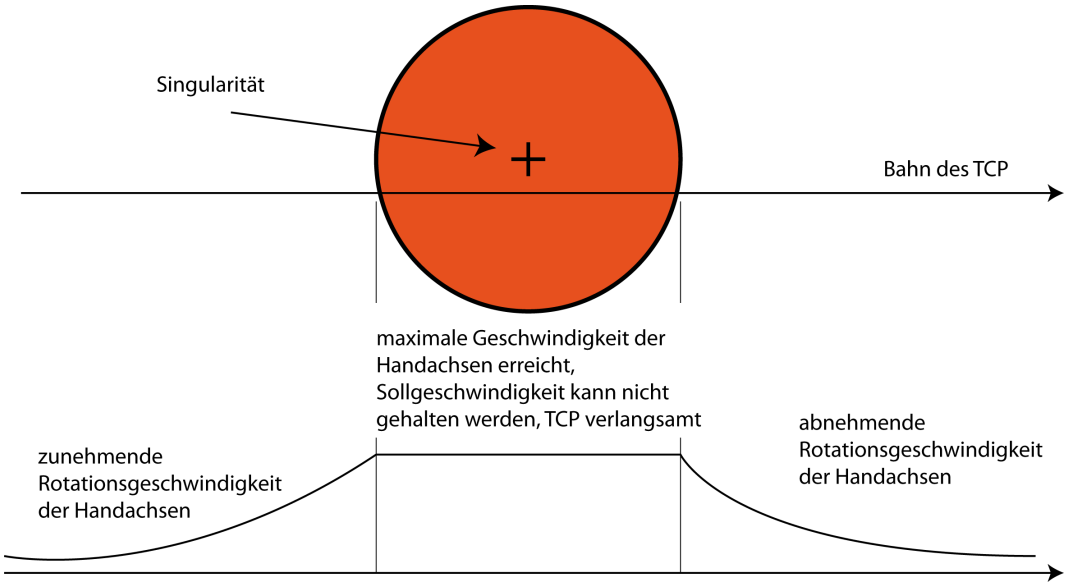


Abbildung 6.23.: Bewegung des TCP in der Nähe einer Singularität

Singularitäten entsprechen besonderen Stellungen des Roboterarmes, nämlich solchen, in denen ein Freiheitsgrad verloren gegangen ist. [1], [3], [4] Man unterscheidet Randsingularitäten und Innere Singularitäten.

Randsingularitäten

Alle Roboter haben am Rande Ihres Arbeitsbereiches Singularitäten, weil durch die völlig Streckung des Armes die Bewegungsmöglichkeiten eingeschränkt sind. In Abb. 6.24 sind unter b) und c) Randsingularitäten gezeigt. In beiden Fällen ist es in einer lokalen Umgebung dieser Stellung nicht möglich, den Effektor radial (dR) zu bewegen, ein Freiheitsgrad ist also verloren gegangen. Abgesehen von der Singularität gibt es hier meist auch Einschränkungen in der Orientierung.

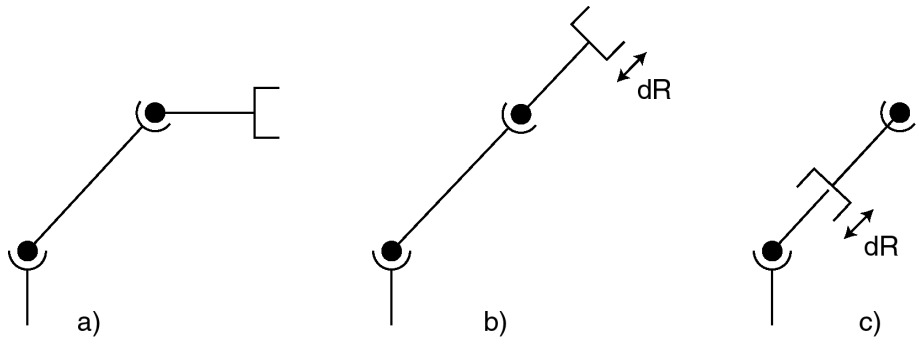


Abbildung 6.24.: a) Nicht-singuläre Stellung, b) Randsingularität am äußeren Rand des Arbeitsraumes und c) Randsingularität am inneren Rand des Arbeistraumes.

Singularitäten am TR-Roboter

Auch unser TR-Beispielroboter weist Singularitäten auf. Eine erste singuläre Stellung ist in Abb.6.25 gezeigt. Beide Maschinenachsen bewirken in unmittelbarer Nähe der Singularität, dass der TCP sich horizontal bewegt. Eine vertikale Bewegung des TCP ist in dieser singulären Stellung nicht möglich, es ist $v_y = 0$. Der Freiheitsgrad in y-Richtung ist also verloren gegangen. Um den TCP in y-Richtung zu bewegen, muss der Roboter zunächst die singuläre Stellung verlassen.

Wir berechnen nun mit Hilfe der Jakobi-Matrix alle singulären Stellungen des TR-Roboters. Eine Singularität liegt dann vor, wenn die Determinante der Jakobi-Matrix 0 ist. Dazu müssen alle Elemente einer Zeile oder einer Spalte in Gl.6.73 gleich 0 sein. Diese Möglichkeit besteht nur für die erste Zeile, wenn gilt:

$$(0, L\cos\theta) = (0, 0)$$

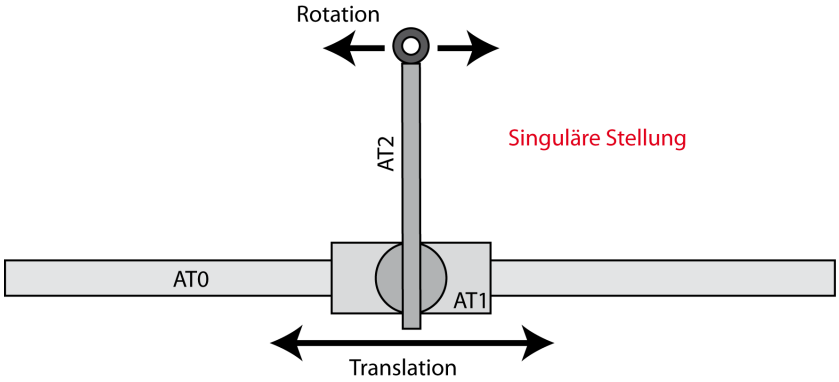


Abbildung 6.25.: Wenn der TR-Roboter sich in dieser Stellung befindet, bewirken beide Achsantriebe nur eine Bewegung des TCP in x-Richtung. Eine Bewegung in y-Richtung (nach oben oder unten) ist nicht möglich, dieser Freiheitsgrad ist verloren gegangen. Der Roboter befindet sich in einer Singularität.

und damit $v_x = 0$. Da $L > 0$ lautet unsere Bedingung

$$\cos\theta = 0$$

Daraus ergeben sich singuläre Stellungen unter der Bedingung:

$$\theta = \pm 90^o$$

Diese Stellungen sind in Abb.6.26 gezeigt.

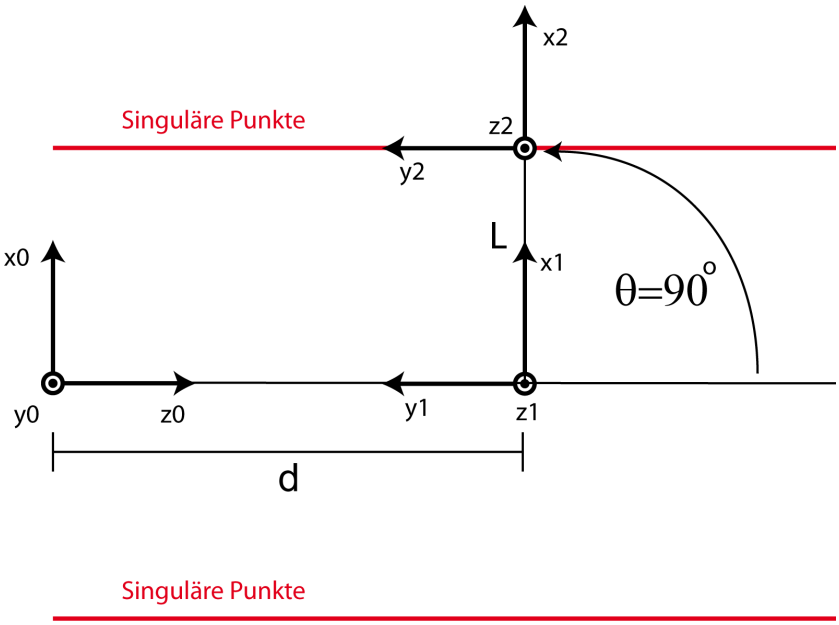


Abbildung 6.26.: Alle Punkte mit $\gamma = \pm 90^o$ sind beim TR-Roboter Singularitäten. Diese Punkte liegen am Rande des Arbeitsraumes.

Innere Singularitäten

Sie treten im Inneren des Arbeitsraumes auf und sind in der praktischen Arbeit mit Robotern das eigentliche Problem. Innere Singularitäten liegen dann vor, wenn mehrere Achsen wirkungsgleich sind. Bei Linearachsen reicht dazu eine Parallelität der Achsen aus. Rotationsachsen sind wirkungsgleich, wenn sie kollinear liegen. In diesem Fall kann die Drehung einer Achse durch die Gegendrehung einer anderen Achse vollständig kompensiert werden. Dies entspricht dem Verlust eines Freiheitsgrades, da ja für eine Drehachse zwei Gelenke verwendet werden. In Abb. 6.27 ist eine innere Singularität dargestellt. Außerdem besteht hier bei der Rückwärtstransformation ein Problem: Es existieren unendlich viele Lösungen für die aktuelle Handstellung.

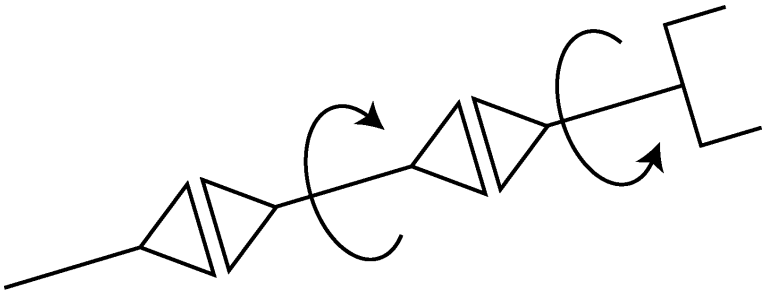


Abbildung 6.27.: Eine innere Singularität liegt z.B. vor, wenn zwei Rotationsachsen kollinear (auf einer Geraden) liegen.

Singularitäten treten auch im praktischen Betrieb häufig auf und sind ein ernst zu nehmendes Problem. Manche Roboter bleiben beim Auftreten einer Bewegungssingularität einfach stehen. Oft bieten die Robotersprachen eine Lösung für die kritischen Bereiche an, z.B. werden in RAPID durch den Befehl `SingArea/Wrist` die Handachsen auf PTP-Steuerung umgeschaltet, womit die Singularität vermieden wird, falls eine der Achsen 4–6 beteiligt ist.

6.5.4. Statische Kräfte am Roboterarm

Wir betrachten auf der Wirkungsseite im kartesischen Raum einen verallgemeinerten Kraftvektor, der sowohl die Kräfte als auch die Drehmomente im/am TCP enthält:

$$\vec{F} = \begin{pmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{pmatrix} \tag{6.74}$$

Auf der Antriebseite haben wir an jeder Linearachse eine wirkende Kraft und an jeder Rotationsachse ein wirkendes Drehmoment. Um eine allgemeine Formulierung zu haben bezeichnen wir alle Gelenkkoordinaten mit θ und die dort

wirkenden Kräfte/Drehmomente mit F_θ. An den Gelenken müssen die Antriebe also folgenden Vektor aufbringen:

$$\vec{F_\theta} = \begin{pmatrix} F_{\theta_1} \\ F_{\theta_2} \\ \vdots \\ F_{\theta_N} \end{pmatrix} \tag{6.75}$$

Es gibt natürlich einen Zusammenhang zwischen den statischen Kräften/Momenten im Antrieb und den statischen Kräften/Momenten am TCP. Dieser Zusammenhang wird wiederum durch die Jacobi-Matrix hergestellt, hier durch die transponierte Jacobi-Matrix J^T.[3], [1]

$$\vec{F_\theta} = J^T \vec{F} \tag{6.76}$$

Diese Gleichung berücksichtigt keine Reibung. Ein Roboter verwandelt also einen Vektor von Kräften/Drehmomenten entsprechend seiner momentanen Stellung. Einen 6-achsigen Roboter kann man daher auch als „6-dimensionalen Hebel“ oder „6-dimensionalen Flaschenzug“ mit variablen Übersetzungsverhältnissen ansehen.

7. Mobile Roboter

7.1. Einführung

7.1.1. Abgrenzung von mobilen und stationären Robotern

Stationäre Roboter sind fest montiert und arbeiten in einem geometrisch begrenzten Arbeitsraum. Dazu zählen alle 5- und 6-achsigen Industrieroboter, die man z.B. aus der Automobilproduktion kennt.

Mobile Roboter sind auf Rädern, Ketten oder Beinen beweglich und haben damit prinzipiell einen unbegrenzten Arbeitsraum mit wechselnden Umweltbedingungen. Sie müssen ihre Antriebsenergie mitführen (Akku oder Benzin) oder während der Fahrt gewinnen (Sonne, verbrennen von gesammeltem Material)

7.1.2. Bezeichnungen für mobile Robotersysteme

Mobiler Roboter, MR Roboter irgendeiner Bauart, der sich fortbewegen kann

Autonomous Vehicle, AV Selbständiges Fahrzeug/Gerät

Autonomous Underwater Vehicle, AUV Selbständiges Unterwasser-Fahrzeug/Gerät

domestic robot Haushaltsroboter

driverless vehicle Fahrerloses Fahrzeug

humanoid robot, Humanoider Roboter menschenähnlicher Roboter, geht auf zwei Beinen

cognitiv robot Roboter der seine Umwelt mit Sensoren wahrnimmt und darauf reagiert

flying robot Fliegender Roboter

7.1.3. Motivation und Anwendungsbereiche

Bei vielen Tätigkeiten, z.B. im Dienstleistungsbereich wäre eine Automatisierung wünschenswert, diese ist aber erst durch Mobilität möglich. Anwendungsfelder der mobilen Robotik sind:

Transportaufgaben in der Produktion Alternative zu spurgeführten Wagen und rollenden Behältern; mobile Roboter kommen mit geringer oder ganz ohne Umgebungsmodifikation aus. Schon etabliert sind die führerlosen Transportsysteme (FTS), die zu den ersten mobilen Robotern im Routineeinsatz gehören; ihre Selbständigkeit ist allerdings begrenzt, weil Ziele und Routen fix sind.

Transportaufgaben in Krankenhäusern Transport von Medikamenten, Essen, Wäsche.

Transportaufgaben in Bürogebäuden Holen und bringen von Post und Dokumenten.

Reinigungsaufgaben Gehflächen, Scheiben, Spiegel in Solaranlagen, lohnt sich bei großen freien Flächen.

Garten Rasenmähen und Schnecken sammeln.

Führung von Menschen In Museen und öffentlichen Gebäuden.

Überwachungsaufgaben Bewachung von Räumen, Gebäuden, Fahrzeugen auf Ausstellungsflächen,

Erkundungsaufgaben In gefährlicher und unzugänglicher Umgebung, verstrahlte Bereiche, Kälte, Hitze, Nähe von Sprengstoffen, Weltall, Planetenerkundung

Unterwasseraufgaben AUVs, Inspektion von Bohrinseln, Tiefseeerforschung

Oft sind die Grenzen zum ferngesteuerten mobilen Gerät (Teleoperation) fließend, zum Beispiel beim Entschärfen von Bomben, wird das Gerät aus sicherer Entfernung gesteuert.

7.2. Unterschiede der mobilen Robotik zur stationären Robotik

In der stationären Robotik ist der Arbeitsraum und der Kollisionsraum exakt bekannt. Menschen können einfach vor dem Roboter geschützt werden durch Absperren der Arbeitszelle. Außerdem findet der stationäre Roboter meist eine wohlbekannte Umgebung vor, die zeitlich konstant ist, bis auf die Bewegungen von Werkstücken und Werkzeugen. (Ausnahme: Mehrere Roboter an

einem Werkstück gegenseitig innerhalb Kollisionsraum) Der stationäre Roboter bewegt sich deshalb auf fest programmierten Bahnen und braucht meist keine Sensoren.

Das alles gilt für den mobilen Roboter nicht mehr:

- Der Arbeitsraum eines mobilen Roboters kann eine Etage, ein Gebäude oder sogar die freie Natur sein. Diesen Arbeitsraum muss sich der mobile Roboter mit anderen mobilen Robotern oder Menschen teilen.
- Der Arbeitsraum ist ständigen Änderungen unterworfen und der mobile Roboter kann keine fest einprogrammierte Bahn abfahren.
- Der mobile Roboter muss selbständig seine Umwelt erfassen und daher über ausreichende Sensoren verfügen.
- Hat er seine Situation erkannt, muss er sich seine Bahn selbst planen.
- Der mobile Roboter muss mit Menschen oder anderen Robotern kommunizieren, wenn er ihnen begegnet. Es muss also irgendeine Art von Interface geben.
- Das Wichtigste: Bei jeder Bewegung muss er sicher gehen, keinen Schaden anzurichten!

7.2.1. Das Sicherheitsproblem in der mobilen Robotik

Die Asimovschen Robotergesetze

1. Ein Roboter darf keinen Menschen verletzen oder zulassen, dass einem Menschen Schaden zugefügt wird.

2. Ein Roboter muss den Befehlen eines Menschen gehorchen, außer, dies führt zu einer Verletzung des ersten Gesetzes.

3. Ein Roboter muss seine eigene Existenz schützen, außer dies führt zum Widerspruch mit einem der beiden anderen Gesetze.

In einem Betrieb gibt es weitergehende Forderungen, beispielsweise: Es dürfen keine Schäden an Sachen entstehen. Ein Roboter muss unterbrechungsfrei in kurzen Abständen seine Umwelt mit Sensoren abscannen, um eine Kollisionsvermeidung zu gewährleisten. (Task höchster Priorität und hoher Abtastrate) Dabei stellt sich die Frage sicherer Software und Algorithmen, Da alle Sensoren auch einen gewissen Anteil fehlerhafter Informationen liefern, werden hier sehr fehlertolerante Algorithmen gebraucht. Auch muss das Mikroprozessorsystem, die Sensoren, das Betriebssystem ausfallsicher und redundant sein. Eine Qualitätskontrolle und Zertifizierung ist notwendig.

7.2.2. Orientierung, Navigation und Routenplanung

Auch ein Mensch muss oft das Problem lösen, eine Route zu planen und zu navigieren. Die zentralen Fragen sind:

1. Wo befinde ich mich?
2. Wohin will ich?
3. Auf welchem Wege komme ich dorthin?

Das Problem der Orientierung stellt sich auf zwei Ebenen.

Globale Orientierung In welchem Abschnitt meines Arbeitsraumes bin ich, welches Zimmer der Etage, welche Etage des Hauses usw. Dazu gehört eine Vorinformation über den Aufbau und die Gliederung des globalen Arbeitsraumes, diese wird dem MR vor Arbeitsantritt einprogrammiert. Die globale Orientierung kann z.B. unterstützt werden, indem bei Wechsel von einem Segment in ein anderes durch RFID oder durch Barcodes eine Information übermittelt wird.

Lokale Orientierung Innerhalb meines Segmentes muss der MR wissen, wo genau ist seine momentane Position, wie ist seine momentane Ausrichtung (Drehwinkel). Damit das möglich ist, muss der MR Teile des Segmentes kennen und wiedererkennen Daraufhin kann er erkannte Teile der Umwelt einordnen und die Verbindungswege zu anderen Segmenten lokalisieren. Wichtig ist auch, in der unmittelbaren Umgebung die Passierbarkeit von Wegen zu beurteilen. Wege können eng sein und der MR muss wissen ob er durchfahren kann.

Für jede Aktivität des MR wird ein Ziel gebraucht; ist es nicht fest vorgegeben, muss er selbst Routinen zur Zielgenerierung nach übergeordneten Gesichtspunkten durchführen. Ist die globale und die lokale Orientierung vorhanden, kann mit einer Bahnplanung begonnen werden. dazu muss ein topologisches Modell des Arbeitsraumes vorliegen. Dieses Modell sagt aus, wie die Segmente zusammen hängen, also beispielsweise: „Zimmer 3 wird erreicht, indem man zunächst Zimmer 7 durchquert, dann in Flur 2 nach Süden fährt und am Flurende die rechte Tür nimmt.“. Ein aktuelles Forschungsgebiet ist daher das *map building* (kartographieren), dabei setzt sich der MR seine topologische Karte selbst zusammen, basierend auf den Ergebnissen seiner Sensoren. Diese Topologie kann sich ändern, wenn Z.B. eine Tür geschlossen wird oder ein Raum durch Gegenstände versperrt wird. Der MR sollte in der Lage sein, dies zu erkennen und seine topologische Karte anzupassen.

Für die anschließende Routenplanung, wird man mehrere Möglichkeiten gegenüber stellen und dann nach Optimierungskriterien entscheiden: Kürzester Weg, schnellster Weg, sicherster Weg ...

7.2.3. Sensorik und Sensordatenfusion

Als Sensoren kommen in Frage Kameras, Triangulationssensoren, Laserscanner, Ultraschall- und Infrarot-Entfernungsmessgeräte.

Interessant ist die Auswertung der Sensordaten. Da Sensordaten immer mit Fehlern behaftet sind, ist ihre Auswertung im Regelfall kompliziert. Man muss mit Messfehlern aller Art rechnen:

- Artefakte (völlige Falschwerte)
- Rauschen (unterschiedliche Werte bei mehrmaliger Messung der gleichen Größe)
- Fehler durch Bauteil-Toleranzen
- Systematische Messfehler durch Umgebungseinflüsse z.B. durch Störlicht.
- Als Folge: Widersprüchliche Messwerte

Die gewünschten Ergebnisse muss man schrittweise aus den primitiven Messwerten gewinnen, man nennt das auch *Sensordatenfusion*. Man wird eine Reduktion der Unsicherheiten durchführen und die Ergebnisse dann auf das nächst höhere Abstraktionsniveau heben. Dann erfolgt wieder eine Reduktion von Unsicherheiten und die Ermittlung des Ergebnisses auf der nächst höheren Ebene. Die (unzuverlässigen) Zwischenergebnisse kann man nur mit statistischen Methoden auswerten. Wichtige Hilfsmittel sind stochastische Methoden (Schätztheorie) und die Fehlerfortpflanzung. Die eigentliche verwertbaren Ergebnisse werden also stufenweise gewonnen und sind mit Unsicherheiten behaftet. Dazu ein Beispiel:

- Eine Kamera liefert helle und dunkle Punkte
- Die Grenzlinie zwischen den hellen und dunklen Punkten in der waagerechten wird bestimmt
- Probehalber wird eine Gerade in diese Punkte gelegt
- 60% der Punkte liegen innerhalb eines Streifens links und rechts der Geraden
- Die Breite des Streifens wird vergrößert
- Danach liegen 90% der Punkte auf der Geraden, die anderen werden als ungültig markiert
- Nun weiß man, dass die Linie mit x% Wahrscheinlichkeit eine Gerade ist
- Man wiederholt die Prozedur an den vertikalen Linien
- Es ergeben sich zwei weitere Linien.

- Mit einer Wahrscheinlichkeit von y % steht der mobile Roboter vor einer Tür.

Dabei können auch Daten aus mehreren Sensoren zusammengefügt werden um dann im Zusammenwirken ein (mehr oder weniger) schlüssiges Bild zu ergeben.

7.3. Programmiermodelle

Bei stationären Robotern wird überwiegend die sensorlose Programmierung angewendet. Sie ist nur möglich, weil Position und Orientierung aller Gegenstände im Arbeitsraum bekannt sind. Der Roboter bewegt sich auf Bahnen, die vorher festgelegt sind. Bei mobilen Robotern muss man andere Wege gehen.

7.3.1. Weltmodellierung nach der Sensordatenfusion

Dieses Programmiermodell kann eingesetzt werden, wenn die Sensordatenfusion gute Ergebnisse liefert. Aus den zusammengeführten Sensordaten ergibt sich ein möglichst vollständiges modellhaftes Bild der Umgebung, das *Weltmodell*. Die fusionierten Sensordaten werden evtl. mit eingespeicherten bekannten Strukturen verglichen. Wenn das Weltmodell aufgestellt ist, kann eine Planung der Aktion zur Lösung der gestellten Aufgabe erfolgen, z.B. eine Bahnplanung, die dann anschließend ausgeführt wird.

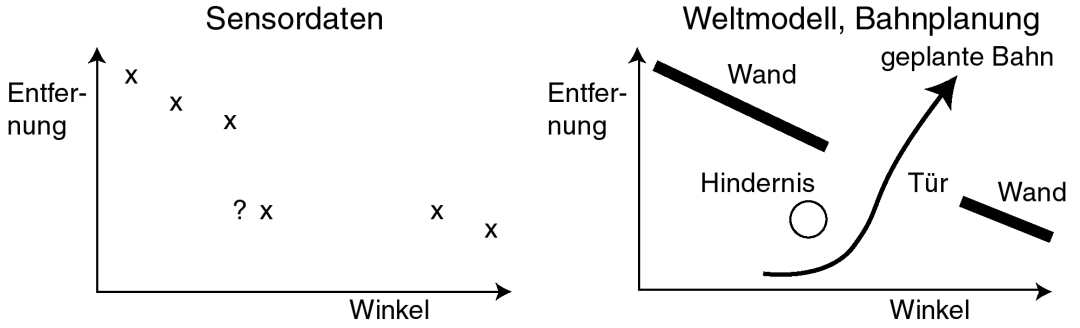


Abbildung 7.1.: Ein mobiler Roboter ist mit einem drehbaren Entfernungssensor ausgestattet. Links: Die gewonnenen Sensordaten. Rechts: Darauf basierend wird ein Weltmodell und eine Bahnplanung erstellt.

Das Ziel des Verfahrens ist, einen garantierten Erfolg der Aktion in mit einer planbaren Ausführungszeit zu haben. Ein Weg oder Zeitoptimierung ist möglich. Voraussetzungen sind allerdings eine hohe Rechenleistung des Steuerrechners, eine große Speicherkapazität, leistungsfähige Sensoren (z.B. Farbkameras) und ausgefeilte Algorithmen. Ein generelles Problem dieser Vorgehensweise ist, dass die vorausgeplante Bahn fehlerhaft werden kann, wenn

sich im Arbeitsbereich des Roboters etwas ändert. Das Verfahren der Sensordatenfusion mit Weltmodellerstellung kann sowohl für mobile Roboter als auch für stationäre Roboter eingesetzt werden.

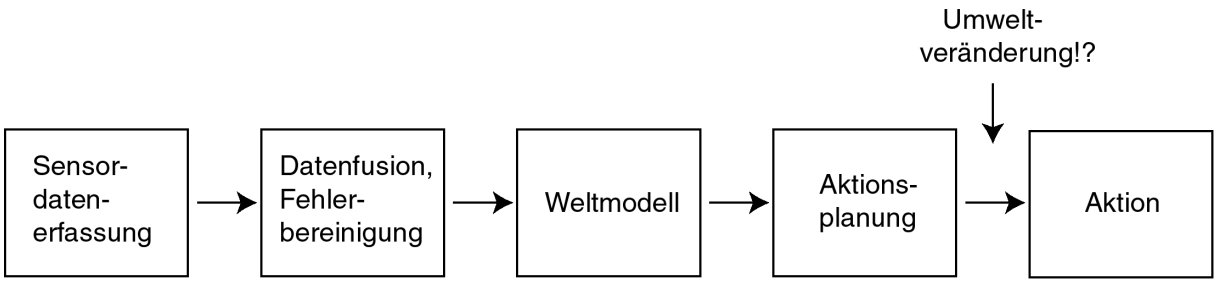


Abbildung 7.2.: Der Ablauf im Verfahren ist geradlinig und nicht zum Wirkungskreis geschlossen. Wenn sich nach der Sensordatenerfassung etwas in der Umwelt ändert, kann das nicht mehr berücksichtigt werden.

7.4. Verhaltensfusion (Subsumtionsansatz)

Diesen Weg kann man gehen, wenn die Sensoren nicht so leistungsfähig sind oder der Mikroprozessor nicht so leistungsfähig ist. Hier wird zu jedem Sensor ein Verhalten des mobilen Roboters festgelegt. Zum Beispiel sollte der Roboter abbremsen, wenn der vordere Entfernungssensor einen Gegenstand frontal vor dem Roboter meldet. Die Daten aller Sensoren werden nun im Betrieb kontinuierlich erfasst. Da nun mehrere Sensoren gleichzeitig Daten liefern, und zu jedem Sensor eine bestimmte Verhaltensweise festgelegt ist, könnten die geforderten Verhaltensweisen widersprüchlich sein. Hier muss nun eine eindeutige Vorrangregelung gemacht werden. Zum Beispiel muss das Ausweichen vor Hindernissen außer Kraft gesetzt werden, wenn der Roboter sich selbständig der Ladestation nähert. Das ganze Modell heißt auch Subsumtionsansatz und die Vorrangregeln Subsumtionsknoten. Ein Weltmodell gibt es hier nicht, der Roboter entscheidet nur „lokal“, was sein Verhalten manchmal etwas ziellos macht.

Vorteile:

- Anforderungen an Speicherplatz, Rechenleistung und Sensorik gering,
- gut parallelisierbar,
- einfach zu implementieren und zu erweitern,
- schnelle Reaktionen auf Veränderungen in Umgebung.

Nachteile:

- kein Überblick (Ameisenperspektive)
- Messfehler schwer zu erkennen,

- Umwege und Endlosschleifen möglich,
- Aufgabenumsetzung weniger zielgerichtet.

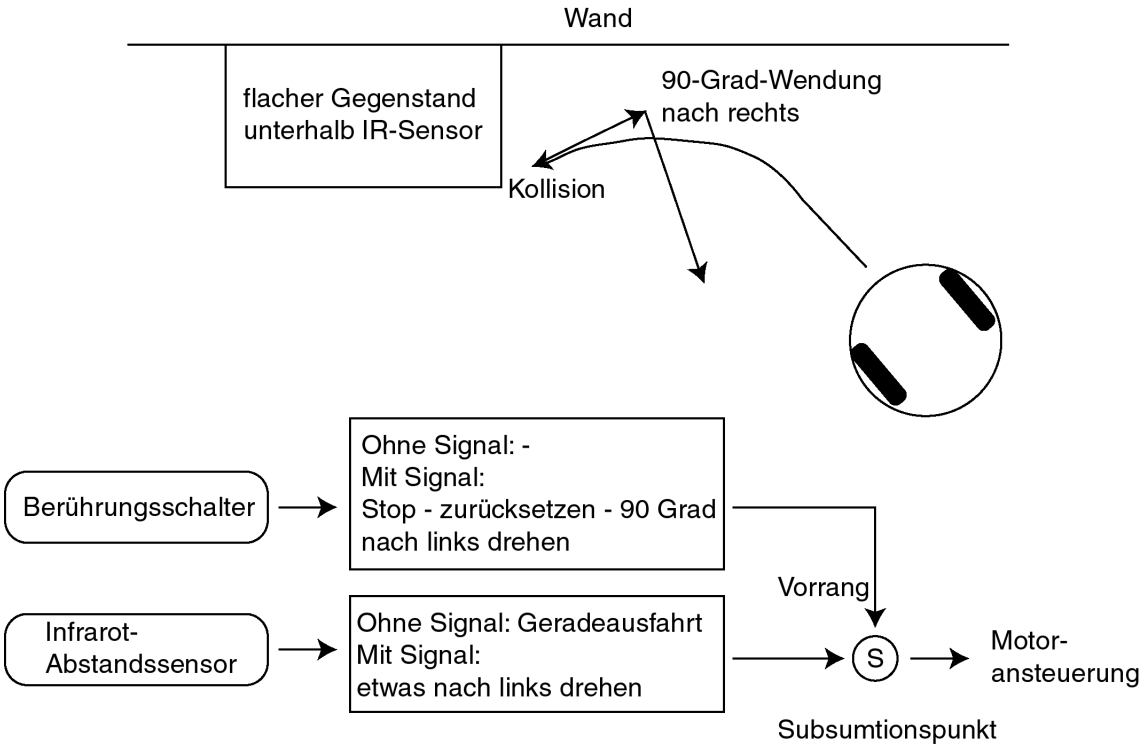


Abbildung 7.3.: Mobiler Roboter mit Verhaltensprogrammierung. Das Signal des Infrarotsensors löst zunächst das Verhalten *leichte Linkskurve* aus. Die Kollision löst das Verhalten *Stop – Zurücksetzen – 90 Grad nach rechts* aus. Letzteres Verhalten hat Vorrang.

7.5. Beispiele

7.5.1. Haushalts- und Unterhaltungsroboter

Abbildung 7.4.: Roomba Staubsaugerroboter der Fa. irobot

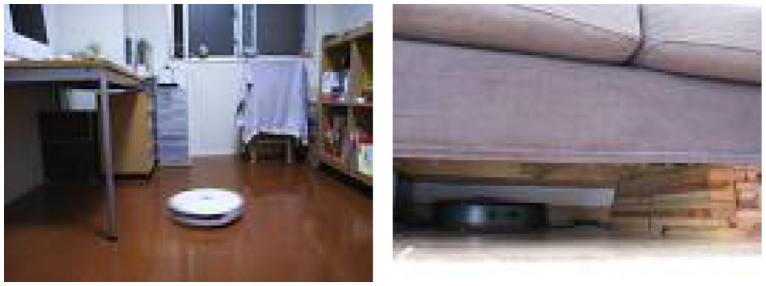


Abbildung 7.5.: Roomba Staubsaugerroboter bei der Arbeit. (Quelle: Wikimedia)

Der Roomba-Staubsaugroboter von irobot ...

- saugt automatisch und gründlich mit gegenläufig rotierenden Bürsten und kraftvollem Saugmotor - auch in Ihrer Abwesenheit.
- erkennt den Raum, navigiert rund um alle Hindernisse und kehrt automatisch zurück an seine Ladestation zum Aufladen.
- erkennt mit seinen Sensoren Abgründe, wie Treppenstufen oder versetzte Ebenen, und verhindert den drohenden Absturz.
- wird mit Virtual Walls (Virtuellen Wänden) kontrolliert. Das Virtual Wall Lighthouse (Virtuelle Leuchttürme) öffnet Roomba den nächsten Raum, wenn der vorherige fertig gesaugt ist

Abbildung 7.6.: Electrolux Trilobyte Staubsaugerroboter. (Quelle: Wikimedia)

Abbildung 7.7.: Rasenmäh-Roboter von Electrolux. (Quelle: Wikimedia)

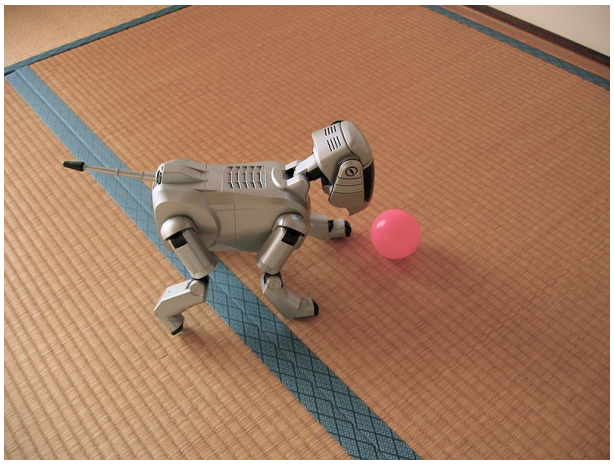


Abbildung 7.8.: Roboterhund AIBO von Sony. (Quelle: Wikimedia)

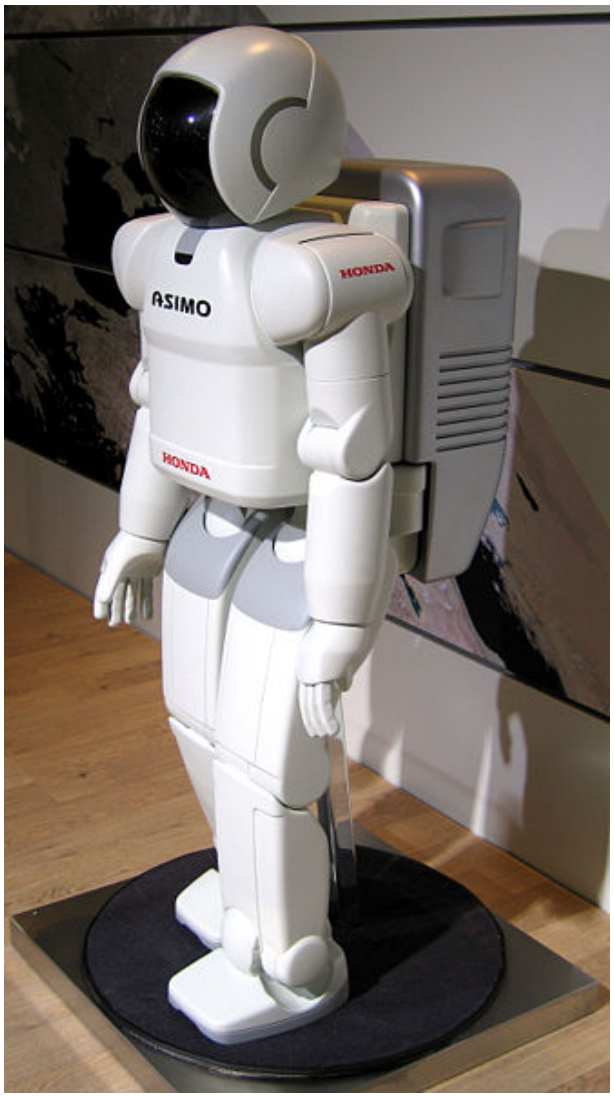


Abbildung 7.9.: Der Asimo von Honda ist der am weitesten entwickelte Humanoide der Welt. (Quelle Wikimedia)

Der Asimo von Honda kann

- gehen und rennen
- Treppen gehen
- hat Gleichgewicht (Gyroskop und Beschleunigungssensoren)
- kann mit jeder Hand 500g heben
- Betriebsdauer 1h
- 34 Freiheitsgrade (34 DOF)

Abbildung 7.10.:
Asimo bei Haushaltsarbeit

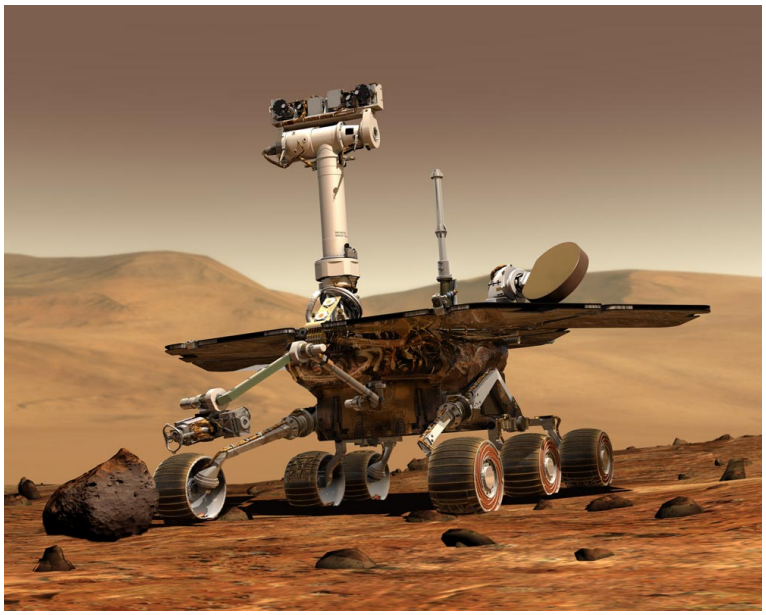


Abbildung 7.11.: Der Marsroboter Mars Rover. (Quelle NASA)

7.5.2. Weltraumroboter

Die Weltraumroboter Pathfinder, Sojourner und Rover erkundeten den Mars. Der Mars Rover Spirit arbeitet von 2004 – 2010 auf dem Mars. Er liefert mehr als eine Viertelmillion Bilder und legt eine Strecke von 26 Kilometern auf dem Mars zurück. Aufbau:

- Sechs unabhängig angetrieben Räder
- Geschwindigkeit 5 cm/s, Stopp alle 10s, dann 20s Untersuchung der Umgebung.
- Panoramakamera
- Vier HazCams (Hazard Avoidance Cameras, Hindernisvermeidungskameras) und zwei Navigationskameras.
- Viele geologische Analysegeräte
- Stromversorgung mit Solarzellen
- 32-Bit Rad 6000 Mikroprozessor (Variante des PowerPC)

Spirit hatte viele Computerprobleme und steckte am Ende im Sand fest.

Abbildung 7.12.: Foto aus dem Mars Rover. (Quelle NASA)

7.5.3. Unterwasser-Roboter und fliegende Roboter

Der Unterwasserroboter SEAL5000 (Sea Exploring Autonomous vehicle)

- Ausgefeilte Unterwasser-Sensorik
- 1.350 kg, Länge von 5,50 m
- Tauchtiefe bis 1400m
- „Long-Distance“ Akustik Modem
- 14 kWh Energievorrat (Li-Ionen Batterien)
- 2,4 kn Geschwindigkeit.

(Quelle: www.marum.de)

Der fliegende Kolibri-Roboter soll als Kundschafter eingesetzt werden.

- 2.6 g schwer
- 30 Flügelschläge pro Sekunde

Abbildung 7.13.: Unterwasser-Roboter SEAL 5000. (Mit freundlicher Genehmigung von Marum)

Abbildung 7.14.: Integration des SEAL 5000. (Mit freundlicher Genehmigung von Marum)

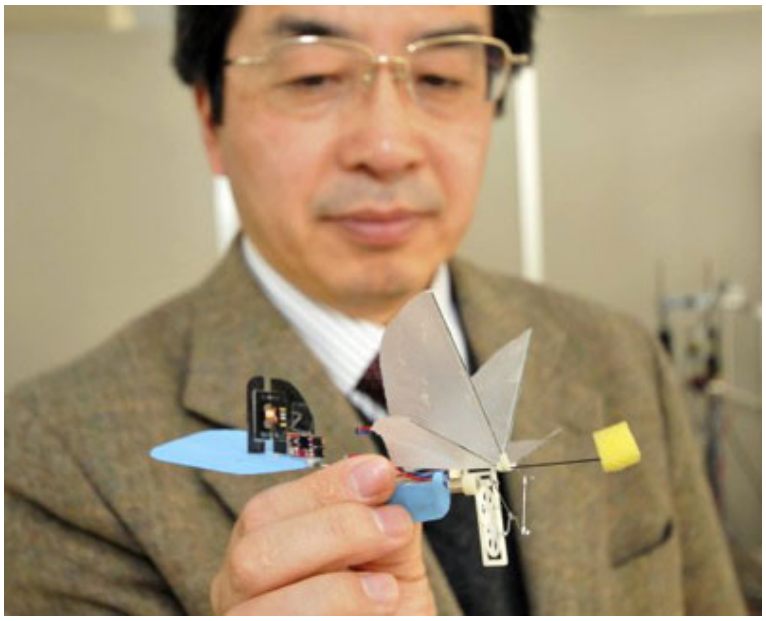


Abbildung 7.15.: Fliegender Kolibri-Roboter des japanischen Wissenschaftlers Hiroshi Liu.

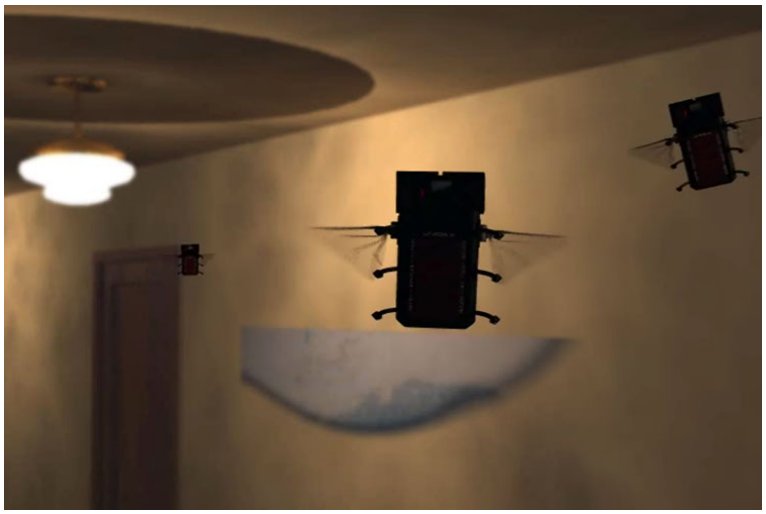


Abbildung 7.16.: Computergrafik einer für 2030 geplanten Mini-Drohne.

A. Die Denavit-Hartenberg-Konventionen

Ein serieller Roboter bildet eine offene kinematische Kette von Armteilen. An jedem Armteil ist also ein weiteres Armteil beweglich angebracht, bis das Ende der Kette beim Effektor erreicht ist. (siehe Abb.6.12)

Die Lage jedes Armteils wird durch ein Koordinatensystem beschrieben. Eine Transformationsmatrix, die den Übergang von einem AT-Koordinatensystem zum nächsten beschreibt, charakterisiert die aktuelle Stellung zweier Armteile zueinander. Zum Beispiel kann das Koordinatensystem K_2 mit der Transformationsmatrix ${}^2T_3$ aus K_3 erzeugt werden:

$$K_2 =^2 T_3 K_3$$

Dieser Übergang kann allerdings auf ganz verschiedenen Wegen geschehen. Welchen Rotations- und Translationsanteil die entstehende Transformationsmatrix ${}^2T_3$ enthält ist also keineswegs eindeutig bestimmt. Man vergleiche dazu Abb.6.1.7. Ebenso wenig klar ist, wo im Armteil die Koordinatensystem liegen sollen.

Um dazu einen Standard zu schaffen wurde von Denavit und Hartenberg vorgeschlagen, nach einem einheitlichen Schema vorzugehen. [7] Dieser Standard umfasst die Nummerierung der Armteile, das Legen eines Koordiantensystems in jedes Armteil und die Beschreibung der relativen Lage dieser Koordinatensysteme in Bezug zum jeweiligen Vorgänger. Leider wird die Denavit-Hartenberg-Norm in der Literatur unterschiedlich interpretiert und angewandt.

1. Nummerierung der Armteile Der festgeschraubte Fuß ist Armteil 0, das erste drehbare Armteil ist Armteil 1 usw. Das letzte Armteil ist der Handflansch/Effektor als Armteil N.

2. Kennzeichnung der Achsen Die Bewegungsachsen können Linearachsen (Gleitachsen) oder Rotationsachsen (Drehachsen) sein. Armteil 1 bewegt sich um bzw. entlang Bewegungsachse 1, Armteil zwei um bzw. entlang Bewegungsachse 2 usw.

In jedes Armteil wird ein Koordinatensystem gelegt. In Armteil 0 liegt Koordinatensystem K_0, in Armteil 1 Koordinatensystem K_1 usw. Jedes Koordinatensystem wird so gelegt, dass die z-Achse mit der Drehachse des nachfolgenden Armteiles übereinstimmt. Es liegt also die z_0-Achse in der Bewegungsachse 1, die z_1-Achse in der Bewegungsachse 2 usw. Die Festlegung der Koordinatensysteme richtet sich grundsätzlich nach der Lage des vorhergehenden Koordinatensystems. K_1 wird also nach K_0 ausgerichtet, K_2 wird also nach K_1 usw. Die x_i-Achse zeigt immer in Richtung auf das nächste Armteil. (Abb.A.1)

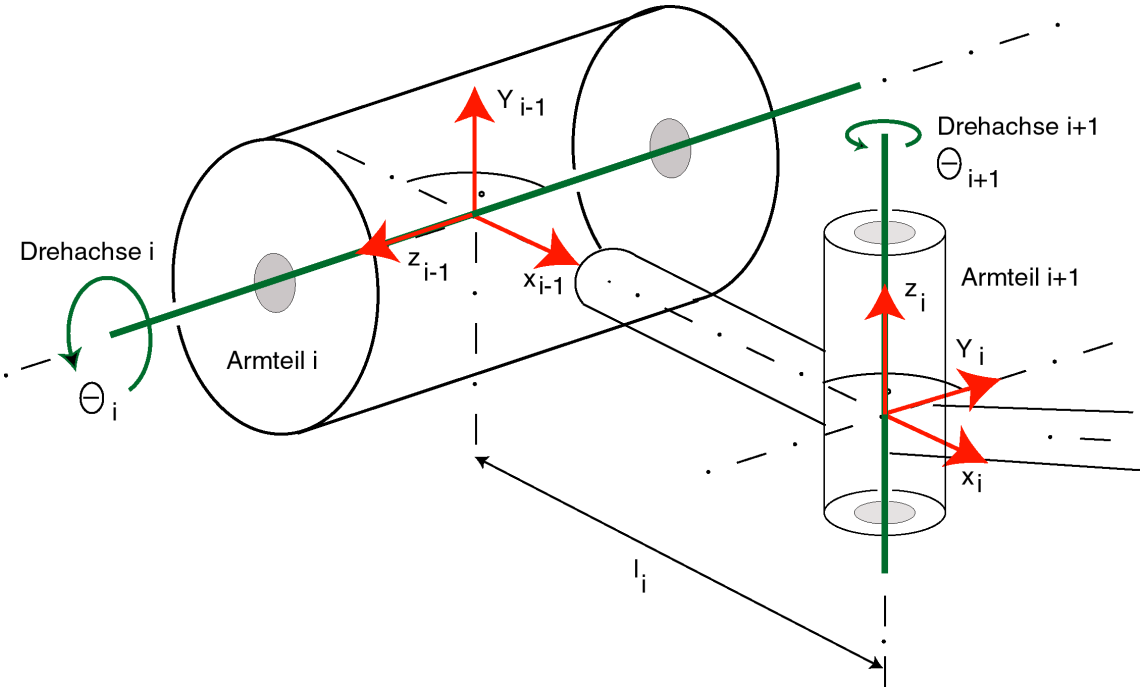


Abbildung A.1.: Beispiele für Koordinatensysteme in Armteilen, die gemäß Denavit-Hartenberg-Konvention gelegt wurden .

3. Festlegung des Basis-Koordinatensystems K_0 Dies ist das raumfeste Koordinatensystem, das fest mit dem Fuß (Armteil 0) verbunden ist. Die z_0-Achse liegt in der ersten Gelenkachse, die x_0- und die y_0-Achse sind frei und werden möglichst sinnvoll gelegt. Die x_0- und die y_0-Achse müssen mit der z_0-Achse ein Rechtssystem bilden. Man kann K_0 in den Fußpunkt des Roboters legen. Wenn man es so legt, dass eine der K_0-Achsen den Ursprung von K_1 schneidet, vereinfachen sich die Transformationsgleichungen.

4. Festlegung der Koordinatensysteme $K_1 \dots K_{N-1}$ Es gibt drei Fälle:

1. Wenn sich die Bewegungsachsen i und $i+1$ (entsprechend z_{i-1} und z_i) nicht schneiden und auch nicht parallel verlaufen, wird die gemeinsame Normale (kürzeste Verbindung) zwischen diesen beiden Bewegungsachsen gesucht. Der Ursprung wird in den Schnittpunkt dieser Normalen

mit der Gelenkachse $i+1$ gelegt. Die z_i-Achse liegt in der Bewegungsachse $i+1$; es kann eine von beiden Richtungen ausgewählt werden. Die x_i-Achse wird entlang der gemeinsamen Normalen, zu z_{i+1} weisend, gelegt. Oft ist dies die Richtung der mechanischen Verbindung des Armteiles zwischen den Gelenkachsen i und $i+1$. Die y_i-Achse wird so gelegt, dass sich ein Rechtssystem ergibt. (s. Abb. A.2)

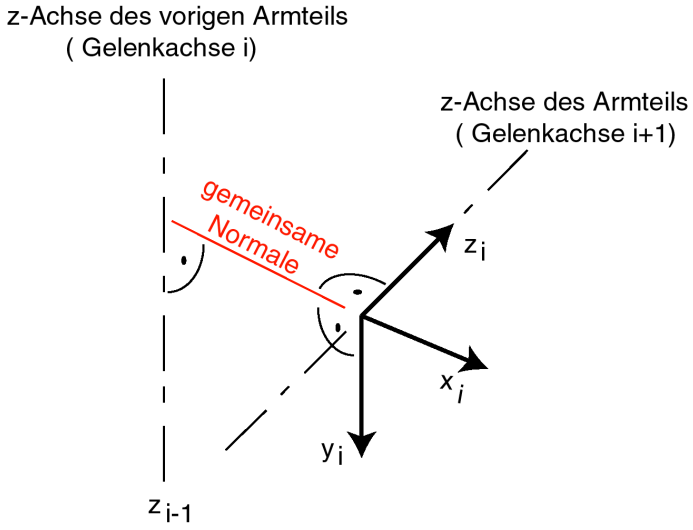


Abbildung A.2.: Die Festlegung des Koordinatensystems wenn die Gelenkachse die vorige Gelenkachse nicht schneidet und nicht parallel dazu verläuft.

2. Wenn die Bewegungsachsen i und $i+1$ sich schneiden, liegt der Ursprung von K_i im Schnittpunkt der Achsen. Die z_i-Achse liegt in der Bewegungsachse $i+1$, die x_i-Achse wird senkrecht zu beiden Achsen (parallel oder antiparallel zum Vektor des Kreuzproduktes $z_i \times z_{i-1}$), zu z_{i+1} weisend, gelegt. Die y_i-Achse wird so gelegt, dass sich ein Rechtssystem ergibt. (Abb. A.3)

3. Wenn die Bewegungsachsen i und $i+1$ parallel verlaufen, gibt es unendlich viele gemeinsame Normalen. Darunter wird entweder die kürzeste Verbindung zu K_{i-1} oder zu K_{i+1} ausgewählt. Der Schnittpunkt dieser Verbindung mit der Bewegungsachse $i+1$ ist der Ursprung von K_i. Die z_i-Achse liegt wieder in der Bewegungsachse $i+1$, die y_i-Achse wird so gelegt, dass sich ein Rechtssystem ergibt. (Abb. A.4 und A.5)

5. Koordinatensystem K_N (Handflansch/Effektorkoordinaten) Hier ist die kinematische Kette zu Ende, man hat daher mehr Freiheit. Das Koordinatensystem K_N muss so gelegt werden, dass es entsprechend den nachfolgenden Transformationsregeln aus K_{N-1} erzeugt werden kann. Dazu muss x_i senkrecht zu z_{N-1} gelegt werden. z_N kann die Annäherungsrichtung des Effektors sein.

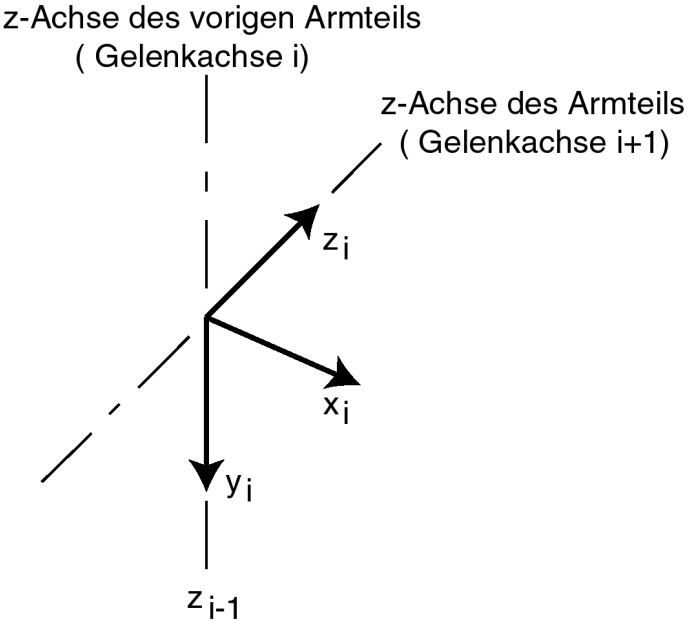


Abbildung A.3.: Die Festlegung des Koordinatensystems wenn die Bewegungsachsen sich schneiden.

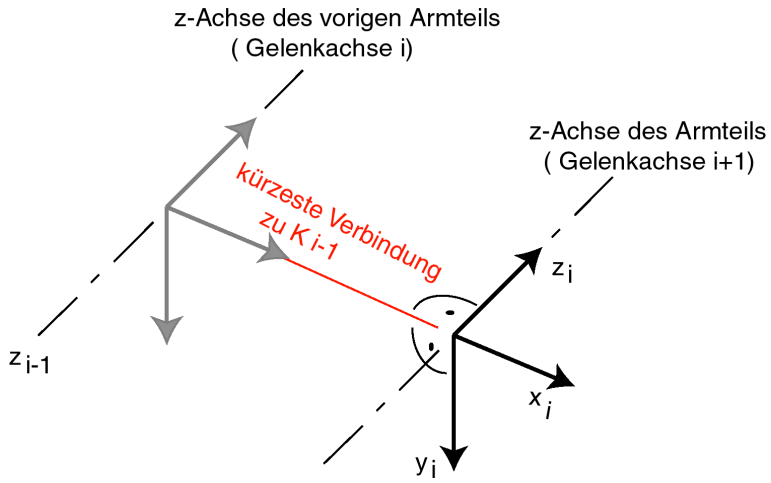


Abbildung A.4.: Die Festlegung des Koordinatensystems wenn die Bewegungsachsen parallel verlaufen und die kürzeste Verbindung zu K_{i-1} benutzt wird.

A.0.3.1. Die Denavit-Hartenberg-Parameter

Zwei benachbarte und nach obigen Konventionen festgelegte Koordinatensysteme können durch Transformationen ineinander transformiert werden. Aus K_{i-1} wird K_i durch folgende vier Operationen in dieser Reihenfolge:

1) Eine Drehung um die Achse x_{i-1} um den Winkel α_i. *Dieser Parameter beschreibt die Verdrehung der Achse* z_i *gegenüber* z_{i-1}

2) Eine Translation um a_i in Richtung der Achse x_{i-1}. *Dieser Parameter be-*

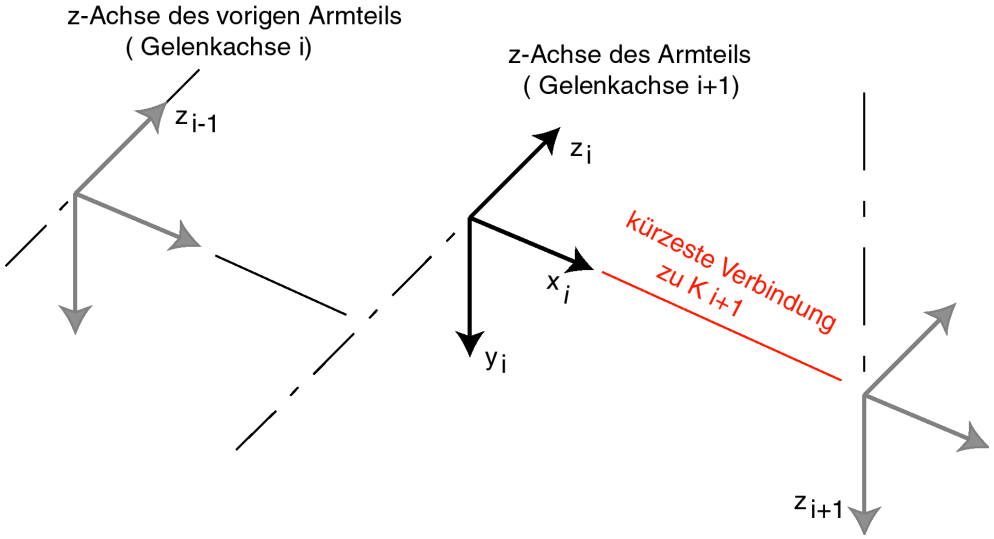


Abbildung A.5.: Die Festlegung des Koordinatensystems wenn die Bewegungsachsen parallel verlaufen und die kürzeste Verbindung zu K_{i+1} benutzt wird.

schreibt die Länge des Verbindungsgliedes

3) Eine Translation um d_i in Richtung der Achse z_{i-1}. *Dieser Parameter beschreibt den Versatz der Gelenke in z-Richtung oder eine Linearachse.*

4) Eine Rotation um den Drehwinkel θ_i um das Gelenk i (Achse z_{i-1}). *Dieser Parameter beschreibt eine Drehachse oder den Drehwinkel um aufeinanderfolgende x-Achsen gleich auszurichten.*

Anmerkungen:

- Ist die Achse i eine Rotationsachse, ist der Drehwinkel θ_i die veränderliche Gelenkkoordinate und d_i, a_i, α_i sind konstante Konstruktionsparameter.
- Ist die Achse i eine Translationsachse, ist die Schublänge d_i die veränderliche Gelenkkoordinate und θ_i, a_i, α_i sind konstante Konstruktionsparameter.
- Bei den üblichen Roboterkonstruktionen kommt es häufig vor, dass einer der Parameter Null ist.

Als Transformationsmatrix ergibt sich:

$$
\begin{aligned}
T &= Rot(z,\theta_i)\cdot Trans(a_i,0,d_i)\cdot Rot(x,\alpha_i) \\
&= \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
\tag{A.1}
$$

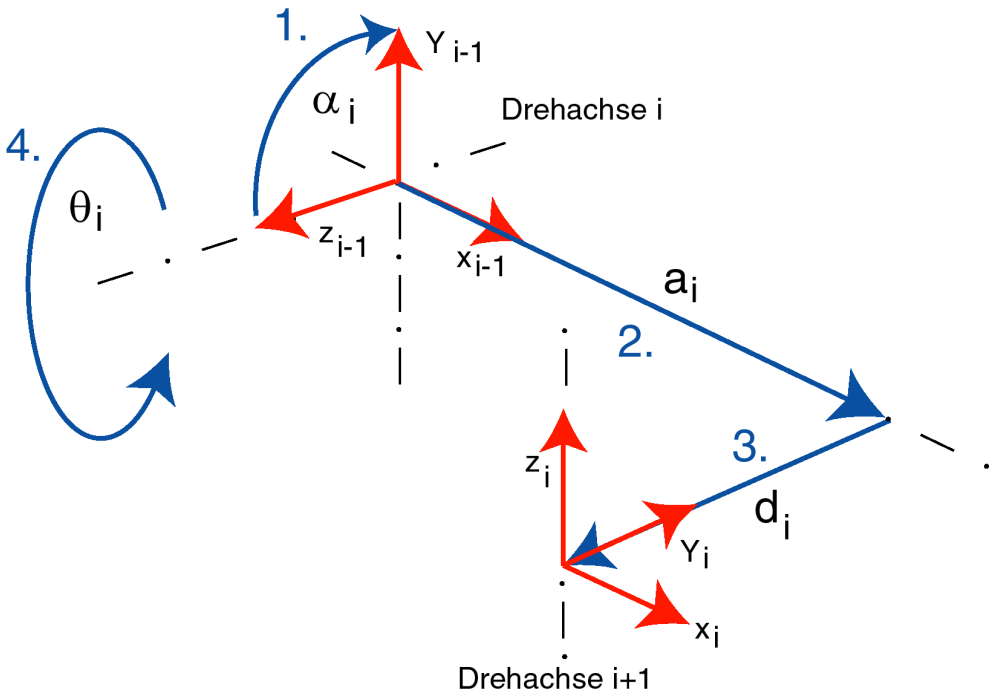


Abbildung A.6.: Die vier Transformationen am Beispiel der beiden Koordinatensysteme aus Abb. A.1. Der Parameter d_i wurde hier als endliche Verschiebung eingezeichnet, ist aber in unserem Fall gleich Null.

Wenn die Koordinatensysteme gemäß den Denavit-Hartenberg-Konventionen gelegt sind, kann jeder Übergang von einem Armteil zum nächsten mit dieser Matrix beschrieben werden. Die darin vorkommenden Parameter heißen Denavit-Hartenberg-Parameter. Dies sind:

1. Der Winkel α_i,
2. die Verschiebung d_i (bei translatorischen Achsen angetrieben und variabel),
3. die Verschiebung a_i,
4. der Winkel θ_i (bei rotatorischen Achsen angetrieben und variabel).

Die Denavit-Hartenberg-Matrizen werden üblicherweise mit A_i bezeichnet, um sie von anderen Transformationen zu unterscheiden. A_1 ist die Matrix, die Koordinatensystem K_0 in Koordinatensystem K_1 transformiert und Koordinaten von der Darstellung in K_1 in die K_0-Darstellung bringt:

$$K_1 = A_1 K_0 \tag{A.2}$$

$${}^0P = A_1 \, {}^1P \tag{A.3}$$

Entsprechendes gilt für alle anderen Gelenke.

Als ein einfaches Beispiel für die Verwendung der DH-Konventionen ist in Abschnitt 6.3.3 die Vorwärtstransformation am TR-Roboter mit DH-Matrizen gezeigt.

B. Ausgewählte Beispiele

B.1. LLR-Manipulator

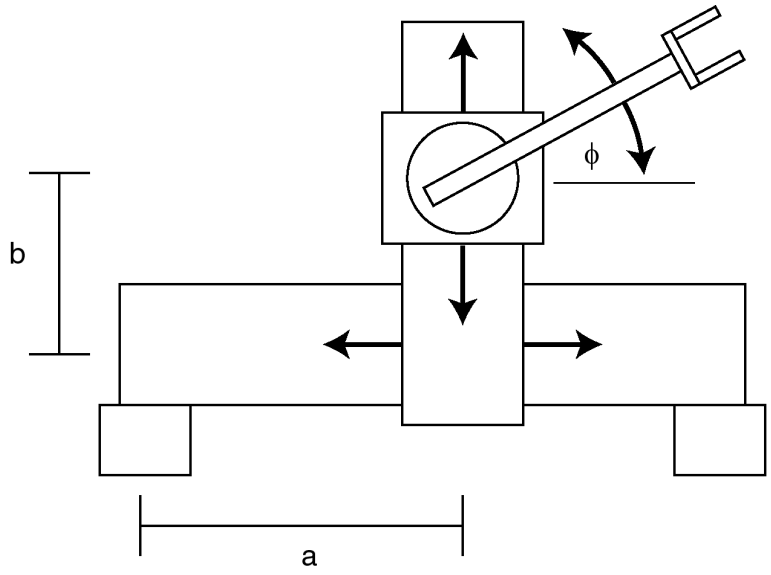


Abbildung B.1.: Roboter mit LLR-Kinematik

Führen Sie an dem oben abgebildeten Roboter (Werkzeugmaschine) folgende Bestimmungen durch:

a) Legen Sie in jedes Armteil ein Koordinatensystem gemäß den Denavit-Hartenberg-Konventionen. Nutzen Sie evtl. vorhandene Wahlmöglichkeiten so, dass die Transformationsmatrizen in Teil b) möglichst einfach werden. Stellen Sie die Koordinatensysteme in einer Skizze dar. Hinweis: Das Fußkoordinatensystem K_0 nach DH stimmt nicht mit den eingezeichneten Achsen (x- und y-Achse) überein.
b) Bestimmten Sie die Denavit-Hartenberg-Parameter für die Armteile und tragen Sie diese in eine Tabelle ein!
c) Bestimmen Sie die Denavit-Hartenberg-Transformationsmatrizen A1, A2, A3.
d) Geben Sie die Gleichung der Vorwärttransformation in Matrizenform an.

Nehmen Sie für die Aufgabenteile e) - g) an, dass eine aktuelle Roboterstellung gegeben ist durch die Gelenkkoordinaten und Konstruktionsparameter: $a = 3, b = 2, \phi = 45^o$ und $r = \sqrt{2}$.

e) Berechnen Sie die Koordinaten des TCP (Ursprung von K_3) im Fußkoordinatensystem K_0.
f) Berechnen Sie die Koordinaten des Punktes, der im Effektorkoordinatensystem KS3 die Koordinaten 3(2,2,1) hat, im Fußkoordinatensystem K_0.
g) Berechnen Sie die Koordinaten des Punktes, der im Fußkoordinatensystem K_0 die Koordinaten ${}^0(4,-3,4)$ hat, im Effektorkoordinatensystem K_3.

Lösung

a) Die Koordinatensysteme werden wie folgt in die Armteile 0 – 3 gelegt:

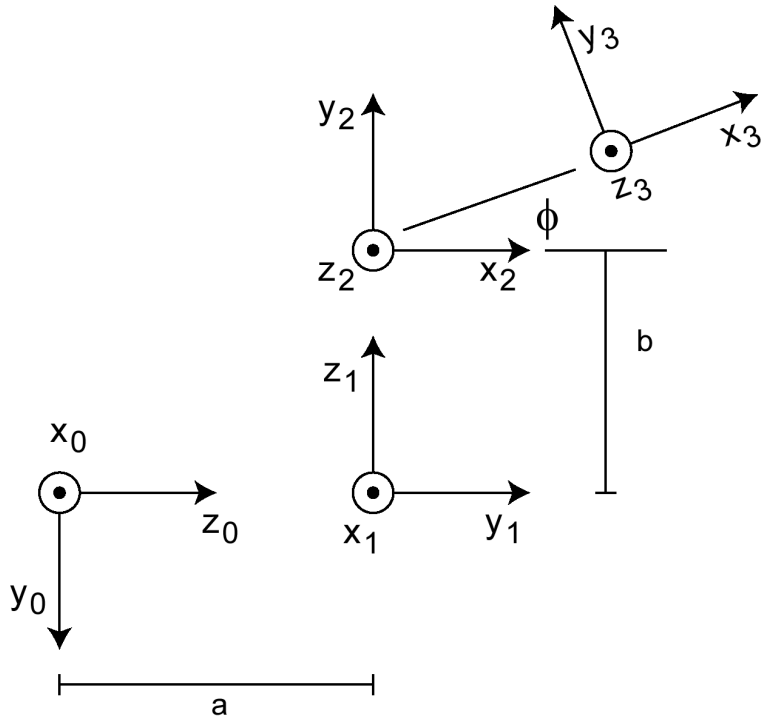


Abbildung B.2.: Koordinatensysteme gemäß Denavit-Hartenberg für obigen Roboter

b) Die Denavit-Hartenberg-Parameter, die diese Koordinatensysteme ineinander überführen, sind:

Armteil	ϕ	d_i	a_i	α_i
1	0	a	0	90°
2	90°	b	0	90°
3	ϕ	0	r	0

c) Aus der obigen Tabelle ergeben sich gemäß Gl. A.1 die folgenden DH-Transformationsmatrizen:

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & a \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$
\begin{aligned}
A_2 &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
A_3 &= \begin{pmatrix} \cos\phi & -\sin\phi & 0 & r\cos\phi \\ \sin\phi & \cos\phi & 0 & r\sin\phi \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{B.1}
$$

d) Die Gesamttransformation wird bestimmt durch folgendes Matrizenprodukt:

$$
A_1A_2A_3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\sin\phi & -\cos\phi & 0 & -r\sin\phi - b \\ \cos\phi & -\sin\phi & 0 & r\cos\phi + a \\ 0 & 0 & 0 & 1 \end{pmatrix} = {}^0T_3
$$

e) Es ist nun $a = 3$, $b = 2$, $\phi = 45^o$, $r = \sqrt{2}$, $\sin\phi = \cos\phi = \sqrt{1/2}$. Damit wird die obige Matrix für die Gesamttransformation zu

$$
A_1A_2A_3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\sqrt{1/2} & -\sqrt{1/2} & 0 & -3 \\ \sqrt{1/2} & -\sqrt{1/2} & 0 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Die Position des TCP in Koordinaten des Koordinatensystems K_0 ist

$$
{}^0P_{TCP} = A_1A_2A_3 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\sqrt{1/2} & -\sqrt{1/2} & 0 & -3 \\ \sqrt{1/2} & -\sqrt{1/2} & 0 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -3 \\ 4 \\ 1 \end{pmatrix}
$$

f) Der Punkt $P = {}^3 (2, 2, 1, 1)^T$ im Koordinatensystem K_0 ist

$$
{}^0P = A_1A_2A_3 \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -4\sqrt{1/2} - 3 \\ 4 \\ 1 \end{pmatrix}
$$

g) Um die Koordinaten eines Punktes P von K_0 auf K_3 umzurechnen braucht man die inverse Matrix der Gesamttransformation:

$$
{}^3P = {}^3T_0 \, {}^0P = (A_1A_2A_3)^{-1} \, {}^0P
$$

Die inverse Matrix errechnet sich nach Gl. 6.16

$$ {}^3T_0 = \begin{pmatrix} 0 & -\sqrt{1/2} & \sqrt{1/2} & -\sqrt{2} - 5\sqrt{1/2} \\ 0 & -\sqrt{1/2} & -\sqrt{1/2} & \sqrt{1/2} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Damit ergibt sich für den gesuchten Punkt:

$$ {}^3P = {}^3T_0 \begin{pmatrix} 4 \\ -3 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 4 \\ 1 \end{pmatrix} $$

B.2. Bahnprogrammierung Lackieranwendung

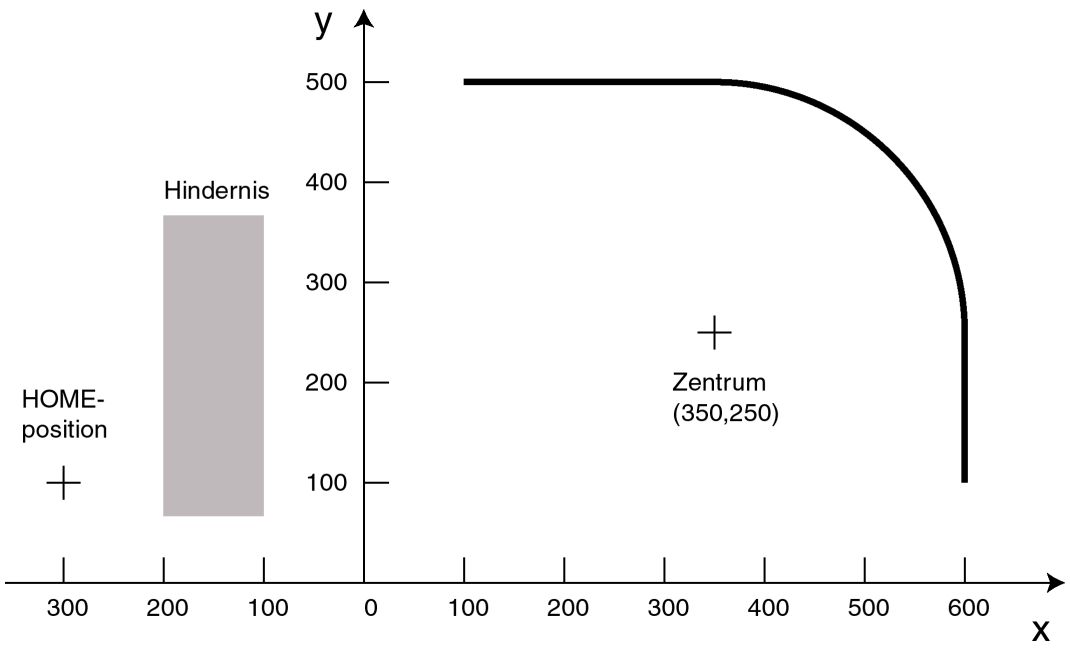


Abbildung B.3.: Werkstück mit Blech für Lackieranwendung.

In einer Lackieranwendung soll ein Blech von innen lackiert werden. Dazu muss die Lackierdüse in gleichmäßigem Abstand an der Innenseite des Bleches vorbeigeführt werden. Die Lackierdüse gibt einen radialsymmetrischen Sprühstrahl ab (z.B. einen Kegel). Die Aufgabe soll als ebenes Problem nur in der x-y-Ebene betrachtet werden. Der Drehwinkel z soll dann gleich Null sein, wenn der Sprühstrahl in Richtung der x-Achse geht. Die Anordnung ist in der folgenden Abb. dargestellt.

a) Geben Sie an, wie viele Freiheitsgrade in diesem Problem sinnvoll genutzt werden sollten gibt und mit welchen Koordinatenangaben hier Position und

Orientierung des Effektors beschrieben werden können.
b) Gehen Sie davon aus, dass der Lackiervorgang in der HOME-Position beginnt und programmieren Sie den Roboter so, dass er auch dort wieder endet. Der Abstand vom Blech soll beim Sprühvorgang 100 mm betragen, die Verfahrgeschwindigkeit am Aufsprühpunkt 80 mm/s. Geben Sie für diese Aufgabe die vollständigen Teachpunkte und die Steuerungsarten sowie den Schaltzustand der Lackierdüse (EIN/AUS) zwischen den Teachpunkten in der Art einer Programmiertabelle an! Zum Anfahren der HOME-Position soll es den Befehl "HOME"geben.

Lösung

a) Es sind drei Freiheitsgrade sinnvoll und notwendig: Translation in x-Richtung, Translation in y-Richtung und Drehung um die z-Achse: x, y, ϕ_z. Weitere Freiheitsgrade sind nicht sinnvoll, da
1. Immer $z = 0$ ist,
2. der Sprühstrahl radialsymmetrisch ist,
3. die Sprühachse immer in der x-y-Ebene liegt.

b) Zunächst werden folgende Festlegungen gemacht:
1. Es ist $\phi_z = 0$ wenn die Sprühachse parallel zur x-Achse liegt. Dies kann durch geeignete Wahl von Bezugskoordinaten erreicht werden.
2. Der TCP wird in den optimalen Auftreffpunkt des Sprühstrahles gelegt, also $100\,mm$ vor das Werkzeug. Dies kann durch geeignete Definition des Werkzeugs erreicht werden.

Die Programmtabelle wird damit:

Teachpunkt	Befehl	x	y	ϕ_z	Kommentar
1	Move,PTP,vmax,zone=50	-300	400	90°	Kollisionsvermeidung
2	Move,PTP,vmax,zone=10	100	400	90°	Annäherungspunkt
3	Move,LIN,v=80,zone=0	100	500	90°	Annäherung
4	Digital OUT LD,1				Lackierdüse EIN
5	Move,LIN,v=80,zone=0	350	500	90°	gerades Stück lackieren
6	Move,ZIRK ZP=(527,427), v=80,zone=0	600	250	0°	Bogenstück lackieren, ZP = Zwischenpunkt gerundet
7	Move,LIN,v=80,zone=0	600	100	0°	gerades Stück lackieren
8	Digital OUT LD,0				Lackierdüse AUS
9	Move,LIN,v=80,zone=10	500	100	0°	Abrücken
10	Move,PTP,vmax,zone=50	-100	400	90°	Kollisionsvermeidung
11	HOME				

C. Praktikum Mobile Roboter (MR)

C.1. Rug Warrior

Der Rug Warrior (deutsch: Teppich Krieger) ist ein preiswerter mobiler Roboter, der am MIT für Zwecke der Forschung und Lehre entwickelt wurde. Er erlaubt praktische Erfahrungen mit Sensorik, Parallelverarbeitung, Verhaltens- und Echtzeitprogrammierung.

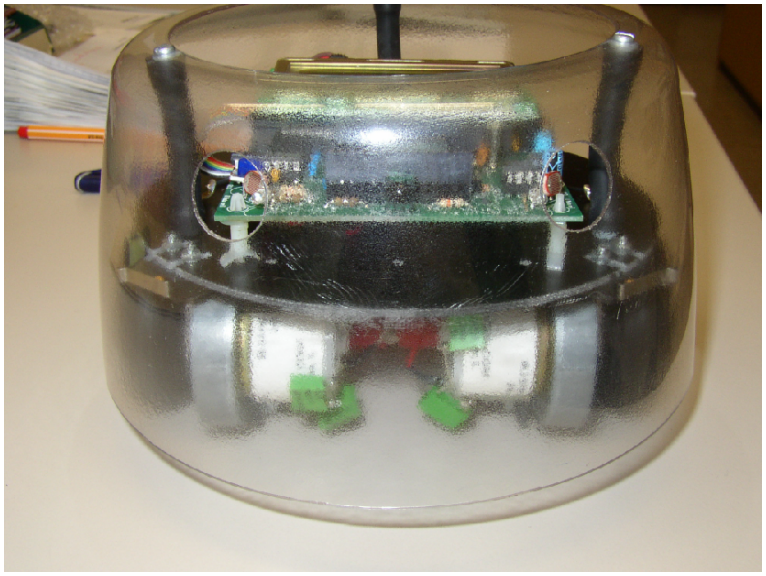


Abbildung C.1.: Rug Warrior von vorne. Rechts und links sind die Öffnungen für die IR-Sendedioden in der Plexiglasschürze zu erkennen.

C.1.1. Fahrwerk und Chassis

Der Rug Warrior ist auf einer kreisförmigen Grundplatte aufgebaut. Auf der Unterseite der Grundplatte befinden sich zwei seitliche Antriebsräder und das hintere frei mitlaufende Stützrad. Die Antriebsräder sind jeweils direkt an eine Motor-Getriebe-Einheit montiert. Ebenfalls auf der Unterseite befindet sich der Batterie-/Akkupack.

Diese Art des Antriebs erlaubt dem Rug Warrior sehr flexible Bewegungsrichtungen, er kann sich sogar auf der Stelle drehen.

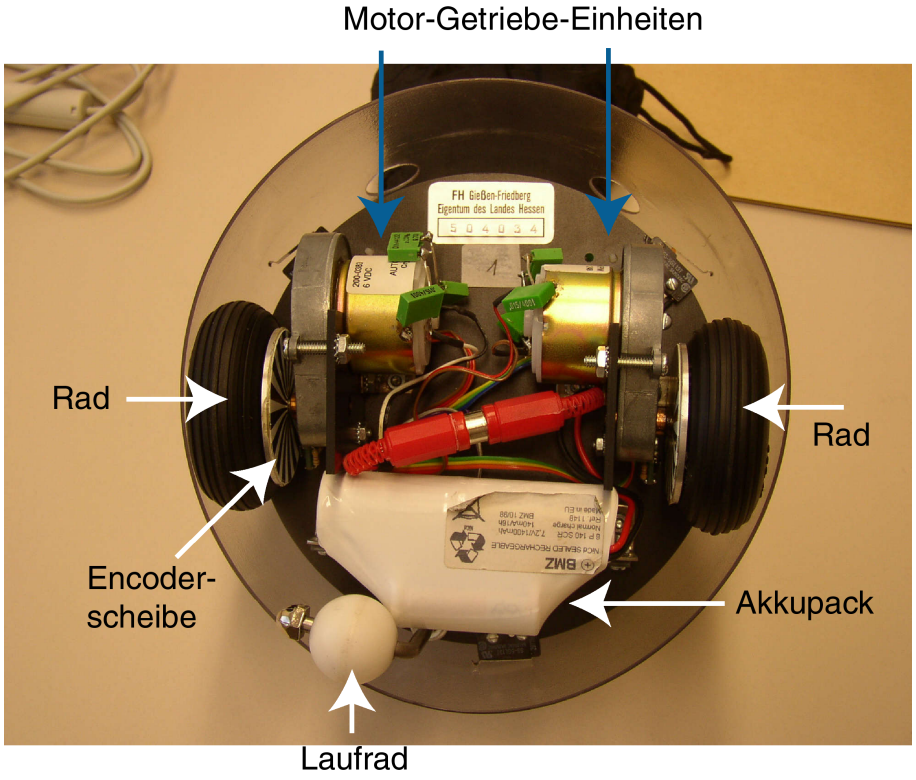


Abbildung C.2.: Der Rug Warrior von unten gesehen.

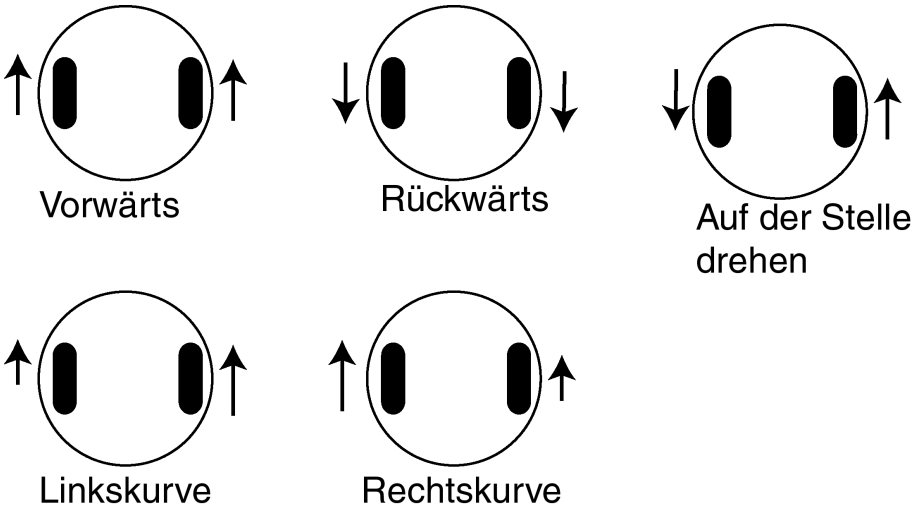


Abbildung C.3.: Bewegungsmöglichkeiten des Rug Warrior.

C.1.2. Steuerung und Programmierung

Der Rug Warrior wird durch einen Motorola 68HC11-Microcontroller gesteuert und hat 32kB RAM. In diesem RAM residiert ein PCode-Interpreter (Pseudo-Code-Ausführer). Der PCode-Interpreter führt in PCode geschriebene Programme aus. Hier werden die Programme in der Programmiersprache C geschrie-

ben, vom *Interactive C-Compiler* (IC) in PCode übersetzt, und nach Download an den PCode-Interpreter (Interpreter-Teil von Interactive C) auf dem Rug Warrior übergeben. Interactive C ist für den Rug Warrior gleichzeitig ein Miniatur-Betriebssystem, das Programme und Prozesse laden, verwalten und ausführen kann. Es ist sogar ein Multiprocessing möglich. Der Download erfolgt über eine serielle RS232-Verbindung.

Der Ablauf für die Ausführung eines Programmes ist:
1) Download des PCode-Interpreters auf den Rug Warrior.
2) Download eines C-Programmes oder einzelner C-Kommandos, dabei Umwandlung in PCode.
3) Ausführung des C-Programmes oder C-Kommandos.

Für das Schreiben eigener Programme gibt es eine Funktionsbibliothek, die viele nützliche Funktionen anbietet. Für die Verwaltung der Prozesse und den Zugriff auf den Arbeitsspeicher stehen folgende Funktionen zur Verfügung:
`load, unload, kill_all, ps, start_process(), kill_process(), sleep(), hog_processor(), defer()`
`peek(), poke(), bit_set(), bit_clear()`

C.1.3. Sensorik

Rund um die kreisförmige Grundplatte ist eine bewegliche Schürze aus Plexiglas angebracht. Innerhalb der Schürze befindet sich auf der Oberseite des Rug Warrior die Mikrocontrollereinheit und die Sensorik.

Mikroschalter (Bumper) In 120^o Winkelabstand rundum angeordnete Kollisionssensoren, schließen bei Kollision; Bibliotheksfunktion: `bumper()`

Infrarot-Abstandssensoren Linke und rechte Sendediode, gemeinsamer mittlerer IR-Detektor gibt HIGH/LOW aus, IR moduliert mit 40 kHz zur Herausfilterung von Fremdlichteinfluss, softwaremäßig Pulse zu 600 μs zur Ergebnisabsicherung; Bibliotheksfunktionen: `ir_detect()`

Lichtabhängige Widerstände LDR, links und rechts vorne, ändern ihren Widerstandswert mit dem Lichteinfall
Bibliotheksfunktionen: `analog(photo_left|photo_right)`

LCD-Anzeige Zwei Zeilen, Punktmatrix Bibliotheksfunktionen: `printf()`

Piezo-Summer
Bibliotheksfunktionen: `beep(), tone(), set_beeper_pitch(), beeper_on(), beeper_off()`

Mikrofon Erfasst Geräusche, Verstärkung, Analog/Digital-Wandlung, Ruhewert ca. 128; Bibliotheksfunktion: `analog(microphone)`

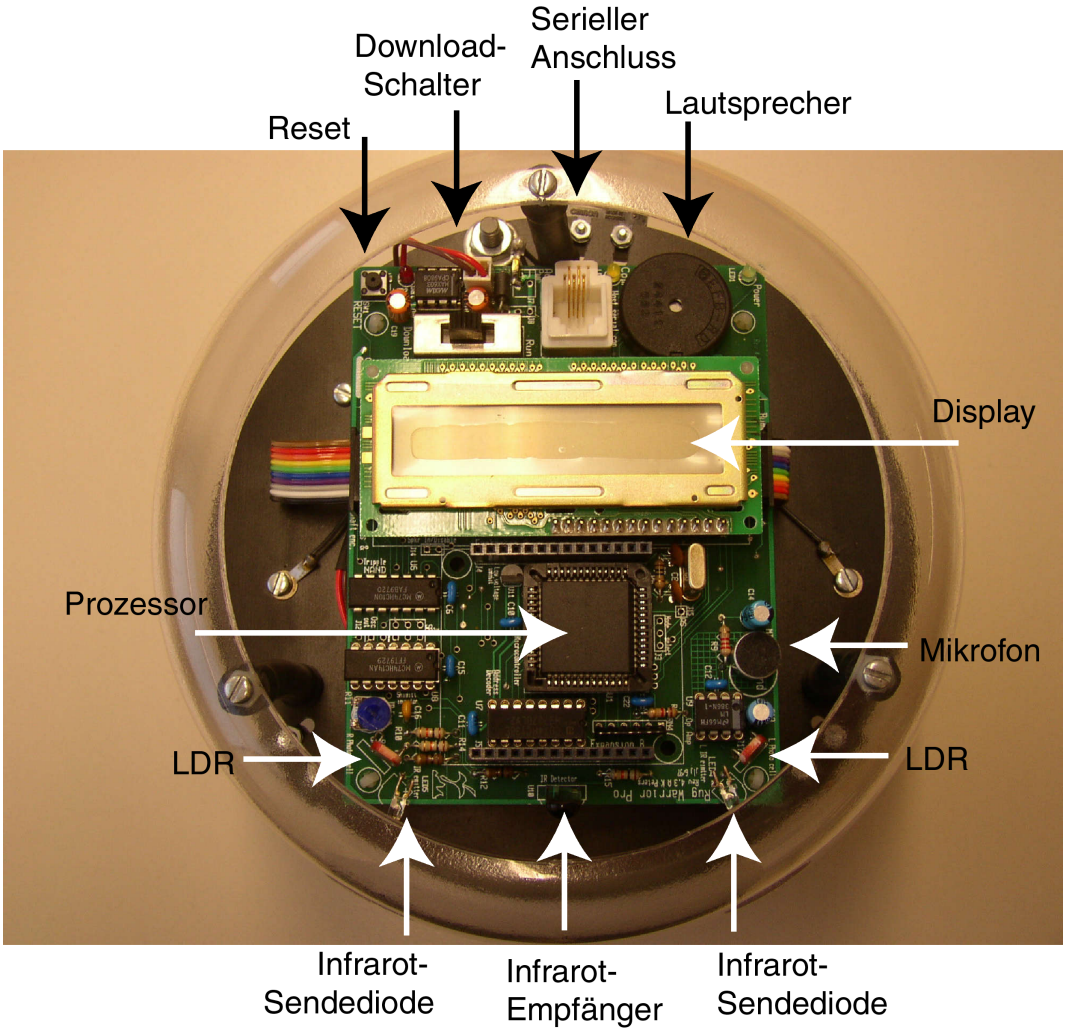


Abbildung C.4.: Rug Warrior von oben.

Radencoder Sektorenscheiben an Innenseite der Räder werden mit Reflexlichtschranken abgetastet, Impulse bei Drehung eines Rades
Bibliotheksfunktionen: `init_velocity()`, `get_left_clicks()`, `get_right_clicks()`, `left_shaft()`, `right_shaft()`

C.1.4. Motoren, LCD, Piezo-Summer

Motoren Zwei unabhängige Motoren an den beiden Rädern, Leistungssteuerung durch PWM möglich
Bibliotheksfunktionen: `init_motors()`, `motor()`, `drive()`, `track()`

Piezo-Summer Bibliotheksfunktionen: `beep()`, `tone()`, `set_beeper_pitch()`,

LCD-Anzeige Zwei Zeilen, Punktmatrix-Anzeige, Bibliotheksfunktion: `printf()` Es können Texte und Zahlen ausgegeben werden, z.B.

```
printf("Zustand Bumper:\%i",bumper());
```

Aufgabe 1) Rug Warrior booten

Um den Rug Warrior und die InteractiveC-Entwicklungsumgebung betriebsbereit zu machen, müssen folgende Schritte durchgeführt werden:

1. Rug Warrior Kabel in seriellen Schnittstellenstecker einstecken.
2. Rug Warrior Boot-Schalter auf *Download* stellen,
3. PC Interactive C starten, Fenster maximieren,
4. PC Configure Board Settings? → *Ja*,
5. PC *Download PCode*, Datei `RugWarrior_Pro.icd` wählen,
6. Rug Warrior *Reset-Taste* drücken,
7. PC *ok* bestätigen, Datenübertragung läuft, gelbe Leuchtdiode flackert,
8. PC Meldung *Download complete*
9. Rug Warrior Boot-Schalter auf *Run* stellen,
10. PC *ok* bestätigen,
11. Rug Warrior Herz schlägt, Rug Warrior ist betriebsbereit.

Testen Sie nun kurz das System, indem Sie vom Entwicklungsrechner (PC) einige Kommandos zum Rug Warrior senden um

- Den Piezo-Summer piepen zu lassen,
- auf der LCD-Anzeige „Hallo Welt“ ausgeben,
- den Prozess-Status abfragen,
- die geladenen Funktionen auflisten,
- die geladenen Dateien auflisten.

Benutzen Sie dazu das Interaction-Fenster in der Entwicklungsumgebung und das Interactive C-Handbuch im Anhang.

Aufgabe 2) Hardwaretest

Laden Sie die Datei `rwp-test.lis` durch Download auf den Rug Warrior. Starten Sie das Programm durch drücken der RESET-Taste am Rug Warrior. Das Programm testet nun Gruppe für Gruppe die auf dem Rug Warrior vorhandene Hardware. Nach jeder Hardwaregruppe drückt man wieder RESET und kommt damit zur nächsten Gruppe.

Beantworten Sie folgende Fragen:

1. Bis zu welchem Abstand wirken die Infrarot-Abstandssensoren?
2. Bis zu welchem Abstand spricht das Mikrofon an, wenn man hineinpfeift?
3. Wie sind die Rückgabewerte der Berührungsschalter bei Auslösung
   - rechts
   - links
   - hinten
   - rechts und links
   - rechts und hinten
   - links und hinten

   Treffen Sie eine bitweise Zuordnung der Schalter zu den Bits im Ergebniswert!
4. Wie viele Impulse werden an den Radencodern gezählt, wenn der Rug Warrior 30 cm (ungefähr die lange Seite eines DIN A4-Blattes) zurücklegt? Den Roboter dazu vorsichtig schieben!

Aufgabe 3) Motorenansteuerung

Öffnen Sie das Programm `demo.c` im Verzeichnis `IC/libs` und versuchen Sie zu erkennen, was es macht. Welche Funktionen gibt es, wie wird es gestartet? Laden Sie das Programm mit der Listendatei `demo.lis` auf den Rug Warrior und probieren Sie es aus.

Ändern Sie nun das Programm so ab, dass weiche Bewegungen entstehen. Dazu kann man die Motorleistung beim Anhalten in der Art einer Rampe allmählich vermindern bis zum Stillstand. Beim Anfahren wird dann die Leistung allmählich wieder erhöht bis zum Maximalwert.

Aufgabe 4) Grundverhalten

Programmieren Sie die elementaren Verhaltensweisen des Rug Warrior beim Fahren:

Verhalten bei Kollision – Ansprechen der Bumper Ein mobiler Roboter muss sofort auf eine Kollision reagieren: Er kann sich zum Beispiel vom Kollisionspunkt entfernen und vor der Weiterfahrt seine Richtung ändern.

Verhalten bei Annäherung an ein Hindernis - IR-Sensoren Ein mobiler Roboter muss versuchen, Kollisionen zu vermeiden. Wenn er mit Sensoren ein Hindernis entdeckt, muss die Richtung und/oder die Geschwindigkeit verändert werden.

Programmieren Sie die obigen Verhaltensweisen. Testen Sie dann, indem Sie den Rug Warrior in dem aufgebauten Testbereich mit Hindernissen zufallsgesteuert umherfahren lassen. Dabei müssen dann Kollisionen vermieden werden bzw. richtig auf Kollisionen reagiert werden.

Aufgabe 5) Versteck suchen

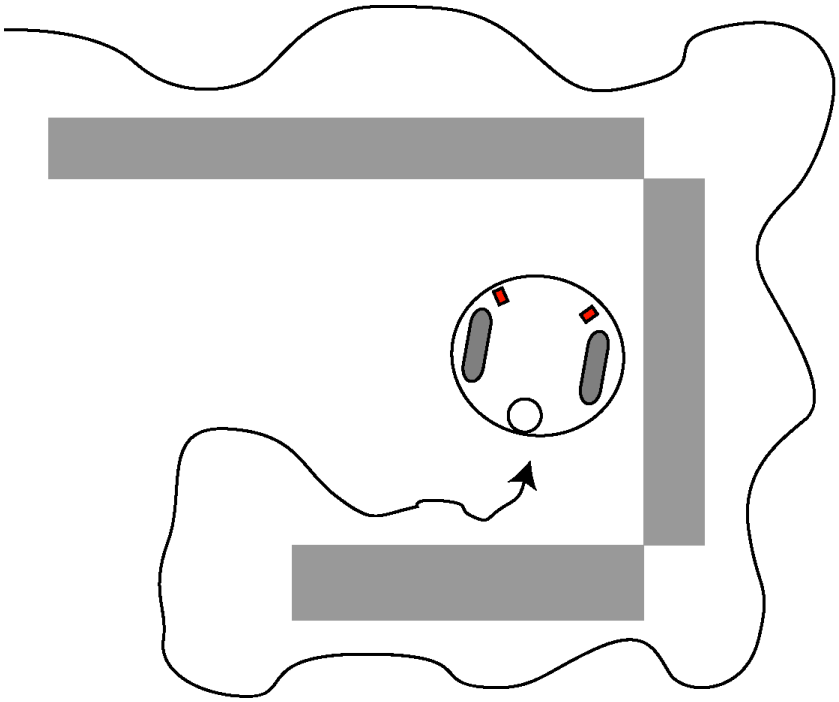
Programmieren Sie den Rug Warrior so, dass er im Raum umherfährt und die dunkelste Stelle sucht. Dabei soll er ein sicheres Grundverhalten zeigen, so wie es in der vorigen Aufgabe programmiert wurde. Nach ca. 2 Minuten soll er an einer möglichst dunklen Stelle stehen bleiben. Falls er danach ein lautes Geräusch detektiert, soll er wieder anfangen zu fahren und aus seinem Versteck hervorkommen. Diesmal soll er die hellste Stelle im Raum suchen, und dort nach ca. 2 Minuten stehen bleiben.

Aufgabe 6) Wandverfolgung (Zusatzaufgabe)

Programmieren Sie den Rug Warrior so, dass er einer Wandoberfläche folgen kann. Er soll der Wandoberfläche auch dann folgen, wenn diese mehrfach in irgendwelche Richtungen abknickt (s. Abb. unten). Auch dabei soll der Rug Warrior ein sicheres Grundverhalten zeigen.

Tipps

- Am Rug Warrior wird der Schalter nur auf Download gestellt, wenn der PCode heruntergeladen wird, nicht beim Herunterladen von C-Programmen!
- Geben Sie den Programmen aussagekräftige Namen!



- Lassen Sie die Programme beim Start eine aussagefähige Meldung auf das LCD ausgeben!
- Wenn das Programm eine Funktion `main()` enthält, wird diese nach einem Reset automatisch ausgeführt.
- Es ist praktisch, den Roboter erst nach dem Drücken eines Bumpers losfahren zu lassen. Dazu kann am Programmanfang eine Warteschleife programmiert werden.
- Während des Programmlaufes können Informationen über den Pieper und das LCD an die Entwickler gegeben werden.
- Benutzen Sie die Funktion `drive(int Translationsgeschw, int Rotationsgesc` wenn Sie die Motoren einfach einschalten und nach einer Wartezeit wieder ausschalten wollen.
- Sie können auch während der Wartezeiten, die mit `sleep()` oder `msleep()` realisiert werden, Aktionen durchführen, wenn Sie einen weiteren Prozess starten.

D. Die Robotersprache RAPID (ABB)

RAPID, die Robotersprache der ABB-Roboter, ist ein Beispiel für eine sorgfältig konstruierte mächtige Robotersprache. Die Dokumentation des Herstellers gibt einen sehr guten Überblick über die Eigenschaften dieser Sprache, speziell das „RAPID Referenzhandbuch - Überblick“ [11] und die „RAPID Referenzhandbuch System Datentypen und Routinen“ [10]. Wir wollen hier nur einige Beispiele zum Umgang mit den charakterstischen Datentypen und Funktionen geben.

Elementare Datentypen

Die elementaren Datentypen in RAPID sind die folgenden:

bool Wert kann nur TRUE und FALSE sein

num Numerische Werte, sowohl ganzzahlige als auch Dezimalzahlen

string Zeichenketten

Alle anderen Datentypen setzen sich aus diesen Datentypen zusammen.

Datentyp pos

Der Datentyp pos (Position) nimmt Positionen von Objekten auf und enthält drei numerische Daten.

pos = (num: x, num:y, num:z)

Hierbei sind x,y,z die Feldnamen, mit denen die drei Komponenten in der Programmierung angesprochen werden können und num ist ihr Datentyp. Verwendungsbeispiel:

```
VAR pos sp1;
VAR pos sp2;

sp1:=[20,10,55];
sp2:=sp1;
sp2.x:=sp1.x-40;
sp2.y:=sp2.y-20;
```

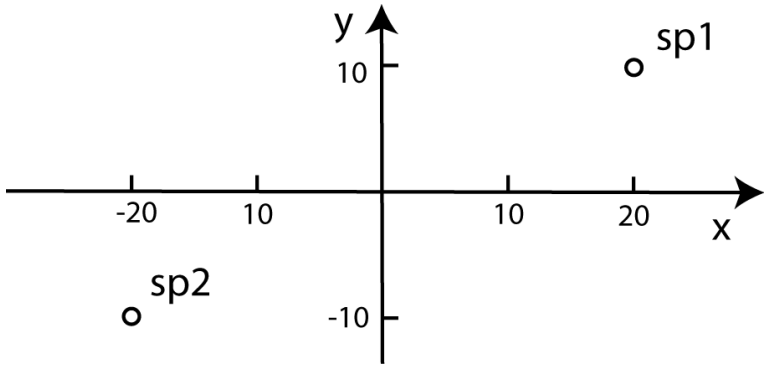


Abbildung D.1.: Eine Position kann durch Zugriff auf die kartesischen Komponenten verschoben werden.

Datentyp orient

Der Datentyp orient (Orientation) nimmt Orientierungen von Objekten auf und enthält vier numerische Daten. die Orientierung ist hier als Quaternion gespeichert, ein direkter Zugriff ist daher eher selten. Man kann aber diese Orientierung umwandeln in Eulerwinkel (Funktion EulerZYX) oder umgekehrt Eulerwinkel in den Datentyp orient (Quaternionendarstellung) umwandeln (Funktion OrientZYX).

Datentyp pose

Der Datentyp pose nimmt Transformationsmatrizen auf. Dazu enthält er einen Translationsanteil und einen Rotationsanteil:

pose = (pos: trans, orient: rot).

Es gibt Funktionen um Rechenoperationen mit diesen Matrizen auszuführen. Verwendungsbeispiel:

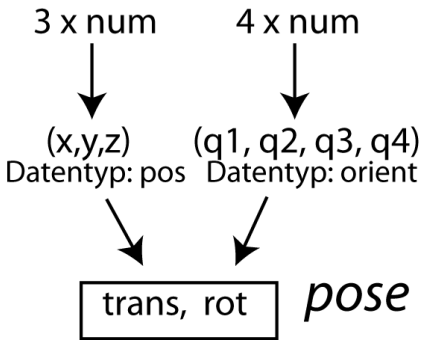


Abbildung D.2.: Struktur des Datentyps „pose“.

```
VAR pose t1;
VAR pose t2;
VAR pose t3;

t1.trans:=[0,0,0];
t1.rot:=OrientZYX(90,0,0);     ! t1 bewirkt eine Rotation

t2.trans:=[50,10,0];
t2.rot:=OrientZYX(0,0,0);      ! t2 bewirkt eine Translation

t3:=PoseMult(t1, t2);          ! t3 = t2*t1
                              ! enthält Rotation + Translation
```

Mit der Funktion PoseVect kann ein Positionsvektor mit einer Transformationsmatrix multipliziert werden. Beispiel: (Abb.D.3)

```
VAR pose t1;
VAR pos vektor1;
VAR pos vektor2;

t1.trans:=[0,0,0];
t1.rot:=OrientZYX(0,0,-90);     ! Matrix t1: -90 Grad um z-Achse

vektor1:=[30,10,0];
vektor2:=PoseVect(t1, vektor1);       ! Vektor2 ist -90 Grad
                                     ! gegen Vektor1 verdreht
```

Datentyp robtarget

Der Datentyp robtarget enthält alle Angaben, die der Roboter braucht, um einen Punkt anzufahren. Er setzt sich wie folgt zusammen:

robtarget = (pos: trans, orient: rot, confdata: robconf, extjoint: extax)

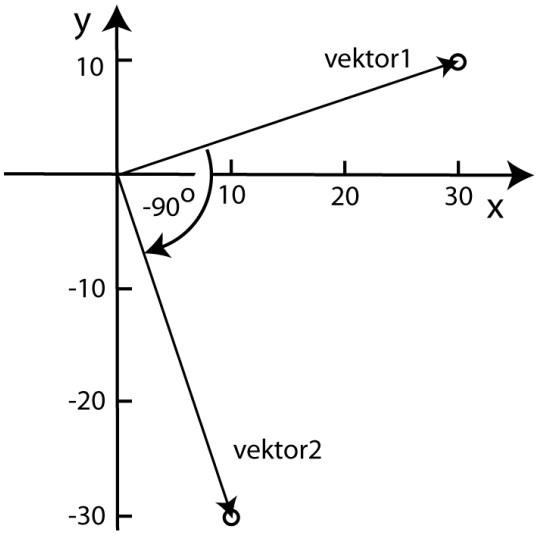


Abbildung D.3.: Ein Punktvektor kann durch Multiplikation mit einer Transformationsmatrix gedreht werden.

Die Datentypen pos und orient kennen wir schon. Der Datentyp confdata nimmt vier numerische Werte auf:

confdata: robconf = (num: cf1, num: cf4, num: cf6, num: cfx)

Diese Konfigurationswerte kennzeichnen, ein welchen Quadranten die Achsen 1,4,6 und eine externe Achse sich befinden. Erst damit ist die Stellung des Roboters eindeutig gekennzeichnet. Der Datentyp extjoint nimmt 6 numerische Werte auf, jeder beschreibt die Stellung einer externen Achse.

extjoint: extax = (num eax_a, num eax_b, num eax_c, num eax_d, num eax_e, num eax_f)

Falls die entsprechende externe Achse nicht vorhanden ist, steht hier der Wert 9E+09. Verwendungsbeispiel:

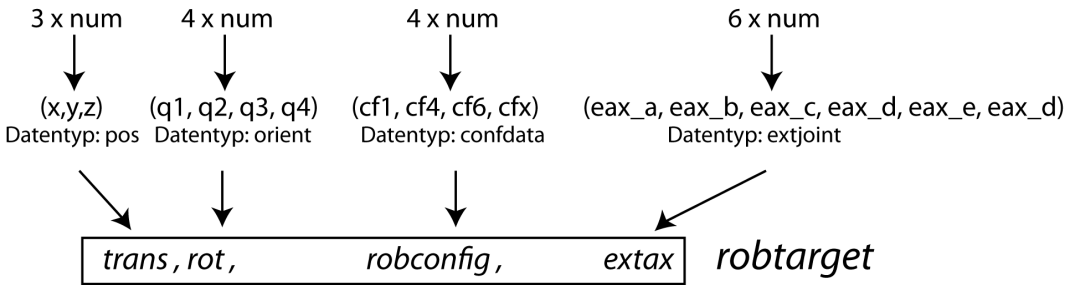


Abbildung D.4.: Struktur des Datentyps robtarget.

```
! Beispielcode 1 zu robtargets
VAR robtarget p1;
VAR robtarget p2;
```

```
p1.trans := [10,-20,0];
p2.trans := p1.trans;
p2.trans.x := p2.trans.x+20;
p2.rot := OrientZYX(0,0,45); ! Translationsanteil von p2 ist (30,-20,0)
                             ! Orientierung ist 0, 0, 45
                             ! robconf und extax sind noch nicht besetzt

! Mit der Funktion Offs (Offset=Verschiebung)
! kann ein robtarget im Raum verschoben werden
p2:=Offs(p2,0,5,0);          ! Translationsanteil von p2 ist (30,-15,0)
```

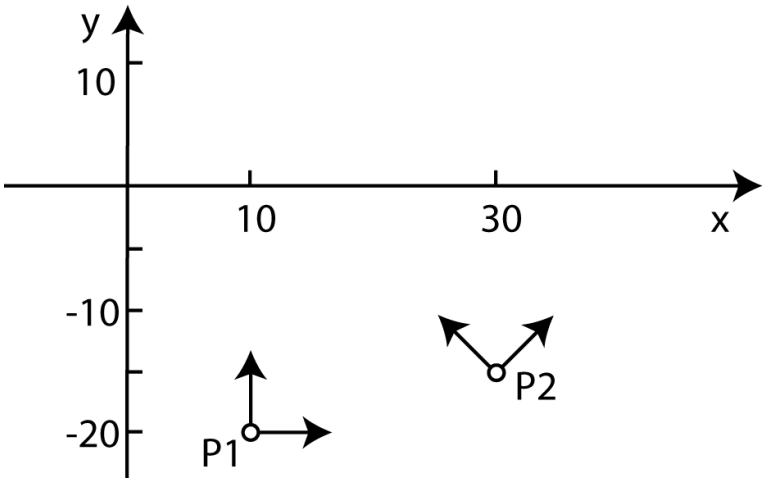


Abbildung D.5.: Robtargets p1 und p2 aus dem Beispielcode 1.

In einem abschließenden Beispiel zur Offline-Programmierung wird gezeigt, wie ein Roboter mit einer Schweißpistole (Arcweld gun) eine gewünschte Figur abfahren kann. dabei ist kein Teach-In nötig.

```
! Beispielcode 2 zu robtargets
VAR robtarget p1;
VAR robtarget p2;   ! eingeteachter Punkt
VAR robtarget vp;

p1:=p2;  ! Alle Komponenten zunächst
         ! von einem gültigen Punkt p2 (robtarget) übernehmen
         ! Roboterkonfiguration und externe Achswerte sind nun gültig

p1.rot:=OrientZYX(0,0,0);    ! Effektor wird nicht verdreht
p1.trans:=[0,0,0];           ! erster Bahnpunkt

vp:=Offs(p1,0,0,5);          ! Vorpunkt sitzt 5 mm über p1
MoveJ vp,v1000,z2,tAW_Gun\WObj=wWerkstueck
MoveL p1,v100,fine,tAW_Gun\WObj=wWerkstueck

p1.trans := [20,0,0];
```

```
MoveL p1,v100,fine,tAW_Gun\WObj=wWerkstueck

p1.trans := [20,10,0];
MoveL p1,v100,fine,tAW_Gun\WObj=wWerkstueck

p1.trans.x:=-p1.trans.x;
p1.trans.y:=-p1.trans.y
MoveL p1,v100,fine,tAW_Gun\WObj=wWerkstueck

p1.trans.y=0;
MoveL p1,v100,fine,tAW_Gun\WObj=wWerkstueck

p1.trans.x:=0;
MoveL p1,v100,fine,tAW_Gun\WObj=wWerkstueck
MoveL vp,v100,z2,tAW_Gun\WObj=wWerkstueck
```

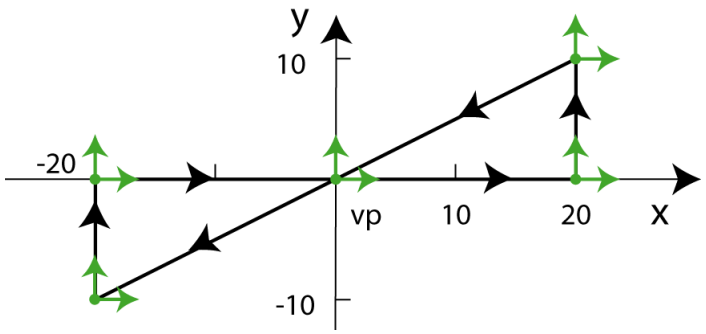


Abbildung D.6.: Zurückgelegter Weg und angefahrene robtargets aus dem Beispielcode 2.

Literaturverzeichnis

[1] Brillowski, K. *Einführung in die Robotik*, Shaker-Verlag, Aachen 2004

[2] Weber, W.: *Industrieroboter*, Methoden der Steuerung und Regelung, Fachbuchverlag Leipzig, München, Wien, 2002

[3] Craig, J.J.: *Introduction to Robotics*, Mechanics and control, Pearson Prentice Hall, 2005

[4] Tsai, L.-W.: *Robot Analysis*, The mechanics of serial and parallel Manipulators, John Wiley&sons, 1999

[5] Paul, R.P.: *Robot Manipulators: mathematics, programming and control*, MIT Press, Cambridge, London 1981

[6] Snyder, W.E.: *Computergesteuerte Industrieroboter*, VCH-Verlag, Weinheim, 1990

[7] Denavit, J., Hartenberg, R.S.: *A Kinematic Notation Lower-Pair Mechanisms based on Matrices*, ASME-Journal of Applied Mechanics, 1955, 215-222

[8] Siegert, H.J., Bocionek, S. *Robotik: Programmierung intelligenter Roboter*, Springer Verlag, Berlin, 1996

[9] Jones, A., Flynn, *Mobile Roboter*, Addison-Wesley, 1996

[10] ABB Robotics *RAPID Referenzhandbuch, Datentypen und Routinen*

[11] ABB Robotics *RAPID Referenzhandbuch, Überblick*

[12] Kai Oliver Arras, Roland Siegwart *Acht häufig gestellte Fragen an die mobile Robotik* Institut de Systèmes Robotique, Lausanne

[13] Daniel Ichbiah *Roboter - Geschichte, Technik, Entwicklung* Knesebeck Verlag, 2005

[14] Alois Knoll, Thomas Chistaller *Robotik* Fischer kompakt, 2003

[15] Wellenreuther, G., Zastrow, D., *Automatisieren mit SPS - Theorie und Praxis*, Vieweg und Teubner, 5.Aufl. 2011

[16] International Federation of Robotics (IFR), *World Robotics Survey: Industrial robots are conquering the world* , www.ifr.org

[17] Becker, N., *Automatisierungstechnik*, Vogel Fachbuch Kamprath-Reihe,Würzburg 2006

[18] Bergmann, J., *Automatisierungs- und Prozeßleittechnik*, mit Demonstrationssoftware, Fachbuchverlag Leipzig, 1999