4 Bewegungsprogrammierung

Zu den wichtigsten Aufgaben einer Robotersteuerung gehört die Bewegung des Roboters. Der Programmierer steuert dabei die Bewegungen des Industrieroboters mit Hilfe von speziellen Bewegungsanweisungen. Diese bilden auch das Hauptmerkmal, das die Robotersprachen von herkömmlichen Programmiersprachen für Computer, wie C oder Pascal, unterscheidet.

Bewegungsanweisungen

Je nach Steuerungsart lassen sich diese Bewegungsanweisungen in Befehle für einfache Punkt–zu–Punkt Bewegungen und in Befehle für Bahnbewegungen unterteilen. Während bei Bahnbewegungen der Endeffektor (z.B. Greifer oder Werkzeug) eine geometrisch genau definierte Bahn im Raum beschreibt (Gerade oder Kreisbogen) ist die Bewegungsbahn bei Punkt–zu–Punkt Bewegungen von der Kinematik des Roboters abhängig und daher nicht genau vorhersehbar. Beiden Bewegungsarten ist gemeinsam, daß die Programmierung von der aktuellen Position in eine neue Position erfolgt. Deshalb ist in einer Bewegungsanweisung im allgemeinen nur die Angabe der Zielposition notwendig (Ausnahme: Kreisbewegungen, siehe 4.3.4).

Die Koordinaten von Positionen können entweder textuell durch Eingabe von Zahlenwerten spezifiziert werden oder durch Anfahren mit dem Roboter und Abspeichern der Ist–Werte (Teachen). Dabei besteht jeweils die Möglichkeit, die Angaben auf verschiedene Koordinatensysteme zu beziehen.

Weitere Bewegungseigenschaften wie Geschwindigkeiten und Beschleunigungen sowie die Orientierungsführung können mittels Systemvariablen eingestellt werden. Das Überschleifen von Zwischenpunkten wird mittels optionaler Parameter in der Bewegungsanweisung initiiert. Zum Überschleifen muß ein Rechnervorlauf eingestellt sein.

4.1 Verwendung verschiedener Koordinatensysteme

Koordinatensystem

Um die Lage bzw. Orientierung eines Punktes im Raum angeben zu können, bedient man sich verschiedener Koordinatensysteme. Eine grundsätzliche Unterscheidung kann in achsspezifische und kartesische Koordinatensystem erfolgen:

achsspezifisch

Im **achsspezifischen Koordinatensystem** werden die Verschiebungen (bei translatorischen Achsen) bzw. Verdrehungen (bei rotatorischen Achsen) jeder Roboterachse angegeben. Bei einem 6–achsigen Knickarmroboter müssen also alle 6 Robotergelenkwinkel angegeben werden, um Lage und Orientierung eindeutig festzulegen (s. Abb. 16).

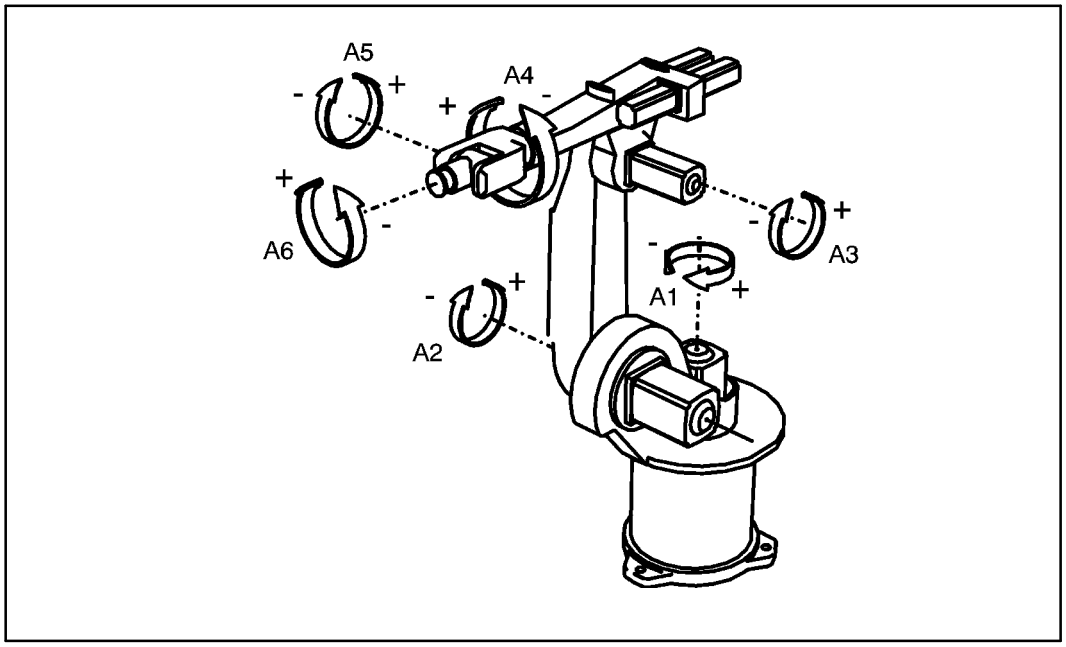


Abb. 16 Achsspezifisches Koordinatensystem eines 6–achsigen Knickarmroboters

Die Beschreibung einer achsspezifischen Stellung erfolgt in der KR C1 durch den vordefinierten Strukturtyp AXIS, dessen Komponenten je nach Achstyp die Bedeutung von Winkeln oder Längen hat.

Abschnitt 3.2.5 (AXIS)

Achsspezifische Positionen können nur in Verbindung mit PTP–Bewegungssätzen abgefahren werden. Wird eine Bahnbewegung mit einer achsspezifischen Roboterposition programmiert, so führt dies zu einem Fehlerfall.

Koordinatentransformation

Da der Mensch als Programmierer in kartesischen Koordinaten denkt, ist das Programmieren im achsspezifischen Koordinatensystem meist sehr unpraktisch. Die Steuerung bietet daher mehrere kartesische Koordinatensysteme zur Programmierung an, deren Koordinaten vor der Bewegungsausführung automatisch in das achsspezifische System transformiert werden (s. Abb. 17).

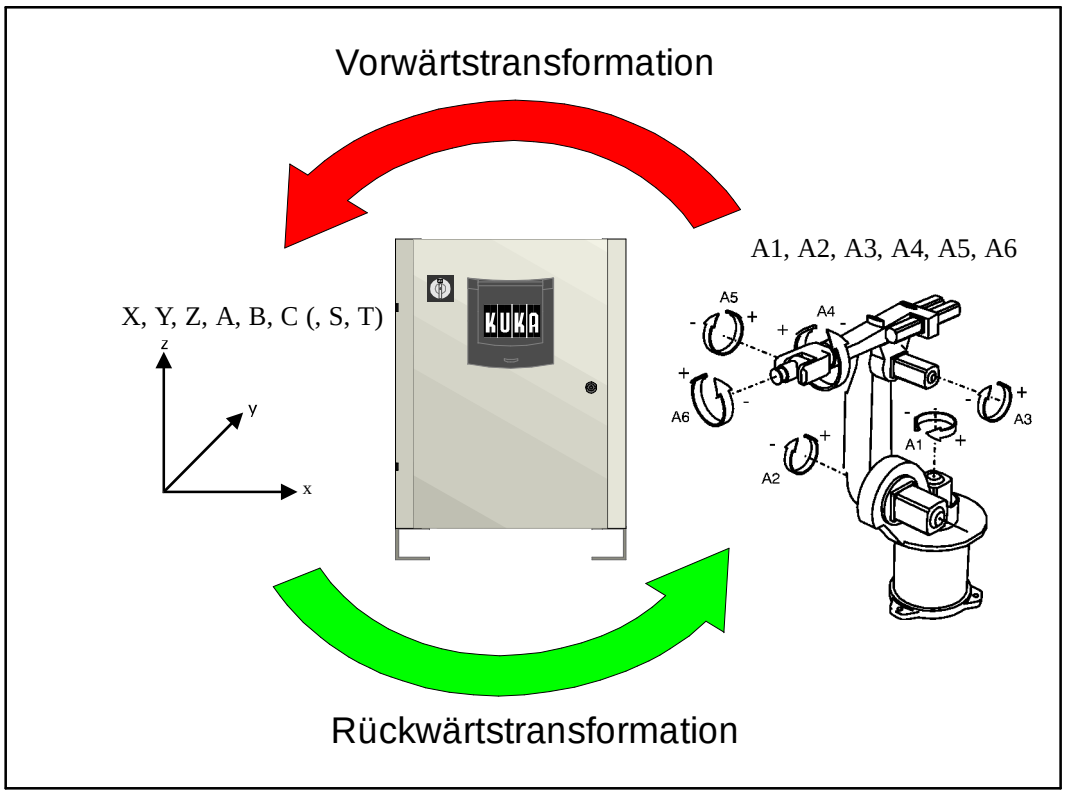


Abb. 17 Koordinatentransformation

kartesisch

In einem **kartesischen Koordinatensystem** stehen die 3 Koordinatenachsen X, Y und Z senkrecht aufeinander und bilden in dieser Reihenfolge ein Rechtssystem.

Die Lage eines Punktes im Raum ist im kartesischen Koordinatensystem eindeutig durch die Angabe der Koordinaten X, Y und Z bestimmt. Diese ergeben sich aus den translatorischen Distanzen jedes Koordinatenwertes zum Koordinatenursprung (s. Abb. 18).

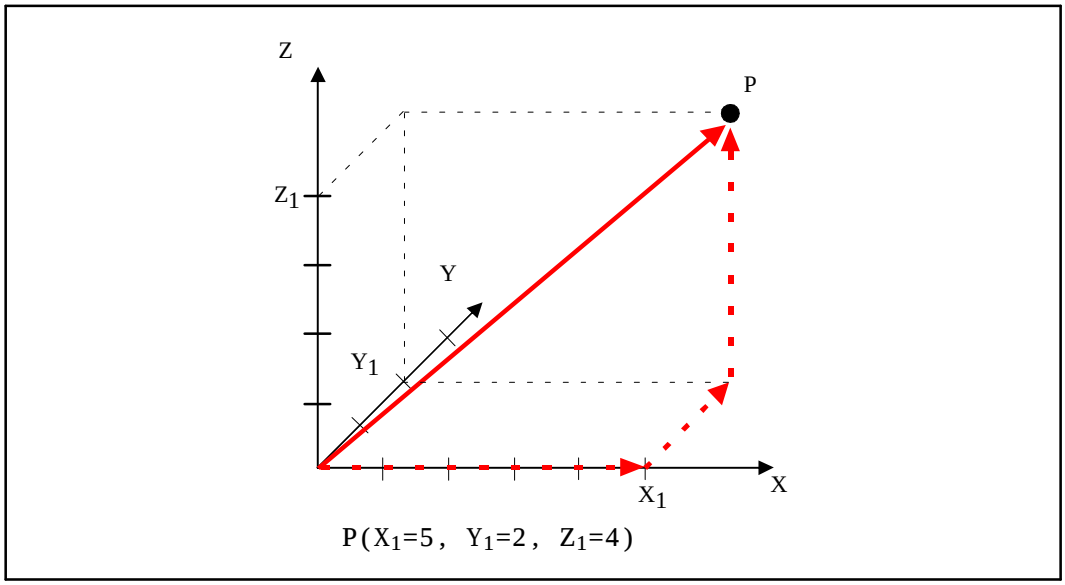


Abb. 18 Translatorische Beschreibung der Lage eines Punktes

Um jeden Punkt im Raum mit beliebiger Orientierung anfahren zu können, sind neben den drei translatorischen Werten noch drei rotatorische Angaben notwendig:

Die in der KR C1 mit A, B und C bezeichneten Winkel beschreiben Drehungen um die Koordinatenachsen Z, Y und X. Dabei ist die Reihenfolge der Rotationen einzuhalten:

1. Drehung um die Z-Achse um den Winkel A
2. Drehung um die neue Y-Achse um den Winkel B
3. Drehung um die neue X-Achse um den Winkel C

Diese Drehreihenfolge entspricht den aus der Luftfahrt bekannten roll-pitch-yaw-Winkeln (Rollen-Nicken-Gieren). Der Winkel C entspricht dabei dem Rollen, Winkel B dem Nicken und Winkel A dem Gieren (s. Abb. 19).

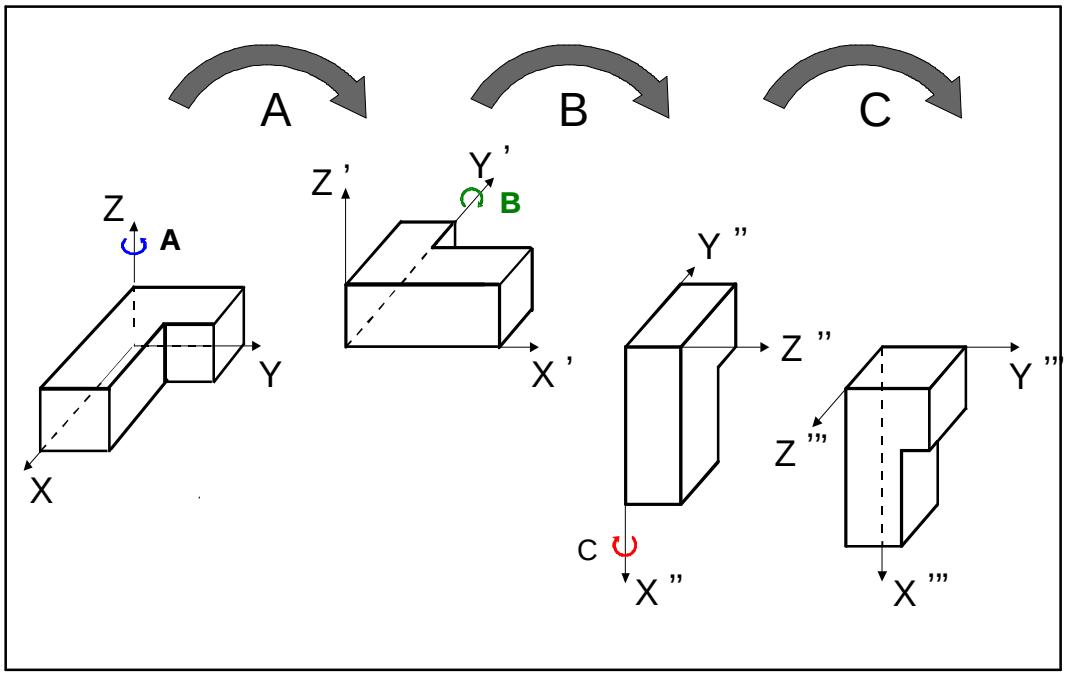
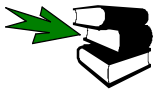


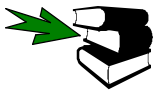
Abb. 19 Rotatorische Beschreibung der Orientierung eines Punktes

Mit den Translationen X, Y und Z sowie den Rotationen A, B und C läßt sich somit die Lage und Orientierung eines Punktes im Raum eindeutig beschreiben. In der KR C1 erfolgt dies mit der vordefinierten Struktur FRAME.



Abschnitt 3.2.5 (FRAME)

Bei Bahnbewegungen ist die Angabe von FRAME-Koordinaten immer eindeutig und ausreichend. Beim Punkt-zu-Punkt-Verfahren kann jedoch bei bestimmten Roboterkinematiken (z.B. 6-achsiger Knickarm) ein und derselbe Punkt (Lage und Orientierung) im Raum mit mehreren Achsstellungen angefahren werden. Mit den beiden weiteren Angaben S und T kann diese Mehrdeutigkeit behoben werden. In der KR C1 ist für ein um S und T erweitertes Frame die Struktur POS vorgesehen.



Abschnitt 3.2.5 (POS)
Abschnitt 4.2.2 (S und T)

In der KR C1 sind folgende kartesische Koordinatensysteme vordefiniert (Tab. 14 und Abb. 20):

Koordinatensystem	Systemvariable	Status
Weltkoordinatensystem	$WORLD	schreibgeschützt
Roboterkoordinatensystem	$ROBROOT	schreibgeschützt (in R1/$MASCHINE.DAT änderbar)
Werkzeugkoordinatensystem	$TOOL*	beschreibbar
Basis(Werkstück-)koordinatensystem	$BASE*	beschreibbar
* bei greiferbezogener Interpolation werden $TOOL und $BASE vertauscht (s.u.)		

Tab. 14 Vordefinierte Koordinatensysteme

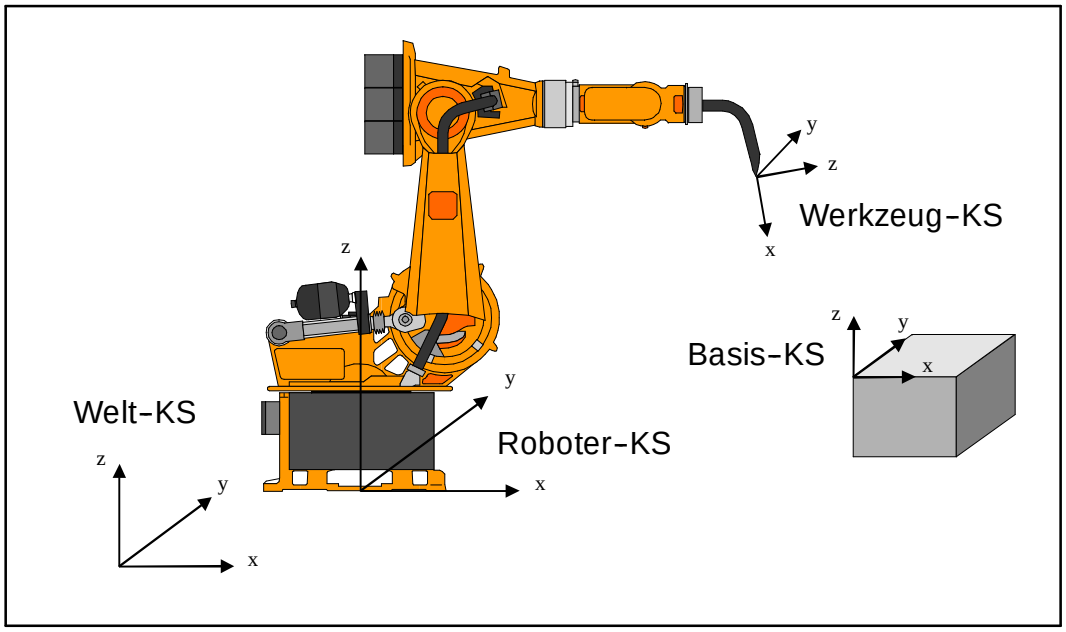


Abb. 20 Kartesische Koordinatensysteme bei Robotern

Weltkoordinatensystem

Das Weltkoordinatensystem ist ein ortsfestes (= bewegt sich bei einer Roboterbewegung nicht mit) Koordinatensystem, das als Ursprungskoordinatensystem für eine Roboterzelle dient. Es stellt somit das Bezugssystem sowohl für das Robotersystem als auch für die Zellenperipherie dar.

Roboterkoordinatensystem

Dieses Koordinatensystem befindet sich im Roboterfuß und dient als Bezugskoordinatensystem für den mechanischen Roboteraufbau. Es bezieht sich selbst wieder auf das Weltkoordinatensystem und ist bei Auslieferung mit ihm identisch. Mit $ROBROOT kann somit eine Verschiebung des Roboters zu $WORLD definiert werden.
Dadurch kann der Roboter innerhalb der Zelle verschoben werden, ohne daß die Koordinaten des Roboterprogramms geändert werden müssen.

Werkzeugkoordinatensystem

Das Werkzeugkoordinatensystem hat seinen Ursprung in der Werkzeugspitze. Die Orientierung ist dabei so gewählt, daß seine X–Achse mit der Werkzeugachse identisch ist und aus dem Werkzeug heraus zeigt. Bei einer Bewegung der Werkzeugspitze wird das Werkzeugkoordinatensystem mitbewegt.
Bei der Auslieferung liegt das Werkzeugkoordinatensystem im Roboterflansch (X–Achse ist identisch mit Achse 6). Es bezieht sich über die Transformation auf das Roboterkoordinatensystem.
Erfolgt ein Werkzeugwechsel, so kann nach Neuvermessung das ursprüngliche Programm weiter verwendet werden, da dem Rechner die Koordinaten der Werkzeugspitze bekannt sind.

Basiskoordinatensystem

Das Basiskoordinatensystem wird als Bezugssystem für die Beschreibung der Lage des Werkstücks herangezogen. Die Programmierung des Roboters erfolgt im Basiskoordinatensystem. Es hat als Bezugskoordinatensystem das Weltkoordinatensystem. Bei Auslieferung ist $BASE=$WORLD.
Das Basiskoordinatensystem ist in der KR C1 immer ortsfest. Durch eine Veränderung von $BASE können z.B. mehrere gleiche Werkstücke an verschiedenen Orten mit demselben Programm bearbeitet werden.

basisbezog. Interpolation

Bei der Interpolation der Bewegungsbahn berechnet die Robotersteuerung im Normalfall (feststehendes Werkstück, Werkzeug am Roboterflansch) die aktuelle Position ($POS_ACT) bezogen auf das $BASE–Koordinatensystem (s. Abb. 21).

$POS_ACT

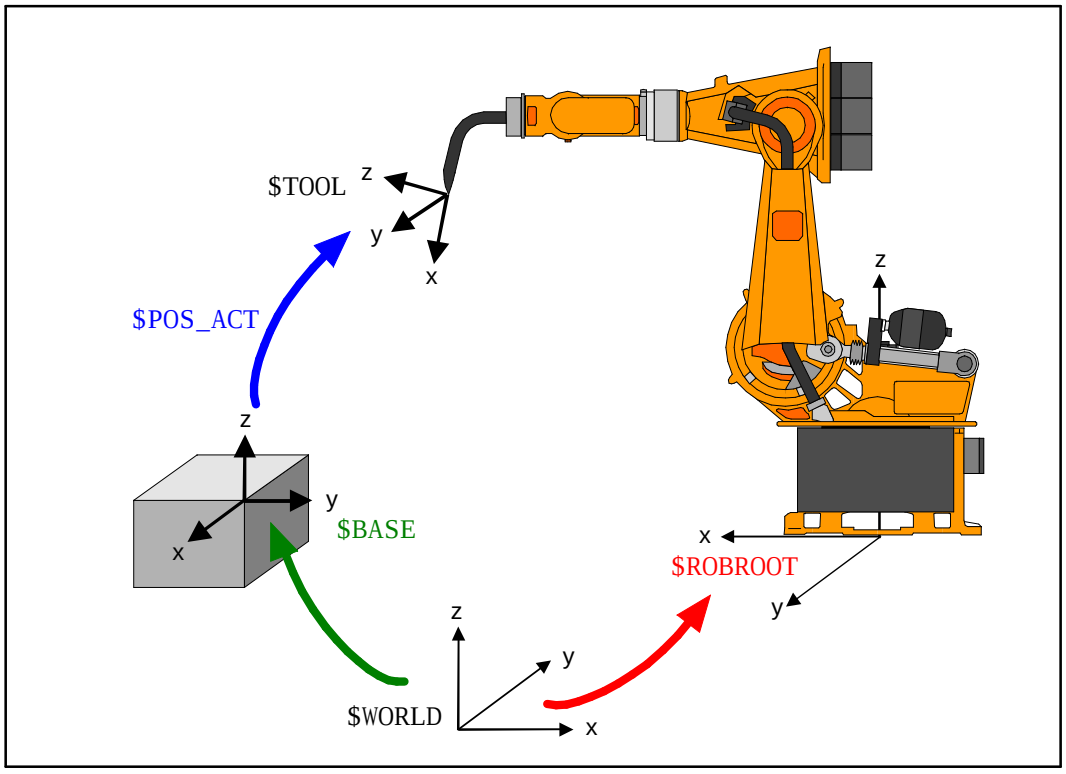


Abb. 21 Kinematische Kette bei basisbezogener Interpolation

feststehendes Werkzeug

In der industriellen Praxis geht man jedoch immer mehr dazu über, das Werkzeug (z.B. Schweißbrenner) fest im Raum zu verankern und das Werkstück selbst mittels eines geeigneten Greifers am feststehenden Werkzeug entlang zu führen.

Da Werkstück und Werkzeug die Position getauscht haben, enthält die Systemvariable $TOOL nun das Basiskoordinatensystem und $BASE das Werkzeugkoordinatensystem.

greiferbezog. Interpolation

Die Bewegung soll aber weiterhin bezüglich des Werkstücks erfolgen, weshalb die Interpolation der Bewegungsbahn jetzt im $TOOL-Koordinatensystem berechnet werden muß. Mit der Systemvariable $IPO_MODE können Sie deshalb die Interpolationsart definieren. Die Programmzeile

```
$IPO_MODE = #TCP
```

ermöglicht eine greiferbezogene Interpolation im $TOOL-Koordinatensystem. Die aktuelle Position $POS_ACT wird nun also bezüglich $TOOL berechnet (s. Abb. 22). Mit

```
$IPO_MODE = #BASE
```

setzen Sie die Interpolationsart wieder zurück auf die basisbezogene Interpolation für den Normalfall. Dies ist auch die Standardeinstellung beim Steuerungshochlauf.

$POS_ACT

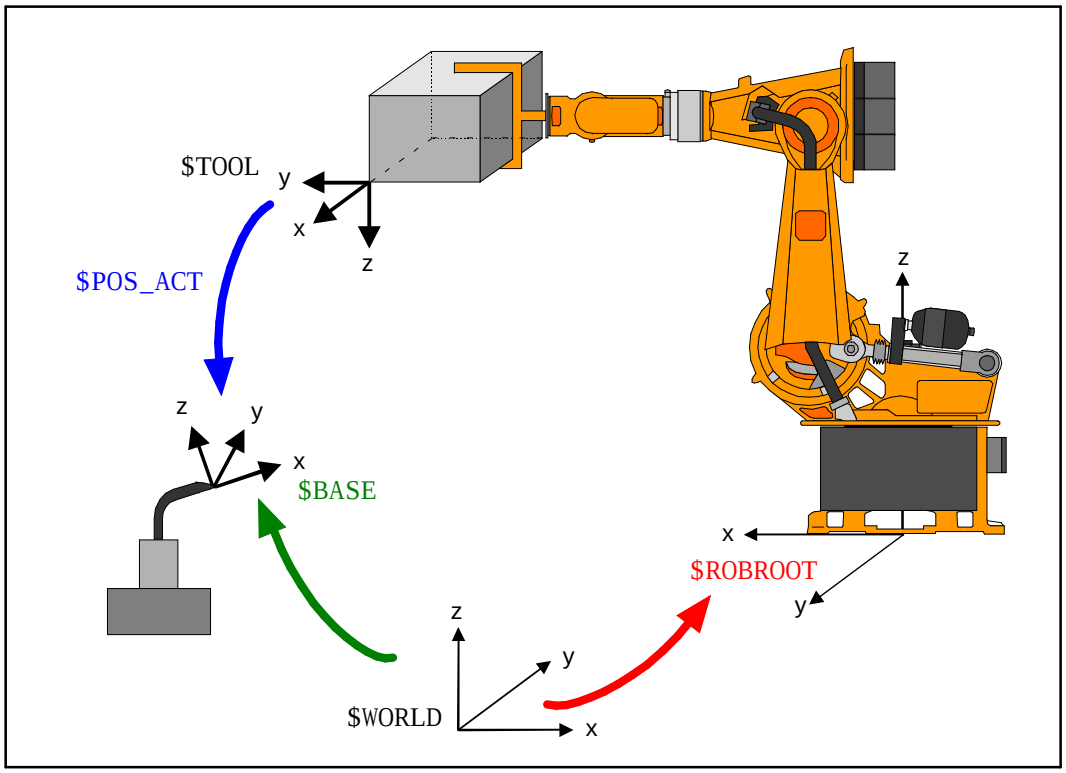


Abb. 22 Kinematische Kette bei greiferbezogener Interpolation

Beispiel 3.2 zur Verschiebung von Koordinatensystemen
Abschnitt 3.3.1.2 (geometrischer Operator)

4.2 Punkt-zu-Punkt Bewegungen

4.2.1 Geschwindigkeit und Beschleunigung

PTP

Die Punkt-zu-Punkt Bewegung (PTP) ist die schnellste Möglichkeit die Werkzeugspitze (Tool Center Point: TCP) von der momentanen Position zu einer programmierten Zielposition zu bewegen. Die Steuerung berechnet hierfür die erforderlichen Winkeldifferenzen für jede Achse.

Mit Hilfe der Systemvariablen

- $VEL_AXIS[Achsnummer]$ werden die maximalen achsspezifischen Geschwindigkeiten,

und mit

- $ACC_AXIS[Achsnummer]$ die maximalen achsspezifischen Beschleunigungen programmiert.

Alle Angaben erfolgen in Prozent, bezogen auf ein in den Maschinendaten definierten Höchstwert. Falls diese beiden Systemvariablen nicht für alle Achsen programmiert wurden, so erfolgt bei Ablaufen des Programms eine entsprechende Fehlermeldung.

Synchron - PTP

Die Bewegungen der einzelnen Achsen werden dabei so synchronisiert (Synchron-PTP), daß alle Achsen die Bewegung gleichzeitig starten und beenden. Dies bedeutet, daß nur die Achse mit dem längsten Weg, die sogenannte führende Achse, mit dem programmierten Grenzwert für Beschleunigung und Geschwindigkeit verfährt (s. Abb. 23). Alle anderen Achsen bewegen sich nur mit den Beschleunigungen und Geschwindigkeiten, die notwendig sind, um zum selben Zeitpunkt den Endpunkt der Bewegung zu erreichen, unabhängig von den in $VEL_AXIS[Nr]$ und $ACC_AXIS[Nr]$ programmierten Werten.

Da bei PTP-Bewegungen mit kartesischen Zielkoordinaten im allgemeinen nicht bekannt ist, welches die führende Achse ist, ist es in diesem Fall meist sinnvoll, die Beschleunigungs- und Geschwindigkeitswerte für alle Achsen gleichzusetzen.

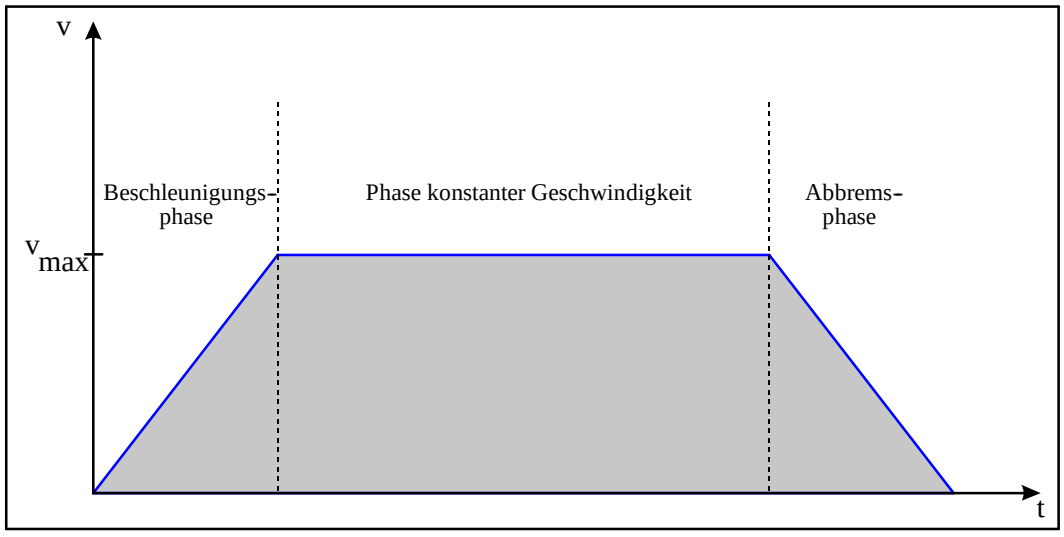


Abb. 23 Geschwindigkeitsprofil einer Achse bei Synchron-PTP Bewegung

Die Synchronisation führt zu einer stetigen Bewegungsbahn im Raum, die allerdings in Überschleifsätzen auf Änderung von Geschwindigkeit und Beschleunigung nicht bahngetreu reagiert. Sie vermindert aber gleichzeitig die mechanische Belastung des Roboters, da Motor- und Getriebemomente für alle Achsen mit kürzeren Verfahrwegen herabgesetzt werden.

Abschnitt 4.5

Nachteilig ist, daß bei der Synchron-PTP die Motorkennlinie für die führende Achse nicht optimal ausgenutzt wird und es in Sonderfällen sogar zu kurzzeitigen Überschreitungen der erlaubten Getriebe- und Motormomente im Konstantgeschwindigkeitsbereich kommen kann.

höheres Fahrprofil

Deshalb wird standardmäßig ein höheres Fahrprofil für PTP-Bewegungen verwendet. Mit dem höheren Fahrprofil wird bei **PTP-Einzelsätzen** und bei **PTP-Überschleifsätzen** zeitoptimal vom Start- zum Zielpunkt gefahren. Das heißt, mit den vorhandenen Getrieben und Motoren ist es nicht möglich schneller zu fahren. Die zulässigen Momente werden auf jedem Punkt der Bahn stets optimal ausgenutzt, dies gilt insbesondere auch in der konstanten Geschwindigkeitsphase. Werden Momente in dieser Konstantphase überschritten, wird die Geschwindigkeit in diesem Bereich abgesenkt (s. Abb. 24). Am Anfang der Beschleunigungs- und der Abbremsphase wird zudem mit einer deutlich höheren Rate beschleunigt bzw. verzögert als beim normalen Synchron-PTP Verfahren.

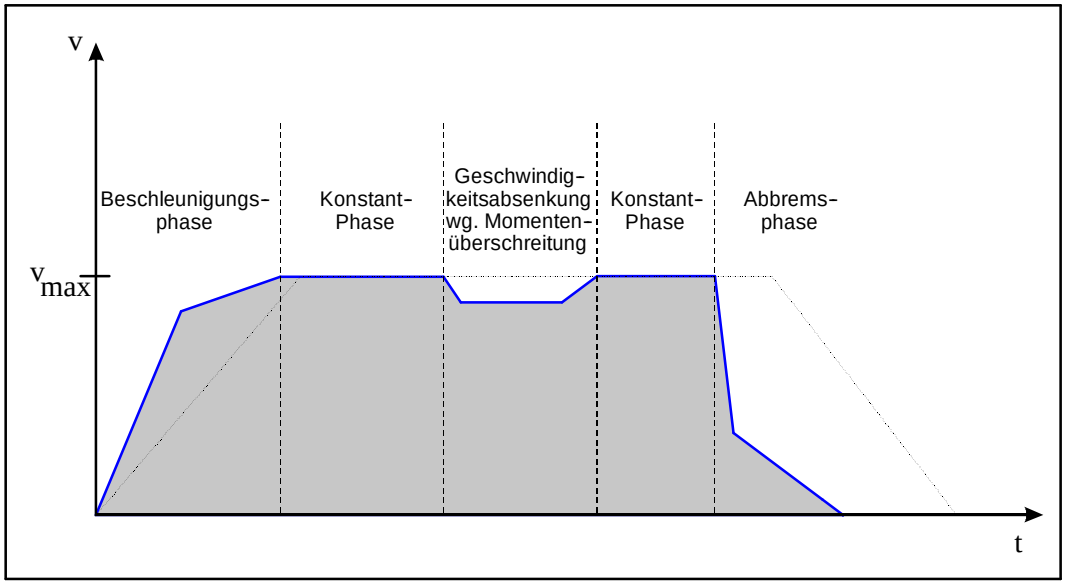


Abb. 24 Geschwindigkeitsprofil einer Achse bei Verwendung des höheren Fahrprofils

Eine Änderung der Geschwindigkeits- oder Beschleunigungswerte bewirkt auch bei Überschleifsätzen nur eine Änderung des Geschwindigkeitsprofils auf der Bahn. Die geometrische Kurve im Raum bleibt unverändert.

Bei Verwendung des höheren Fahrprofils können die Beschleunigungsgrenzwerte mit $ACC_AXIS[Nr] nicht mehr für jede einzelne Achse festgelegt werden. Stattdessen wird die angegebene Beschleunigung für Achse 1 nun auch für alle anderen Achsen verwendet. Die Beschleunigungswerte der anderen Achsen werden ignoriert. Im Gegensatz dazu sind die Geschwindigkeitszuweisungen weiter für alle Achsen gültig.

4.2.2 Bewegungsbefehle

Das folgende Programmbeispiel PTP_AXIS.SRC stellt prinzipiell das kleinste lauffähige KRL-Programm dar:

```
DEF PTP_AXIS()      ;der Name des Programms ist PTP_AXIS

$VEL_AXIS[1]=100    ;Festlegung der Achsgeschwindigkeiten
$VEL_AXIS[2]=100
$VEL_AXIS[3]=100
$VEL_AXIS[4]=100
$VEL_AXIS[5]=100
$VEL_AXIS[6]=100

$ACC_AXIS[1]=100    ;Festlegung der Achsbeschleunigungen
$ACC_AXIS[2]=100
$ACC_AXIS[3]=100
$ACC_AXIS[4]=100
$ACC_AXIS[5]=100
$ACC_AXIS[6]=100

PTP {AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

END
```

Zunächst werden in diesem Programm die Achsgeschwindigkeiten und -beschleunigungen festgelegt. Bevor eine Punkt-zu-Punkt Bewegung ausgeführt werden kann, müssen diese Zuweisungen erfolgt sein.

Danach verfährt der Roboter jede Achse in die in der Struktur AXIS spezifizierten Winkellagen. Also Achse 1 auf 0 _, Achse 2 auf -90 _, Achse 3 auf 90 _, Achse 4 auf 0 _, Achse 5 auf 0 _und Achse 6 auf 0 _. Der Roboter steht nun in der mechanischen Nullstellung, in die der Standardroboter auch zur Justage gebracht wird (s. Abb. 25).

Grundstellung

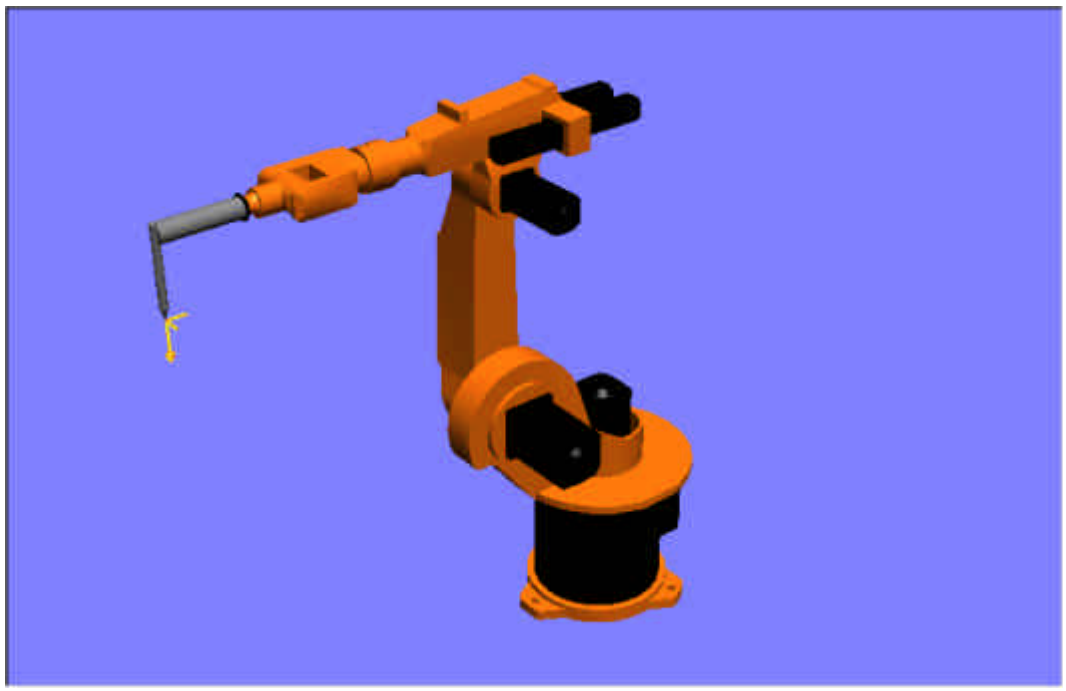


Abb. 25 Mechanische Nullstellung

Werden bei der Angabe der Achskoordinaten einzelne Komponenten weggelassen, so verfährt der Roboter nur die angegebenen Achsen, die anderen verändern ihre Lage nicht. Mit

PTP

```
PTP {A3 45}
```

wird also z.B. nur die Achse 3 auf 45 _gebracht. Zu beachten ist, daß es sich bei den Winkelangaben in der PTP–Anweisung um absolute Werte handelt. Der Roboter dreht die Achse also nicht um 45 _weiter, sondern verfährt auf die absolute 45 _–Lage der Achse.

Zum relativen Verfahren dient die Anweisung PTP_REL. Um also z.B. die Achsen 1 und 4 um je 35 _zu verdrehen, programmieren Sie einfach:

PTP_REL

```
PTP_REL {A1 35,A4 35}
```

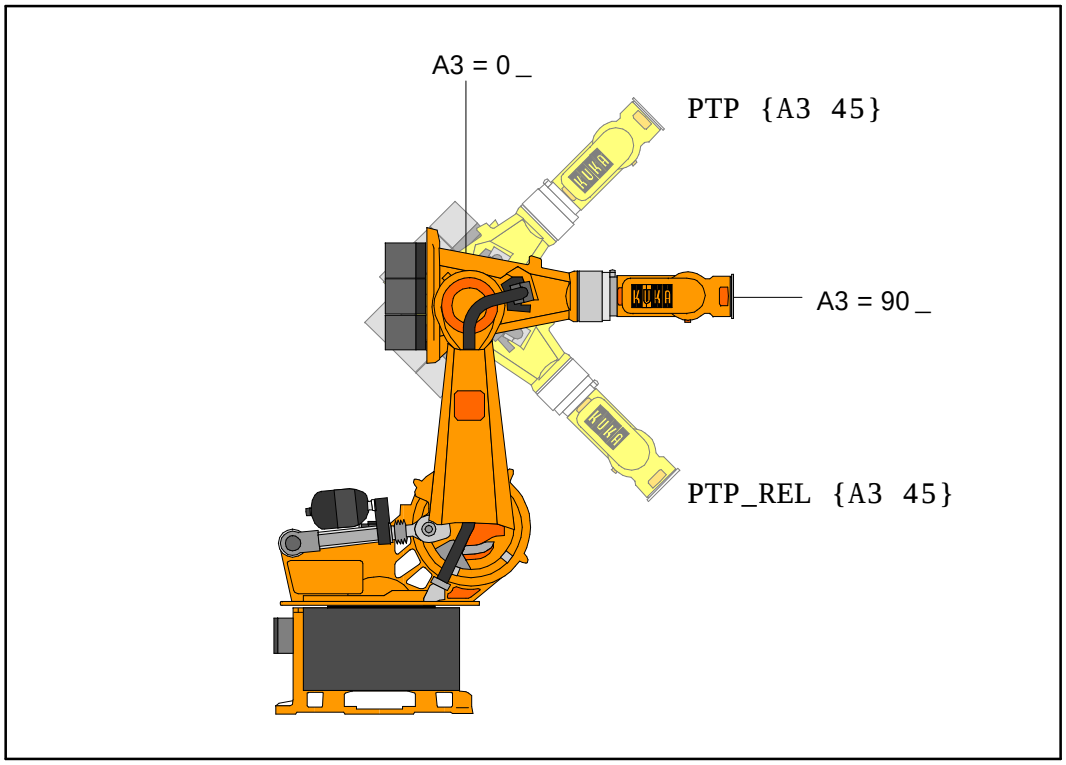


Abb. 26 Unterschied zwischen absoluten und relativen achsspezfischen Koordinaten

Beim relativen Verfahren ist allerdings zu beachten, daß eine während der Ausführung gestoppte Bewegung nicht wieder problemlos fortgesetzt werden kann. Die Steuerung kann nach einem Neustart und erneuter Satzanwahl bzw. Wechsel der Programmlaufart den bereits zurückgelegten Weg nicht berücksichtigen und wird die programmierte Relativdistanz wieder komplett abfahren, was schließlich zu einem falschen Endpunkt führt.

Das Verfahren in achsspezifischen Koordinaten ist meistens jedoch unpraktisch, da der Mensch im kartesischen Raum denkt und handelt. Dazu dient die Angabe der kartesischen Koordinaten mittels einer POS-Struktur, wie sie im folgenden Beispiel verwendet wird:


```
DEF PTP_POS()

$BASE = $WORLD      ;Setzen des Basiskoordinatensystems
$TOOL = $NULLFRAME  ;Setzen des Werkzeugkoordinatensystems

$VEL_AXIS[1]=100    ;Festlegung der Achsgeschwindigkeiten
$VEL_AXIS[2]=100
$VEL_AXIS[3]=100
$VEL_AXIS[4]=100
$VEL_AXIS[5]=100
$VEL_AXIS[6]=100

$ACC_AXIS[1]=100    ;Festlegung der Achsbeschleunigungen
$ACC_AXIS[2]=100
$ACC_AXIS[3]=100
$ACC_AXIS[4]=100
$ACC_AXIS[5]=100
$ACC_AXIS[6]=100

PTP {POS:X 1025,Y 0,Z 1480,A 0,B 90,C 0,S 'B 010',T 'B 000010'}

END
```

Zu beachten ist jetzt, daß bei Zielpunktangabe in kartesischen Koordinaten neben den Geschwindigkeits- und Beschleunigungsangaben auch zwingend das Basiskoordinatensystem und das Werkzeugkoordinatensystem definiert sein müssen.

Koordinatensysteme

In unserem Fall wurde das Basiskoordinatensystem ($BASE) gleich dem Weltkoordinatensystem ($WORLD) gesetzt, welches standardmäßig im Roboterfuß ($ROBROOT) liegt. Das Werkzeugkoordinatensystem ($TOOL) wurde mit dem Nullframe ($NULLFRAME = {FRAME: X 0, Y 0, Z 0,A 0,B 0,C 0}) besetzt, was bedeutet, daß sich alle Angaben auf den Flanschmittelpunkt beziehen. Der Werkzeugmittelpunkt (Tool Center Point: TCP) liegt also sozusagen im Flanschmittelpunkt. Falls ein Werkzeug angeflanscht ist, müßten die Werte entsprechend korrigiert werden. Dazu sei auf die Unterlagen zum Vermessen von Werkzeugen verwiesen.

Mit der obigen PTP-Anweisung verfährt der Roboter nun so, daß im Endpunkt der Bewegung der TCP 1025 mm in x-Richtung, 0 mm in y-Richtung und 1480 mm in z-Richtung vom Roboterfuß verschoben ist. Die Angaben A, B und C definieren die Orientierung des TCP. Status S und Turn T definieren die Stellung der Achsen.

Sofern Sie das Beispiel an einem KR6 - Roboter testen, erhalten Sie das gleiche Ergebnis, wie im vorherigen Beispiel. Der Roboter fährt in die mechanische Nullstellung. Die beiden Anweisungen sind also für diesen Robotertyp identisch.

Auch bei Zielpunktangabe in kartesischen Koordinaten können wieder einzelne Komponenten der Geometrieangabe weggelassen werden. Die Anweisung

```
PTP {Z 1300, B 180}
```

bewirkt demnach eine Bewegung des TCP in Richtung der z-Achse auf die Absolutposition 1300 mm und ein "Nicken" des TCP um 180 _.

Zum relativen Verfahren des Roboters verwendet man wieder den PTP_REL-Befehl. Mit

```
PTP_REL {Z 180, B -90}
```

kann man demnach den Roboter wieder in seine ursprünglich Position zurückbringen. Zu beachten ist wieder, daß Relativbewegungen nach Unterbrechung nicht erneut angewählt werden dürfen.

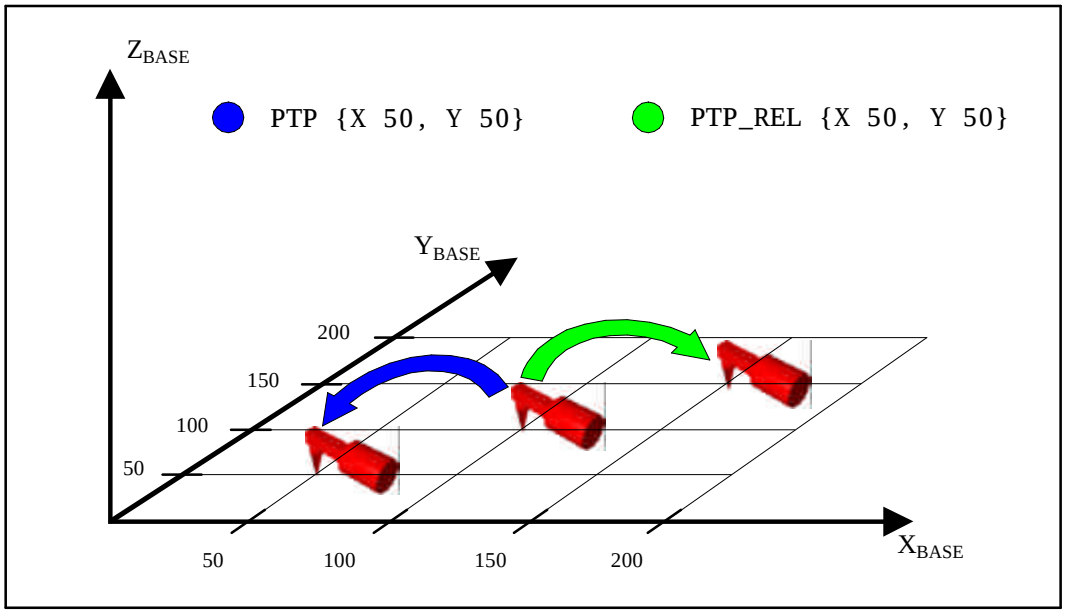


Abb. 27 Unterschiedzwischenabsolutenundrelativenkartesischen Koordinaten

Bei kartesischen Koordinaten ist es möglich, eine Frameverknüpfung mittels des geometrischen Operators direkt in der Bewegungsanweisung durchzuführen. Dadurch kann man z.B. eine Verschiebung zum Basiskoordinatensystem initiieren, ohne die Variable $BASE zu ändern. Man kann sich damit also beliebig viele eigene Basen (max. 16) für die Programmierung definieren.

Die Basisverschiebung mit Hilfe des Doppelpunkt-Operators hat außerdem einen entscheidenden Vorteil gegenüber einer Neubesetzung von $BASE:

Die Verschiebung erfolgt im Bewegungssatz, während eine $BASE-Besetzung irgendwann vor dem Bewegungssatz erfolgen muß. Dadurch ist auch bei Programmstop und nachfolgender Satzanwahl immer die richtige Basis für die Bewegung angewählt.

Eine mehrfache Neubesetzung von $BASE, wie in folgender Sequenz,

```
¼
$BASE = $WORLD
¼
PTP POS_1
$BASE = {X 100,Y -200,Z 1000,A 0, B 180,C 45)
PTP POS_2
¼
```

würde dagegen nach Abbruch des POS_2-Bewegungssatzes und Neuanwahl des POS_1-Satzes zu einem falschen Zielpunkt führen, da jetzt auch für den POS_1-Bewegungssatz die neue Basis herangezogen würde. Gleiches passiert übrigens auch schon bei einem Stop des ersten Bewegungssatzes, wenn ein entsprechender Rechnervorlauf eingestellt ist.

Aus diesem Grund sollten, wenn möglich, $BASE und $TOOL nur einmal, z.B. im Initialisierungsteil des Programms, besetzt werden. Weitere Verschiebungen können dann mit dem geometrischen Operator vorgenommen werden.

Im folgenden Beispiel wird im zweiten PTP-Befehl eine Verschiebung der Zielpunktkoordinaten um 300 mm in X-Richtung, -100 mm in Y-Richtung sowie eine 90 _-Drehung um die Z-Achse vorgenommen.

```
DEF   FR_VERS ( )

;--------- Deklarationsteil ---------
EXT   BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL AXIS HOME       ;Variable HOME vom Typ AXIS
DECL FRAME BASE1    ;Variable BASE1 vom Typ FRAME

;---------- Initialisierung ---------
BAS (#INITMOV,0 )    ;Initialisierung von Geschwindigkeiten,
                     ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
BASE1={FRAME: X 300,Y -100,Z 0,A 90,B 0,C 0}

;------------- Hauptteil ------------
PTP  HOME ; SAK-Fahrt
; Bewegung bezueglich des $BASE-Koordinatensystems
PTP  {POS: X 540,Y 630,Z 1500,A 0,B 90,C 0,S 2,T 35}
; Bewegung bezueglich des um BASIS1 verschobenen $BASE-KS
PTP  BASE1:{POS: X 540,Y 630,Z 1500,A 0,B 90,C 0,S 2,T 35}
PTP  HOME
END
```

In diesem Beispiel werden außerdem die notwendigen Belegungen der Geschwindigkeiten, Beschleunigungen sowie $BASE- und $TOOL-Koordinatensystemen nicht mehr "von Hand" vorgenommen. Stattdessen wird das standardmäßig vorhandene Basis-Packet "BAS.SRC" hierzu verwendet. Dazu muß es zunächst mit der EXT-Anweisung dem Programm bekannt gemacht werden.

Mit der Initialisierungsanweisung

INI `BAS (#INITMOV, 0)`

werden schließlich alle wichtigen Systemvariablen mit Standardwerten besetzt.

Abschnitt 8.1

SAK

Bevor ein Programm bearbeitet werden kann, muß zunächst Satzkoinzidenz (SAK), d.h. Übereinstimmung von aktueller Roboterposition und programmierter Position, hergestellt werden. Da die SAK-Fahrt keine programmierte, ausgetestete Bewegung darstellt, muß sie durch Gedrückthalten der Starttaste (Totmann-Funktion) und mit automatisch reduzierter Geschwindigkeit durchgeführt werden. Bei Erreichen der programmierten Bahn stoppt die Bewegung und das Programm kann mit erneutem Betätigen der Starttaste fortgesetzt werden.

Im "Automatik Extern"-Betrieb wird keine SAK-Fahrt durchgeführt!

Als erste Bewegungsanweisung empfiehlt sich daher eine "Home"-Fahrt, bei der der Roboter in eine eindeutig definierte und unkritische Ausgangsstellung verfahren wird, in der dann auch Satzkoinzidenz hergestellt wird. In diese Lage sollte der Roboter auch am Ende des Programms wieder gebracht werden.

S und T

Die Angaben S und T in einer POS-Angabe dienen dazu, um aus mehreren möglichen Roboterstellungen für ein und dieselbe Position im Raum (aufgrund der Kinematik-Singularitäten), eine ganz bestimmte, eindeutig definierte Stellung auszuwählen.

Bei der ersten Bewegungsanweisung ist es im Fall der Verwendung kartesischer Koordinaten daher sehr wichtig auch S und T zu programmieren, damit eine eindeutige Ausgangslage definiert wird. Da bei Bahnbewegungen (s. 4.3) S und T nicht berücksichtigt werden, sollte also auf jeden Fall die erste Bewegungsanweisung eines Programms (Home-Fahrt) eine vollständige PTP-Anweisung mit Angabe von Status und Turn sein (oder vollständige PTP-Anweisung mit Achskoordinaten).

In den nachfolgenden PTP-Anweisungen können nun die Angaben S und T entfallen, sofern eine bestimmte Achsstellung, z.B aufgrund von Hindernissen, nicht notwendig ist. Der Roboter verfährt in diesem Fall immer auf dem kürzesten axialen Weg, welcher aufgrund der einmaligen Programmierung von S und T im ersten PTP-Satz bei verschiedenen Programmläufen auch immer der gleiche ist.

Status und Turn erfordern beide Integerangaben, die zweckmäßigerweise in binärer Form gemacht werden sollten.

Turn

Die Erweiterung einer kartesischen Positionsangabe um die Turn-Angabe ermöglicht es auch Achswinkel, die größer +180 _bzw. kleiner -180 _sind, ohne besondere Verfahrstrategie (z.B. Zwischenpunkte) anfahren zu können. Die einzelnen Bits bestimmen bei rotatorischen Achsen das Vorzeichen des Achswertes in folgender Form:

Bit x = 0: Winkel der Achse x ³ 0 _
Bit x = 1: Winkel der Achse x < 0 _

Wert	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A6 ³ 0 _	A5 ³ 0 _	A4 ³ 0 _	A3 ³ 0 _	A2 ³ 0 _	A1 ³ 0 _
1	A6 < 0 _	A5 < 0 _	A4 < 0 _	A3 < 0 _	A2 < 0 _	A1 < 0 _

Tab. 15 Bedeutung der Turn-Bits

Also bedeutet eine Angabe T 'B 10011', daß die Winkel der Achsen 1, 2 und 5 negativ sind, die Winkel der Achsen 3, 4 und 6 dagegen positiv (alle höherwertigen 0-Bits können weggelassen werden).

Status

Mit dem Status S werden Mehrdeutigkeiten in der Achsstellung gehandelt (s. Abb. 28). S ist daher abhängig von der jeweiligen Roboterkinematik.

Mehrdeutigkeiten

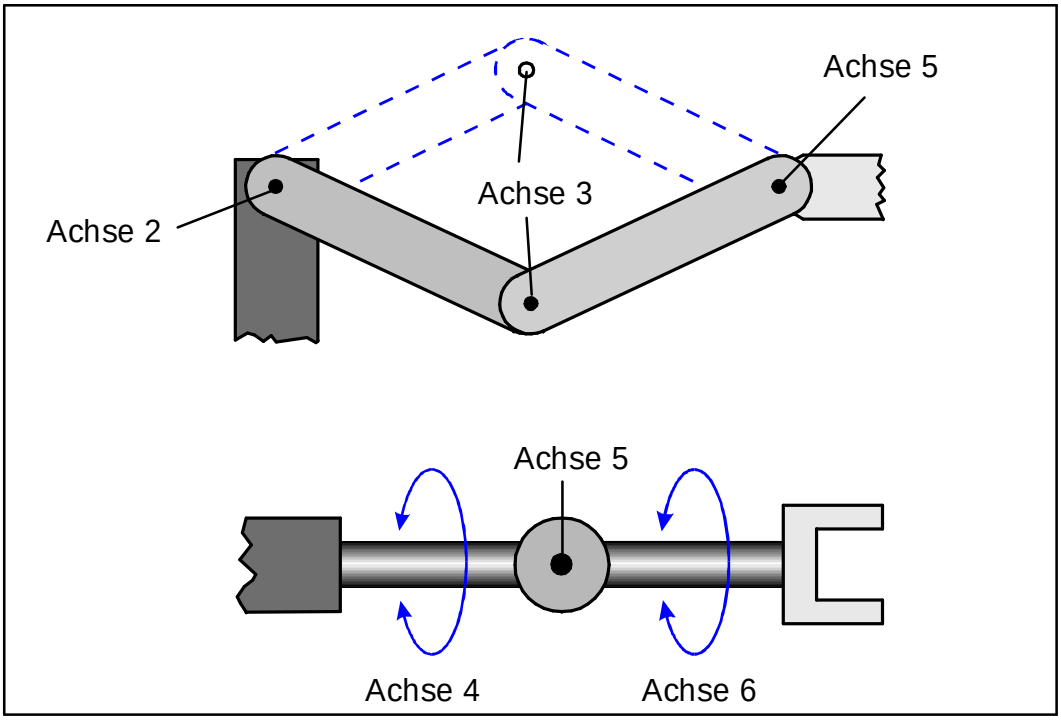


Abb. 28 Beispiele für mehrdeutige Roboterkinematiken

Die Bedeutung der einzelnen Bits ist:

Bit 0: Position des Handwurzelpunktes (Grundbereich/Überkopfbereich)
Bit 1: Armkonfiguration
Bit 2: Handkonfiguration

Die Bits werden für alle 6achsigen Knickarmroboter nach folgender Tabelle gesetzt:

Wert	Bit 2	Bit 1	Bit 0
0	0 _ £ A5 < 180 _ A5 < -180 _	A3 < f (f hängt vom Robotertyp ab)	Grundbereich
1	-180 _ £ A5 < 0 _ A5 ³ 180 _	A3 ³ f (f hängt vom Robotertyp ab)	Überkopfbereich

Tab. 16 Status-Bits für 6achsige Knickarmroboter

Anschaulich kann man sich den Grund-/Überkopfbereich kartesisch vorstellen. Dazu werden folgende Begriffe definiert:

Handwurzelpunkt: Schnittpunkt der Handachsen

A1-Koordinatensystem: Steht die Achse 1 auf 0 _, ist es mit dem $ROBROOT-Koordinatensystem identisch. Bei Werten ungleich 0 _wird es mit der Achse 1 mitbewegt.

Damit läßt sich der Grund-/Überkopfbereich wie folgt definieren:

- Ist der x-Wert des Handwurzelpunktes, ausgedrückt im A1-Koordinatensystem, positiv, befindet sich der Roboter im Grundbereich.
- Ist der x-Wert des Handwurzelpunktes, ausgedrückt im A1-Koordinatensystem, negativ, befindet sich der Roboter im Überkopfbereich.

Bit 1 gibt die Armstellung an. Das Setzen des Bits ist abhängig vom jeweiligen Robotertyp. Bei Robotern, deren Achsen 3 und 4 sich schneiden, gilt: Bit 1 hat den Wert 0, wenn die Achse 3 < 0 _ist, ansonsten ist Bit 1 = 1. Bei Robotern mit einem Offset zwischen Achse 3 und 4 (z.B. KR 30, s. Abb. 29), ist die Winkellage, in der sich der Wert von Bit 1 ändert, von der Größe dieses Offsets abhängig.

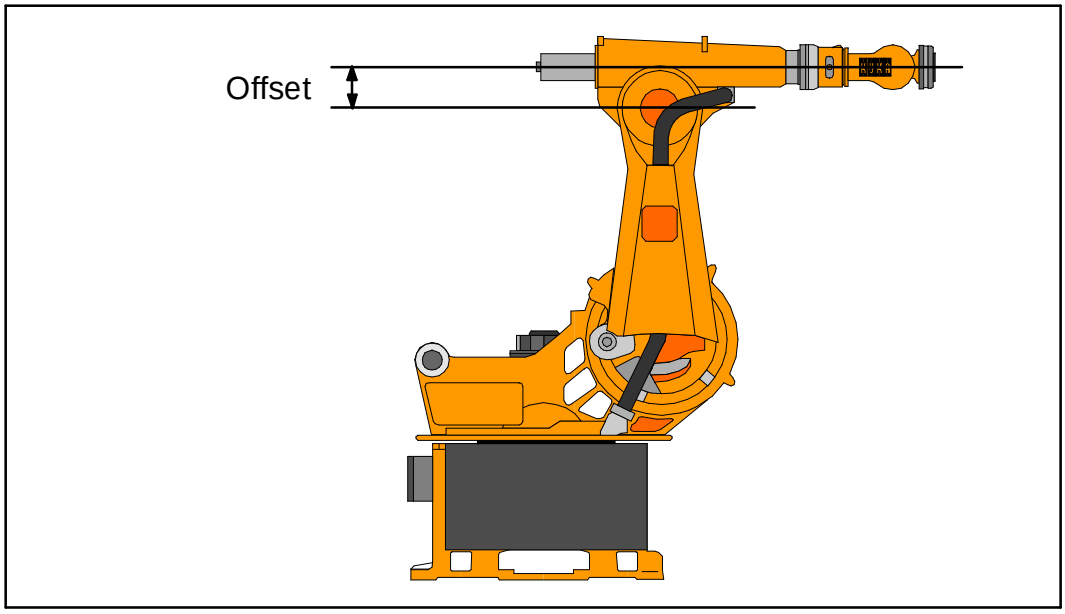


Abb. 29 Offset zwischen Achse 3 und 4 bei einem KR 30

In Abb. 30 sind die Auswirkungen der Status-Bits auf die Roboterkonfiguration dargestellt. Derselbe Punkt im Raum wurde mit vier unterschiedlichen Roboterstellungen angefahren. In der ersten Stellung befindet sich der Roboter in Grundstellung, Achse 5 hat einen Wert von ca. 45 _, Achse 3 ca. 80 _.

Zur zweiten Roboterkonfiguration ist kaum ein Unterschied zu erkennen. Es wurde lediglich die Achse 4 um 180 _gedreht und die anderen Achsen entsprechend nachgeführt. Während also die Armkonfiguration gleich blieb, hat sich die Handkonfiguration geändert: Achse 5 hat nun ca. -45 _, Status-Bit 2 ist folglich 1.

Von Stellung 2 nach 3 ändert sich nun die Armkonfiguration. Achse 3 dreht auf eine Winkellage von ca. -50 _, das Status-Bit 1 nimmt den Wert 0 an.

In der vierten Stellung befindet sich der Roboter schließlich in der Überkopfposition. Dazu wurde insbesondere Achse 1 um 180 _gedreht. Status-Bit 0 wird zu 1.

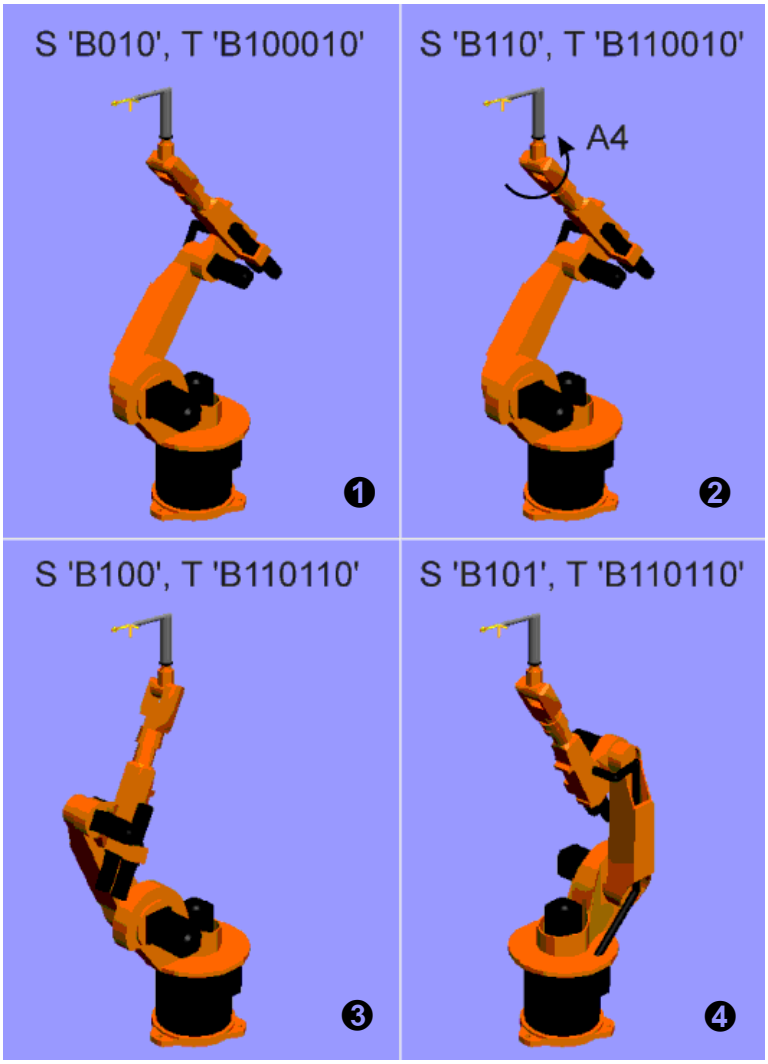


Abb. 30 Auswirkungen der Status–Bits auf die Roboterstellung

NOTIZEN:

4.3 Bahnbewegungen

4.3.1 Geschwindigkeit und Beschleunigung

Im Gegensatz zu PTP–Bewegungen sind bei Bahnbewegungen nicht nur Start– und Zielposition vorgegeben. Es wird zusätzlich gefordert, daß die Werkzeugspitze des Roboters sich auf einer linearen oder kreisförmigen Bahn zwischen diesen Punkten bewegt.

Deshalb beziehen sich die anzugebenden Geschwindigkeiten und Beschleunigungen nun nicht mehr auf die einzelnen Achsen, sondern auf die Bewegung des TCP. Die Werkzeugspitze wird dadurch mit genau definierter Geschwindigkeit bewegt. Die Geschwindigkeiten und Beschleunigungen müssen für die Translation, den Schwenkwinkel und den Drehwinkel programmiert werden. Tab. 17 gibt einen Überblick über die zu programmierenden Systemvariablen und deren Einheit.

	Variablen-name	Datentyp	Einheit	Funktion
Geschwindigkeiten	$VEL.CP	REAL	m/s	Bahngeschwindigkeit
	$VEL.ORI1	REAL	_/s	Schwenkgeschwindigkeit
	$VEL.ORI2	REAL	_/s	Drehgeschwindigkeit
Beschleunigungen	$ACC.CP	REAL	m/s@	Bahnbeschleunigung
	$ACC.ORI1	REAL	_/s@	Schwenkbeschleunigung
	$ACC.ORI2	REAL	_/s@	Drehbeschleunigung

Tab. 17 Systemvariablen für Geschwindigkeiten und Beschleunigungen Bahnbewegungen

Zumindest eine der Bewegungskomponenten Translation, Schwenken und Drehen fährt zu jedem Zeitpunkt der Bewegungsausführung mit programmierter Beschleunigung oder Geschwindigkeit. Die nichtdominierenden Komponenten werden zeitsynchron angepaßt.

Auch die Geschwindigkeiten und Beschleunigungen für die Bahnbewegungen werden bei Aufruf der Initialisierungssequenz des Basis–Pakets mit den in den Maschinendaten bzw. in $CONFIG.DAT definierten Maximalwerten vorbesetzt.

4.3.2 Orientierungsführung

Soll sich während der Bahnbewegung die Orientierung des Werkzeuges im Raum ändern, so kann die Art der Orientierungsführung mit Hilfe der Systemvariablen $ORI_TYPE eingestellt werden (s. Abb. 31):

G $ORI_TYPE = #CONST (ab Software R2.x #CONSTANT)	Während der Bahnbewegung bleibt die Orientierung konstant; für den Endpunkt wird die programmierte Orientierung ignoriert und die des Anfangspunktes benutzt. Dieser Wert wird auch bei der Initialisierung mittels BAS(#INITMOV,0) gesetzt.
G $ORI_TYPE = #VAR	Während der Bahnbewegung ändert sich die Orientierung kontinuierlich von der Anfangsorientierung zur Endorientierung um.

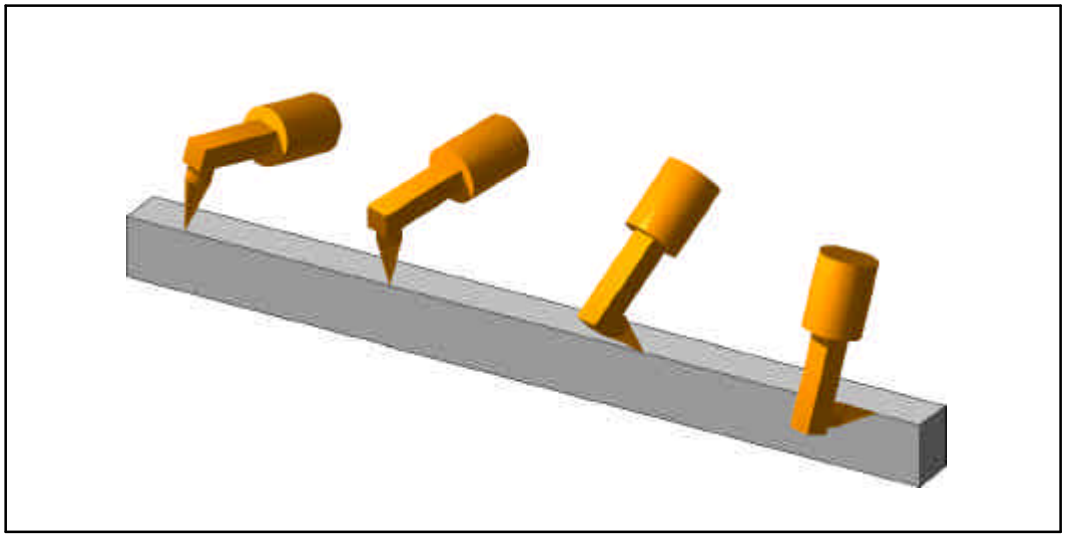


Abb. 31 Orientierungsänderung bei einer Linearbewegung

Bei Kreisbewegungen kann zusätzlich zu konstanter und variabler Orientierung noch zwischen raum- und bahnbezogener Orientierung gewählt werden:

G $CIRC_TYPE = #BASE	raumbezogene Orientierungsführung während der Kreisbewegung. Dieser Wert wird auch bei der Initialisierung mittels BAS(#INITMOV,0) gesetzt.
G $CIRC_TYPE = #PATH	bahnbezogene Orientierungsführung während der Kreisbewegung.

konstant + bahnbezog.

Bei der bahnbezogenen Orientierungsführung wird die Werkzeuglängsachse relativ zu Kreisebene und Kreistangente geführt. Dieser Zusammenhang läßt sich mit Hilfe des sogenannten bahnbegleitenden Dreibeines anschaulich erläutern (s. Abb. 32).

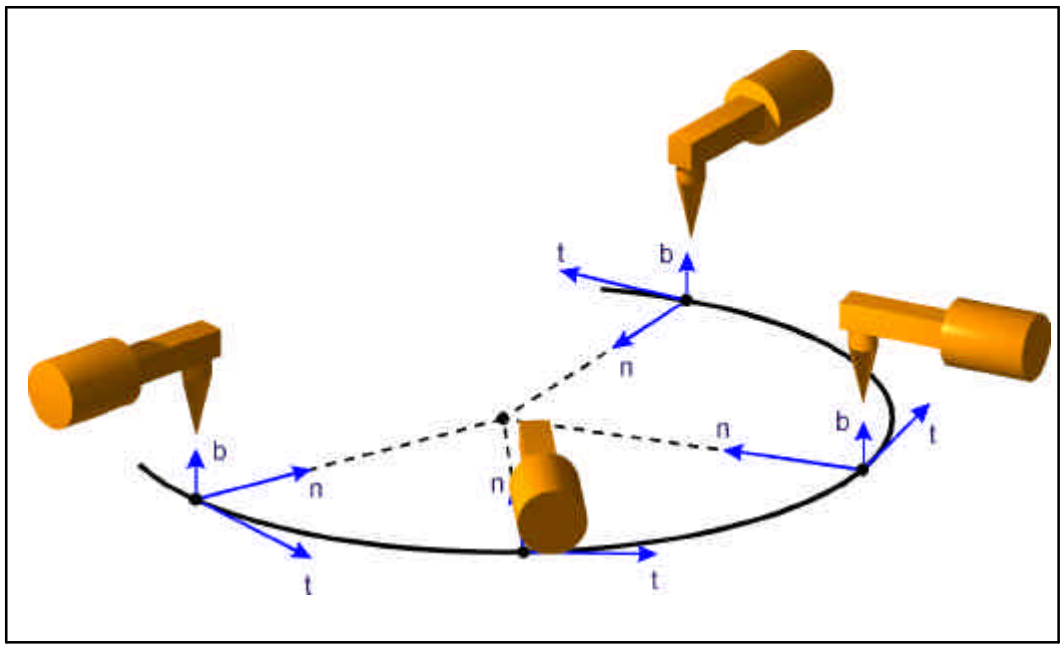


Abb. 32 Konstante bahnbezogene Orientierungsführung bei Kreisbewegungen

Das bahnbegleitende Dreibein setzt sich aus dem Kreistangentenvektor **t**, dem Normalenvektor **n** und dem Binormalenvektor **b** zusammen. Die Werkzeugorientierung wird in Abb. 32 mit dem bahnbegleitenden Dreibein auf dem Kreissegment nachgeführt. Bezogen auf das bahnbegleitende Dreibein ergibt sich für die Werkzeugpositionen keine Orientierungsänderung. Dies ist unter anderem eine wichtige Anforderung beim Lichtbogenschweißen.

variabel + bahnbezog.

Im gezeigten Beispiel wird die Werkzeugorientierung relativ zum bahnbegleitenden Dreibein während der Bewegung vom Start- zum Zielpunkt nicht verändert ($ORI_TYPE=#CONST). Wünscht man eine bahnbezogene Orientierungsänderung zwischen Start- und Zielposition ($ORI_TYPE=#VAR), so wird diese relativ zum bahnbegleitenden Dreibein durch überlagertes Drehen und Schwenken ausgeführt (s. Abb. 33). Die Orientierungsführung im bahnbegleitenden Dreibein ist bei Kreisbewegungen also vollkommen analog zur Orientierungsführung bei Linearbewegungen.

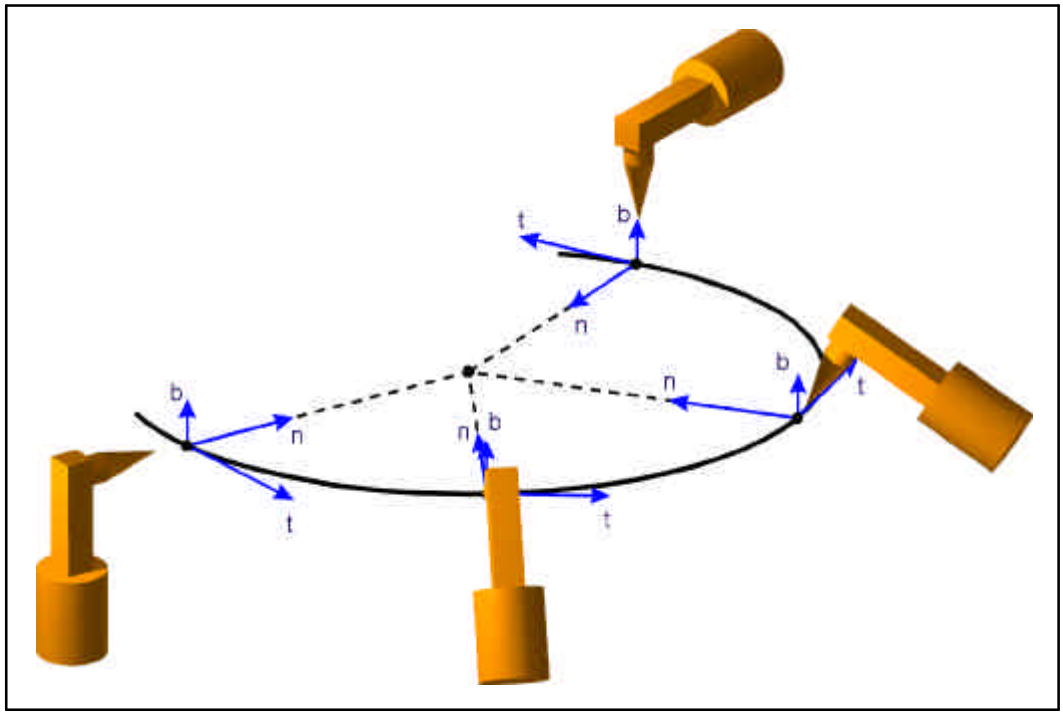


Abb. 33 Variable bahnbezogene Orientierungsführung bei Kreisbewegungen

konstant + raumbezog.

Bei der raumbezogenen Orientierungsführung wird die Orientierung relativ zum aktuellen Basissystem ($BASE) geführt.

Die raumbezogene Orientierungsführung ist vor allem für die Anwendungen sinnvoll, bei denen der Schwerpunkt bei der Bahnbewegung liegt, d.h. das Führen der Werkzeugspitze auf der Kreisbahn. Insbesondere bei Anwendungen mit geringer Orientierungsänderung zwischen Start- und Zielpunkt bzw. bei Anwendungen mit exakt raumkonstanter Orientierung (s. Abb. 34) während einer Kreisbewegung (z.B. Kleberauftrag mit rotationsymmetrischer Kleberdüse).

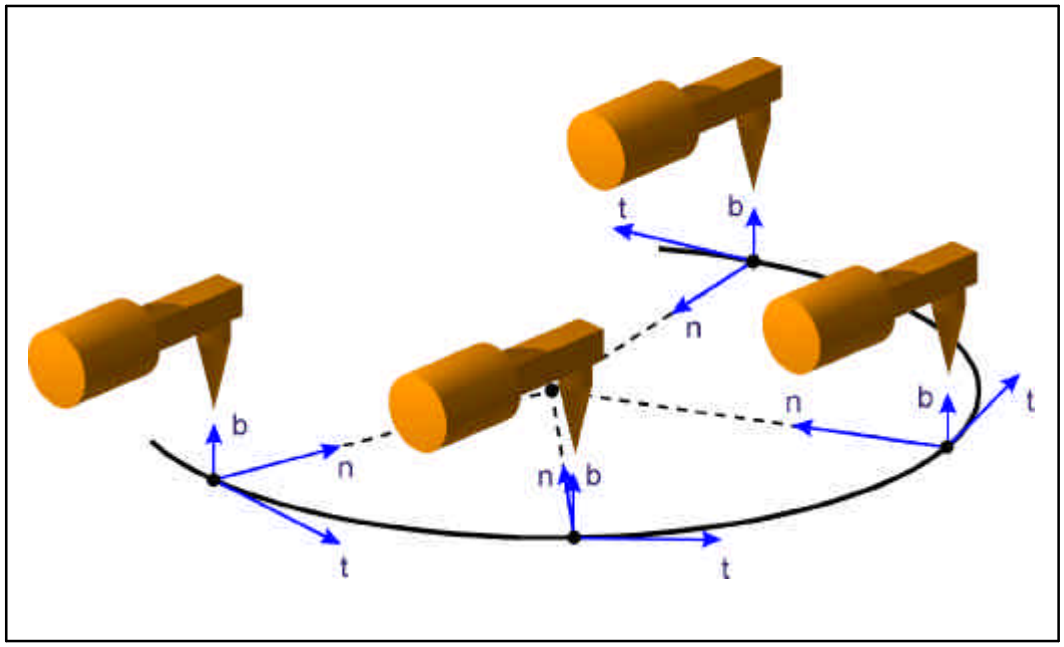


Abb. 34 Konstante raumbezogene Orientierungsführung bei Kreisbewegungen

variabel + raumbezog.

Eine raumbezogene Orientierungsänderung ($ORI_TYPE=#VAR) zwischen Start- und Zielposition wird wieder durch Überlagerung von Schwenk- und Drehbewegungen ausgeführt (s. Abb. 35). In diesem Fall allerdings relativ zum Basiskoordinatensystem.

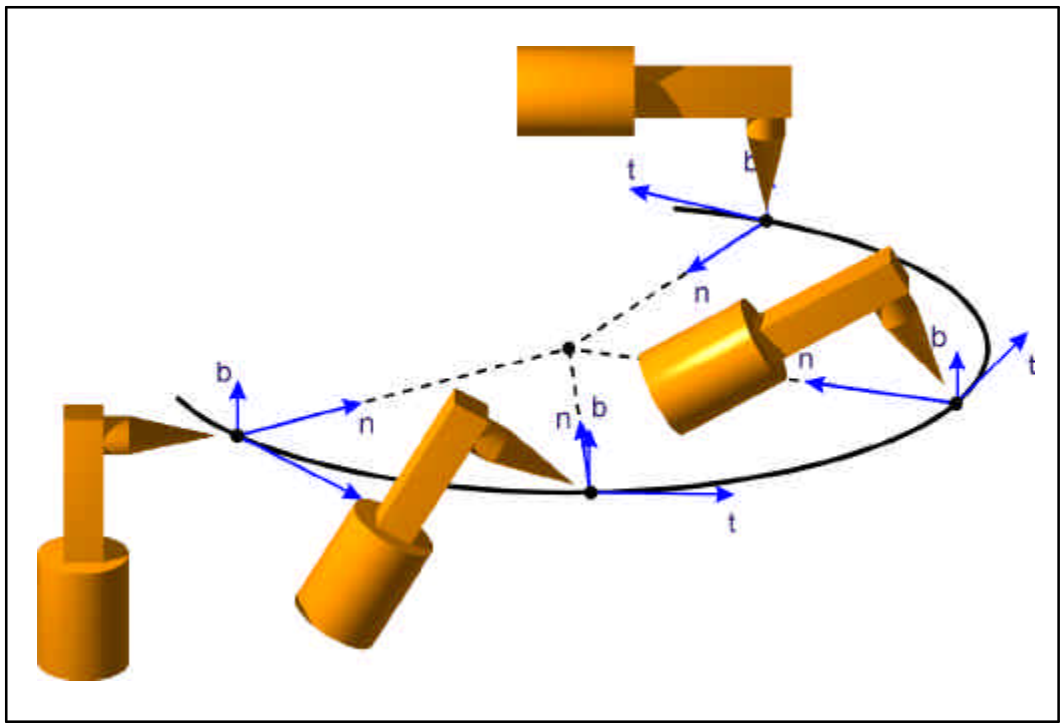


Abb. 35 Variable raumbezogene Orientierungsführung bei Kreisbewegungen

In Tab. 18 sind nochmals die Voreinstellungen der Systemvariablen zur Orientierungsführung bei Bahnbewegungen aufgelistet:

	im **System**	durch BAS(#INITMOV,0)
$ORI_TYPE	#VAR	
$CIRC_TYPE	#PATH	#BASE

Tab. 18 Voreinstellungen von $ORI_TYPE und $CIRC_TYPE

4.3.3 Linearbewegungen

LIN

Bei einer Linearbewegung berechnet die KR C1 eine Gerade von der aktuellen Position (im Programm ist dies der letzte programmierte Punkt) zu der Position, die im Bewegungsbefehl angegeben wurde.

Um die Werkzeugspitze entlang einer Geraden zu bewegen, werden Zwischenpunkte im Abstand von 1 Interpolationstakt (IPO-Takt, z. Zeit 12 ms) berechnet und diese abgefahren.

Die Programmierung einer Linearbewegung erfolgt durch die Schlüsselworte LIN oder LIN_REL in Verbindung mit der Zielpunktangabe, also analog zur PTP-Programmierung. Für lineare Bewegungen wird die Zielposition kartesisch angegeben. Es sind also nur die Datentypen FRAME oder POS zulässig.

Bei Linearbewegungen muß der Winkelstatus des Endpunktes gleich dem des Anfangspunktes sein. Eine Angabe von Status und Turn in einem Zielpunkt des Datentyps POS wird daher ignoriert. Deshalb muß vor der ersten LIN-Anweisung bereits eine PTP-Bewegung mit vollständiger Koordinatenangabe programmiert sein (z.B. HOME-Fahrt).

Das für Bahnbewegungen notwendige Besetzen der Geschwindigkeits- und Beschleunigungsvariablen sowie das Setzen von Tool- und Base-Koordinatensystem wird im folgenden Beispielprogramm wieder mittels der Initialisierungsroutine BAS.SRC vorgenommen.


```
DEF LIN_BEW ()

;--------- Deklarationsteil ---------
EXT BAS (BAS_COMMAND: IN, REAL: IN)
DECL AXIS HOME      ;Variable HOME vom Typ AXIS

;---------- Initialisierung ---------
BAS (#INITMOV, 0)   ;Initialisierung von Geschwindigkeiten,
                    ;Beschleunigungen, $BASE, $TOOL, etc.
HOME = {AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;------------- Hauptteil ------------
PTP HOME ; SAK-Fahrt
PTP {A5 30}

; Linearbewegung zur angegebenen Position, die Orientierung
; wird kontinuierlich auf die Zielorientierung veraendert
LIN  {X 1030,Y 350,Z 1300,A 160,B 45,C 130}

; Linearbewegung in der Y-Z-Ebene, S und T werden ignoriert
LIN  {POS: Y 0,Z 800,A 0,S 2,T 35}

; Linearbewegung zur angegebenen Position, die Orientierung
; wird dabei nicht veraendert
$ORI_TYPE=#CONST
LIN  {FRAME: X 700,Y -300,Z 1000,A 23,B 230,C -90}

; die Orientierung wird weiterhin nicht geaendert
LIN  {FRAME: Z 1200,A 90,B 0,C 0}

; Relativbewegung entlang der X-Achse
LIN_REL  {FRAME: X 300}

PTP HOME
END
```

4.3.4 Kreisbewegungen

CIRC

Zur eindeutigen Definition eines Kreises bzw. Kreisbogens im Raum benötigt man 3 Punkte, die voneinander verschieden sind und sich nicht auf einer Gerade befinden.

Der Anfangspunkt einer Kreisbewegung ist wie bei PTP oder LIN wieder durch die aktuelle Position gegeben.

Zum Programmieren einer Kreisbewegung mit den Anweisungen CIRC bzw. CIRC_REL muß deshalb neben dem Zielpunkt auch ein Hilfspunkt definiert werden. Bei der Berechnung der Bewegungsbahn durch die Steuerung werden vom Hilfspunkt nur die translatorischen Komponenten (X, Y, Z) ausgewertet. Die Orientierung der Werkzeugspitze ändert sich also je nach Orientierungsführung kontinuierlich vom Start- zum Zielpunkt oder bleibt konstant.

CA

Zusätzlich zu Hilfs- und Zielposition kann noch ein Kreiswinkel mit der Option CA (Circular Angle) programmiert werden. Die Geometrie des Kreisbogens wird dabei nach wie vor durch Start-, Hilfs- und Zielpunkt festgelegt. Die tatsächlich anzufahrende Zielposition auf dem Kreisbogen wird jedoch über den programmierten Kreiswinkel festgelegt. Dies ist vor allem zum Nachprogrammieren der Zielposition - ohne Änderung der Kreisgeometrie - hilfreich.

Der abzufahrende Kreisbogen kann entsprechend dem Kreiswinkel verlängert oder verkürzt werden. Die programmierte Zielorientierung wird dann im tatsächlichen Zielpunkt erreicht. Über das Vorzeichen des Kreiswinkels kann der Drehsinn, d.h. die Richtung in der der Kreisbogen abgefahren werden soll, festgelegt werden (s. Abb. 36):

CA > 0 _ im programmierten Sinn (Startpunkt ® Hilfspunkt ® Zielpunkt)

CA < 0 _ entgegen dem programmierten Sinn (Startpunkt ® Zielpunkt ® Hilfspunkt)

Der Wert des Kreiswinkels ist nicht begrenzt. Insbesondere können auch Vollkreise (> 360 _) programmiert werden.

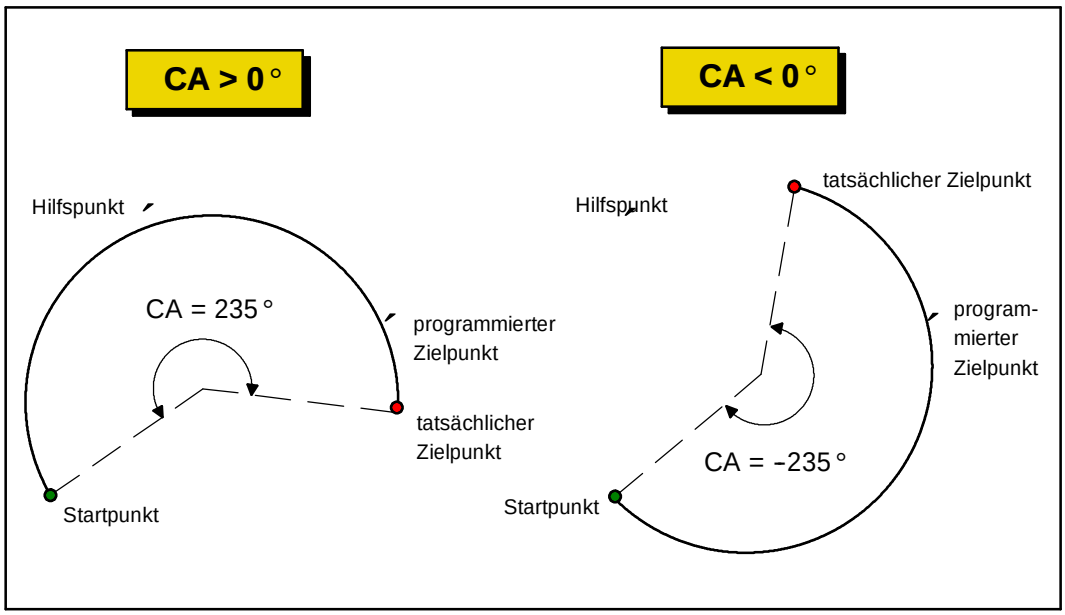


Abb. 36 Auswirkung der CA-Option im CIRC bzw. CIRC_REL-Befehl

Relativangaben für Hilfs- und Zielposition (CIRC_REL) beziehen sich jeweils auf die Startposition. Achsspezifische Positionsangaben sind wie bei LIN-Bewegungen nicht zulässig. Ebenso müssen $BASE und $TOOL vor Ausführung einer Kreisbewegung vollständig zugewiesen sein.

```
DEF  CIRC_BEW ( )

;--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL AXIS HOME

;---------- Initialisierung ---------
BAS (#INITMOV,0 )   ;Initialisierung von Geschwindigkeiten,
                    ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;------------- Hauptteil ------------
PTP  HOME           ;SAK-Fahrt
PTP  {POS: X 980,Y -238,Z 718,A 133,B 66,C 146,S 6,T 50}

; raumbezogene variable Orientierungsfuehrung (Voreinstellung)
CIRC {X 925,Y -285,Z 718},{X 867,Y -192,Z 718,A 155,B 75,C 160}

; raumbezogene konstante Orientierungsfuehrung
; Zielpunkt durch Winkelangabe festgelegt
$ORI_TYPE=#CONST
CIRC {X 982,Y -221,Z 718,A 50,B 60,C 0},{X 1061,Y -118,Z 718,
A -162,B 60,C 177}, CA 300.0

; bahnbezogene konstante Orientierungsfuehrung
; Zielpunkt durch Winkelangabe (rueckwaerts)
$CIRC_TYPE=#PATH
CIRC {X 867,Y -192,Z 718},{X 982,Y -221,Z 718,A 0}, CA -150

$ORI_TYPE=#VAR
LIN  {A 100}    ; Umorientierung des TCP

; bahnbezogene variable Orientierungsfuehrung
CIRC {X 963.08,Y -85.39,Z 718},{X 892.05,Y 67.25,Z 718.01,
A 97.34,B 57.07,C 151.11}

; relative Kreisbewegung
CIRC_REL {X -50,Y 50},{X 0,Y 100}

PTP  HOME
END
```

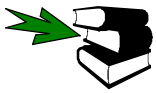
4.4 Rechnervorlauf

Ein ganz wesentliches Leistungsmerkmal eines Industrieroboters ist die Schnelligkeit, mit der er seine Arbeiten erledigen kann. Neben der Dynamik des Roboters ist hierfür auch die Effektivität der Bearbeitung des Anwenderprogramms, das neben den Bewegungen ja auch aus arithmetischen und die Peripherie steuernden Anweisungen besteht, von entscheidender Bedeutung.

Eine schnellere Programmbearbeitung kann

- durch die Verringerung der Bewegungsdauer und
- durch die Verkürzung der Stillstandszeit zwischen den Bewegungen erreicht werden.

Bei vorgegebenen Randbedingungen, wie maximalen Achsgeschwindigkeiten und –beschleunigungen, wird erstes durch das zeitoptimale Überschleifen von Bewegungen erreicht.



Abschnitt 4.5

Die Stillstandszeit zwischen den Bewegungen kann verkürzt werden, wenn man die aufwendigen Arithmetik– und Logik–Anweisungen zwischen den Bewegungssätzen während der Roboterbewegung abarbeitet, d.h. im Vorlauf bearbeitet (die Anweisungen "laufen" der Bewegung "vor").

$ADVANCE

Über die Systemvariable $ADVANCE kann festgelegt werden, wie viele Bewegungssätze der Vorlauf dem Hauptlauf (also dem aktuell bearbeiteten Bewegungssatz) maximal vorauseilen darf. Der während der Programmbearbeitung auf der Bedienoberfläche sichtbare Hauptlaufzeiger zeigt immer auf den Bewegungssatz, der gerade abgefahren wird.

Der Vorlaufzeiger ist dagegen nicht sichtbar und kann sowohl auf Anweisungen zeigen, die von der Steuerung komplett abgearbeitet werden, als auch auf Bewegungssätze, die von der Steuerung nur aufbereitet und zeitlich später im Hauptlauf ausgeführt werden (s. Abb. 37).

```
13
14   $ADVANCE = 1
15
16   LIN  {X 1620,Y 0,Z 1910,A 0,B 90,C 0}
17
18   STROM=(STROM*1.2)/0.5
19   FOR I=1 TO  6
20       $VEL_AXIS[I]=60
21       $ACC_AXIS[I]=35
22   ENDFOR
23
24   PTP  PUNKT6
25
26   SPANNUNG=110
27
28   PTP PUNKT7
29
```

Vorlaufzeiger

Abb. 37 Hauptlauf– und Vorlaufzeiger

Im obigen Programmausschnitt ist der Vorlauf auf 1 gesetzt und der Hauptlaufzeiger befindet sich in Zeile 16 (d.h. die LIN-Bewegung wird gerade ausgeführt). Ein Rechnervorlauf von 1 bedeutet nun, daß die Anweisungen von Zeile 16 bis 22 sowie Zeile 26 parallel zur Bewegungsausführung bereits komplett abgearbeitet wurden, und die Bewegungsdaten für die PTP-Bewegungen in Zeile 24 fertig aufbereitet sind und für Zeile 28 gerade aufbereitet werden. Das heißt aber auch, daß selbst bei einem Vorlauf von 0 immer noch die nächste Bewegungssequenz im Vorlauf aufbereitet werden kann.

Ob der in $ADVANCE vorgegebene Vorlauf allerdings auch tatsächlich erreicht wird, hängt von der zeitlichen Belastung des Prozessors in der Steuerung ab. Die Belastung ist zum Beispiel besonders groß, wenn viele kurze Bewegungssätze programmiert worden sind.

automat. Vorlaufstop

Aus Sicherheitsgründen lösen Anweisungen und Daten, die die Peripherie beeinflussen (z.B. Ein-/Ausgabeanweisungen), einen Vorlaufstop aus (s. Tab. 19).

Anweisungen	ANOUT ON	ANOUT OFF		
	ANIN ON	ANIN OFF		
	DIGIN ON	DIGIN OFF		
	PULSE			
	HALT	WAIT		
	CREAD	CWRITE	COPEN	CCLOSE
Systemvariablen	$ANOUT[*Nr*]	$ANIN[*Nr*]		
	$DIGIN1	$DIGIN2	¼	$DIGIN6
	$OUT[*Nr*]	$IN[*Nr*]		
	$AXIS_ACT	$AXIS_BACK	$AXIS_FOR	$AXIS_RET
	$POS_ACT	$POS_BACK	$POS_FOR	$POS_RET
	$OV_PRO			
importierte Variablen	alle, bei Zugriff			

Tab. 19 Anweisungen und Variablen die einen automatischen Vorlaufstop auslösen

CONTINUE

In Anwendungsfällen, bei denen dieser Vorlaufstop verhindert werden soll, muß direkt vor der betreffenden Anweisung der Befehl CONTINUE programmiert werden. Die Steuerung setzt dann den Vorlauf fort. Die Wirkung ist aber nur auf die nächste Programmzeile beschränkt (auch Leerzeile!!).

Wollen Sie dagegen an einer bestimmten Stelle den Vorlauf stoppen, ohne dazu die Systemvariable $ADVANCE ändern zu müssen, so können Sie sich mit einem kleinen Trick behelfen: Programmieren Sie an dieser Stelle einfach eine Wartezeit von 0 Sekunden. Die Anweisung WAIT löst dann einen automatischen Vorlaufstop aus, tut aber sonst nichts:

```
WAIT SEC 0
```

Abschnitt 6.3.2

Um ein Überschleifen zu ermöglichen, muß mindestens ein Rechnervorlauf von 1 eingestellt sein.
In einem Interrupt-Unterprogramm ist kein Rechnervorlauf möglich. Die Steuerung arbeitet Interruptprogramme immer zeilenweise ab, daher ist ein Überschleifen in Interruptprogrammen nicht möglich.

$ADVANCE kann Werte von 0 bis 5 annehmen. Die Voreinstellungen sind in Tab. 20 aufgelistet:

	im **System**	durch **BAS (#INITMOV, 0)**
$ADVANCE	0	3

Tab. 20 Voreinstellungen von $ADVANCE

NOTIZEN:

4.5 Überschleifbewegungen

Um Kollisionen des Roboters mit Objekten innerhalb seines Arbeitsraumes zu vermeiden, ist es häufig notwendig, einen Bewegungsvorgang durch mehrere Einzelsätze vorzugeben um so die Hindernisse zu umfahren.

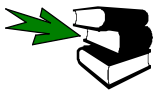
Da es nicht notwendig ist, die Zwischenpunkte in einem zeitraubenden Positioniervorgang exakt anzufahren, kann man in einem definierbaren Abstand zum Zielpunkt des Einzelsatzes damit beginnen, in den nächsten Bewegungssatz überzuschleifen. Damit erhält man möglichst schnell ausgeführte Bewegungsfolgen mit abgerundeten Bahnen (s. Abb. 38).

Überschleifkontur

Die Überschleifkontur wird dazu automatisch von der Steuerung generiert. Der Programmierer hat nur einen Einfluß auf den Beginn und das Ende des Überschleifens. Zur Berechnung des Überschleifsatzes benötigt die Steuerung die Daten des Anfangspunktes, des Überschleifpunktes und des Zielpunktes.

Um ein Überschleifen zu ermöglichen, muß daher der Rechnervorlauf ($ADVANCE) mindestens auf 1 gesetzt sein. Falls der Rechnervorlauf zu klein ist, erscheint die Meldung "Überschleifen nicht möglich" und die Punkte werden genau angefahren.

Wenn Sie nach einer Überschleifanweisung eine Anweisung programmieren, die einen automatischen Vorlaufstop auslöst (s. Tab. 19), ist ein Überschleifen nicht möglich.



TRIGGER-Anweisung als Abhilfe, Abschnitt 10

Überschleifen

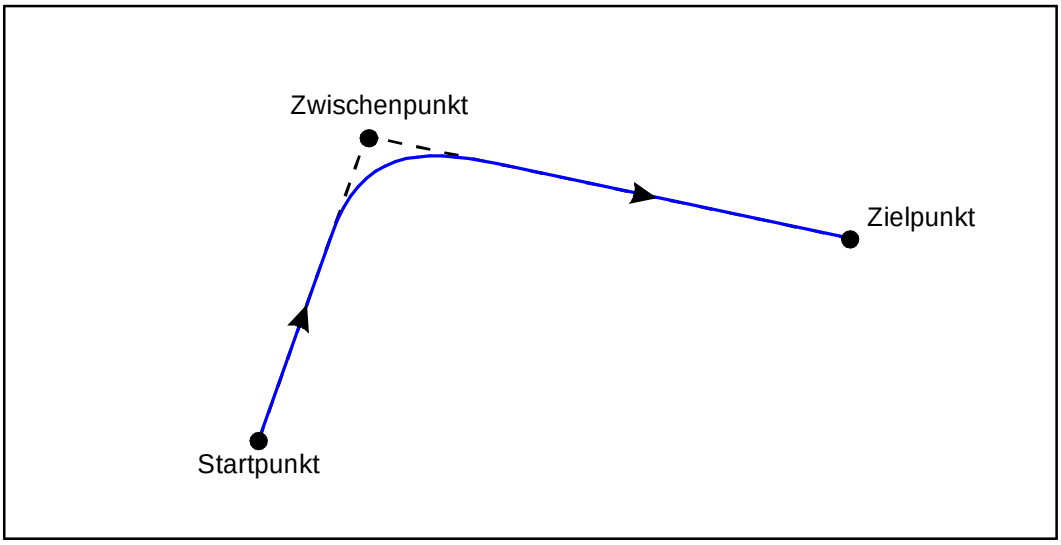


Abb. 38 Überschleifen von Zwischenpunkten

4.5.1 PTP-PTP-Überschleifen

Zum PTP-Überschleifen berechnet die Steuerung die axial zu verfahrenden Distanzen im Überschleifbereich und plant für jede Achse Geschwindigkeitsprofile, die tangentiale Übergänge von den Einzelsätzen zur Überschleifkontur sicherstellen.

Überschleifbeginn

Das Überschleifen wird begonnen, wenn die letzte (= führende) Achse einen bestimmten Winkel zum Überschleifpunkt unterschreitet. In den Maschinendaten ist für jede Achse ein Winkel vordefiniert:

```
$APO_DIS_PTP[1] = 90
                  ?
$APO_DIS_PTP[6] = 90
```

Im Programm läßt sich durch $APO.CPTP der Beginn des Überschleifens als Prozentsatz von diesen Maximalwerten angeben. Zum Beispiel:

```
$APO.CPTP = 50
```

In diesem Fall wird also das Überschleifen begonnen, wenn die erste Achse einen Restwinkel von 45 _(50% von 90 _) zum Überschleifpunkt zurückzulegen hat. Das Überschleifende erfolgt genau dann, wenn die erste Achse einen Winkel von 45 _vom Überschleifpunkt verfahren ist.

Þ Je größer $APO.CPTP, desto mehr wird die Bahn abgerundet.

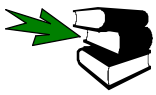
Grundsätzlich kann das Überschleifen nicht über die Satzmitte erfolgen! In diesem Fall wird das System selbständig auf die Satzmitte begrenzen.

C_PTP

Das Überschleifen eines Punktes wird im PTP-Befehl durch Anhängen des Schlüsselwortes C_PTP angezeigt:

```
PTP PUNKT4 C_PTP
```

Auch der PTP-Überschleifsatz wird zeitoptimal ausgeführt, d.h. während überschliffen wird, fährt stets mindestens eine Achse mit dem programmierten Grenzwert für Beschleunigung oder Geschwindigkeit. Gleichzeitig wird für jede Achse sichergestellt, daß die zulässigen Getriebe- und Motorenmomente nicht überschritten werden. Das standardmäßig eingestellte höhere Fahrprofil gewährleistet weiterhin eine geschwindigkeits- und beschleunigungsinvariante Bewegungsausführung.



Abschnitt 4.2.1

Aus folgendem Beispiel können Sie die Wirkung der Überschleifanweisung und der Variablen $APO.CPTP ersehen. Die abgefahrene Bahn ist in Abb. 39 in der x-y-Ebene dargestellt. Insbesondere ist darin auch ersichtlich, daß bei PTP-Bewegungen der TCP nicht auf einer Geraden zwischen den Zielpunkten verfahren wird.

```
DEF  UEBERPTP ( )

;--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL AXIS HOME

;---------- Initialisierung ---------
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;------------- Hauptteil ------------
PTP  HOME ; SAK-Fahrt

PTP {POS:X 1159.08,Y -232.06,Z 716.38,A 171.85,B 67.32,C
162.65,S 2,T 10}

;Ueberschleifen des Punktes
PTP {POS:X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C 111.66,S 2,T
10} C_PTP

PTP {POS:X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95,S 2,T
10}

;Ueberschleifen von zwei Punkten
$APO.CPTP=20
PTP {POS:X 1296.61,Y 133.41,Z 715,A 150.32,B 55.07,C 130.23,S
2,T 11} C_PTP
PTP {POS:X 988.45,Y 238.53,Z 715,A 114.65,B 50.46,C 84.62,S 2,T
11} C_PTP

PTP {POS:X 1209.5,Y 381.09,Z 715,A -141.91,B 82.41,C -159.41,S
2,T 11}

PTP  HOME
END
```

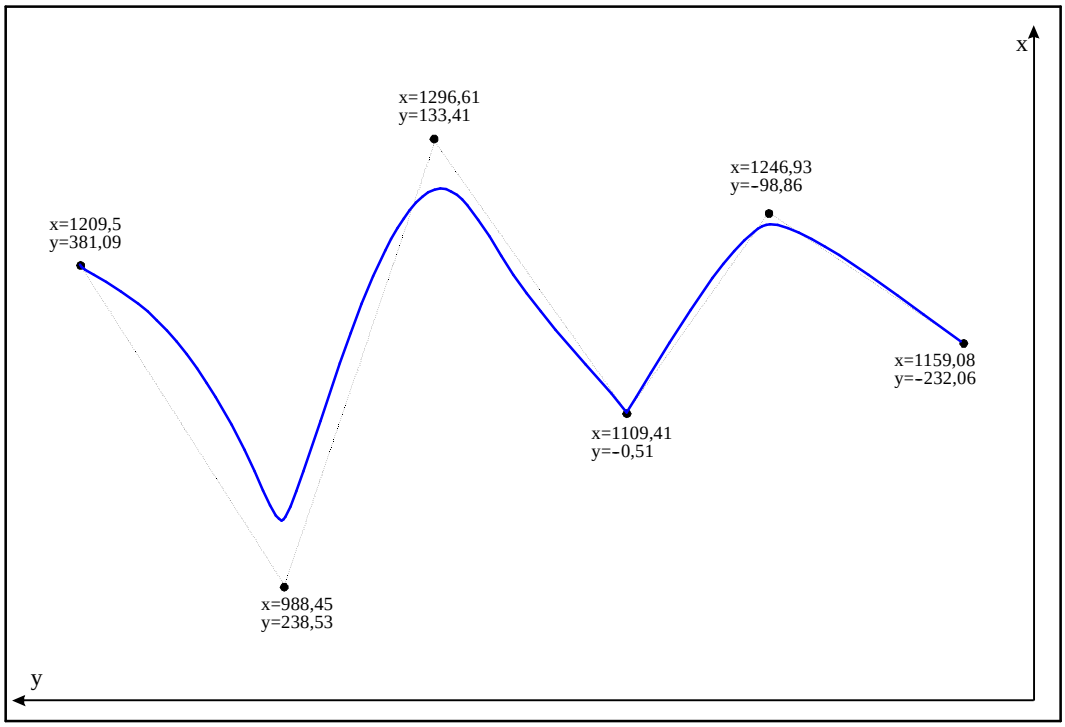


Abb. 39 Beispiel für PTP-PTP-Überschleifen

Da die Bahn einer PTP-Bewegung im allgemeinen weder eine Gerade ist, noch in einer Raumebene liegt, kann man sie streng genommen eigentlich nicht wie in Abb. 39 darstellen. Trotz der Tatsache, daß der z-Wert aller Punkte des Beispiels gleich ist, liegt nicht jeder Punkt der Bewegungsbahn in der Ebene z=715 mm. Die abgebildete Bahn ist also lediglich eine Projektion der wirklichen Bahn in die x-y-Ebene.

4.5.2 LIN-LIN-Überschleifen

Zur kontinuierlichen Bewegung entlang komplexer Bahnen besteht die Forderung, auch zwischen linearen Einzelsätzen zu überschleifen. Die unterschiedlichen Orientierungsbewegungen in den LIN-Sätzen müssen dabei ebenso stufenlos ineinander überführt werden.

Als Überschleifkontur berechnet die Steuerung eine parabelförmige Bahn, da diese Konturform den Bahnverlauf der Einzelsätze bei bester Nutzung der Beschleunigungsreserven im Überschleifbereich sehr gut approximiert.

Überschleifbeginn

Zur Festlegung des Überschleifbeginns stehen drei vordefinierte Variablen bereit (s. Tab. 21):

Variable	Datentyp	Einheit	Bedeutung	Schlüsselwort im Befehl
$APO.CDIS	REAL	mm	Translatorisches Distanzkriterium	C_DIS
$APO.CORI	REAL	_	Orientierungsdistanz	C_ORI
$APO.CVEL	INT	%	Geschwindigkeitskriterium	C_VEL

Tab. 21 Systemvariablen zur Festlegung des Überschleifbeginns bei Bahnbewegungen

Distanzkriterium

Der Variablen $APO.CDIS kann eine translatorische Distanz zugewiesen werden. Wird das Überschleifen auf diese Variable getriggert, verläßt die Steuerung die Einzelsatzkontur genau dann, wenn die Entfernung zum Zielpunkt den Wert in $APO.CDIS unterschreitet.

Orientierungskriterium

Der Variablen $APO.CORI kann eine Orientierungsdistanz zugewiesen werden. Die Einzelsatzkontur wird in diesem Fall genau dann verlassen, wenn der dominierende Orientierungswinkel (Schwenken oder Drehen der Werkzeuglängsachse) die in $APO.CORI festgelegte Distanz zum programmierten Überschleifpunkt unterschreitet.

Geschwindigkeitskriterium

Der Variablen $APO.CVEL kann ein Prozentwert zugewiesen werden. Dieser Wert gibt an, bei wieviel Prozent der programmierten Geschwindigkeit ($VEL.CP) der Überschleifvorgang in der Abbremsphase des Einzelsatzes begonnen wird. Dabei wird jene Komponente aus Translation, Schwenken und Drehen ausgewertet, die bei der Bewegung den programmierten Geschwindigkeitswert erreicht, bzw. ihm am nächsten kommt.

Þ Je größer die Werte in $APO.CDIS, $APO.CORI oder $APO.CVEL sind, desto früher wird mit dem Überschleifen begonnen.

C_DIS
C_ORI
C_VEL

Das Überschleifen wird durch Hinzufügen eines der Schlüsselworte C_DIS, C_ORI oder C_VEL zur LIN- oder LIN_REL-Anweisung aktiviert.

Zur Verdeutlichung dient folgendes Beispiel in Zusammenhang mit Abb. 40:

```
DEF  UEBERLIN ( )

;--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL AXIS HOME

;---------- Initialisierung ---------
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;------------- Hauptteil ------------
PTP  HOME ; SAK-Fahrt

PTP  {POS: X 1159.08,Y -232.06,Z 716.38,A 171.85,B 67.32,C
162.65,S 2,T 10}

;Ueberschleifen des Punktes nach Distanzkriterium
$APO.CDIS=20
LIN  {X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C 111.66} C_DIS
LIN  {X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95}

;ueberschleifen von zwei punkten
LIN  {X 1296.61,Y 133.41,Z 714.99,A 150.32,B 55.07,C 130.23}
C_ORI
LIN  {X 988.45,Y 238.53,Z 714.99,A 114.65,B 50.46,C 84.62} C_VEL

LIN  {X 1209.5,Y 381.09,Z 715,A -141.91,B 82.41,C -159.41}

PTP  HOME
END
```

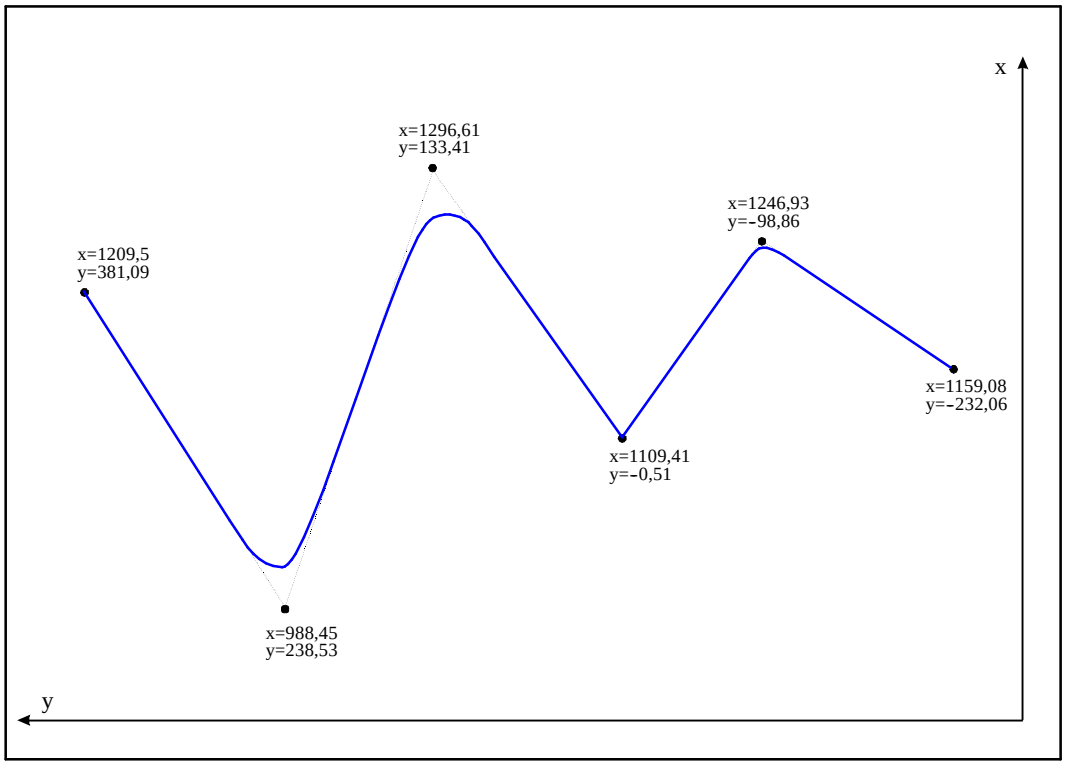


Abb. 40 Beispiel für LIN-LIN-Überschleifen

Die Position, bei der die Überschleifkontur in den nachfolgenden LIN-Satz mündet, wird von der Steuerung automatisch berechnet. Falls $ACC und $VEL für beide Einzelsätze identisch sind und die Satzlängen ausreichen, ist die Überschleifparabel symmetrisch zur Winkelhalbierenden zwischen den beiden Einzelsätzen. Bei kurzen Einzelsätzen wird der Überschleifbeginn auf die halbe Satzlänge begrenzt, um Kollisionen mit einem eventuell vorangehenden Überschleifen auszuschließen. Die Geschwindigkeit wird dabei derart reduziert, daß ein gegebenenfalls nachfolgender Genauhalt immer eingehalten werden kann.

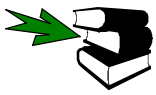
Die Übergänge zwischen Einzelsätzen und Überschleifkontur sind stetig und verlaufen tangential. Das gewährleistet einen "weichen", mechanikschonenden Übergang, da die Geschwindigkeitskomponenten stets stetig sind.

Die von der Steuerung generierte Kontur im Überschleifbereich ist unabhängig von Override-Änderungen, die zu jedem beliebigen Zeitpunkt der Bewegung zulässig sind.

Eine Programmunterbrechung, wie NOT-AUS, Stop oder Interrupt, kann bei `$ADVANCE=1` **nach erfolgtem Start zu Verlust des Überschleifkriteriums führen. Der Roboter fährt dann den nächsten Raumpunkt als Genauhalt an. Durch Vergrößerung des Vorlaufs kann diesem Verhalten entgegen gewirkt werden (in der Initialisierungssequenz INI wird** `$ADVANCE=3` **gesetzt).**

4.5.3 CIRC-CIRC-Überschleifen und CIRC-LIN-Überschleifen

Die Aufgabenstellung beim Überschleifen zwischen CIRC-Sätzen und anderen CP-Sätzen (LIN oder CIRC) ist nahezu identisch zum Überschleifen zwischen zwei LIN-Sätzen. Sowohl die Orientierungsbewegung als auch die translatorische Bewegung sollen ohne Geschwindigkeitssprünge von der einen in die andere Einzelsatzkontur überführt werden. Der Überschleifbeginn wird wieder durch die Variablen $APO.CDIS, $APO.CORI oder $APO.CVEL definiert, deren Auswertung vollständig identisch zu LIN-Sätzen ausgeführt wird. Das Einstellen des gewünschten Überschleifkriterium erfolgt ebenfalls mit Hilfe der Schlüsselworte C_DIS, C_ORI oder C_VEL (s. Tab. 21).



Abschnitt 4.5.2

Das CIRC-CIRC-Überschleifen soll wieder anhand eines Beispiels und der abgebildeten Bewegungsbahn (s. Abb. 41) verdeutlicht werden:

```
DEF  UEBERCIR ( );--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL AXIS HOME;---------- Initialisierung ---------
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0};-------------
Hauptteil ------------
PTP  HOME ; SAK-Fahrt
PTP  {POS: X 980,Y -238,Z 718,A 133,B 66,C 146,S 6,T 50}
; raumbezogene variable Orientierungsfuehrung
; Ueberschleifen nach dem Distanzkriterium
$APO.CDIS=20
CIRC {X 925,Y -285,Z 718},{X 867,Y -192,Z 718,A 155,B 75,C 160}
C_DIS

; raumbezogene konstante Orientierungsfuehrung
; Zielpunkt durch Winkelangabe festgelegt
; kein Ueberschleifen moeglich wegen Vorlaufstop durch $OUT
$ORI_TYPE=#CONST
CIRC {X 982,Y -221,Z 718,A 50,B 60,C 0},{X 1061,Y -118,Z 718,A
-162,B 60,C 177}, CA 150 C_ORI
$OUT[3]=TRUE

; bahnbezogene variable Orientierungsfuehrung
; Ueberschleifen nach dem Orientierungskriterium
$ORI_TYPE=#VAR
$CIRC_TYPE=#PATH
CIRC {X 963.08,Y -85.39,Z 718},{X 892.05,Y 67.25,Z 718.01,A
97.34,B 57.07,C 151.11} C_ORI

; relative Kreisbewegungen

; Ueberschleifen nach dem Geschwindigkeitskriterium
$APO.CVEL=50
CIRC_REL {X -50,Y 50},{X 0,Y 100} C_VEL

; Ueberschleifen nach dem Distanzkriterium
$APO.CDIS=40
CIRC_REL {X -50,Y 50},{X 0,Y 100} C_DIS

CIRC_REL {X -50,Y 50},{X 0,Y 100}

PTP  HOME
END
```

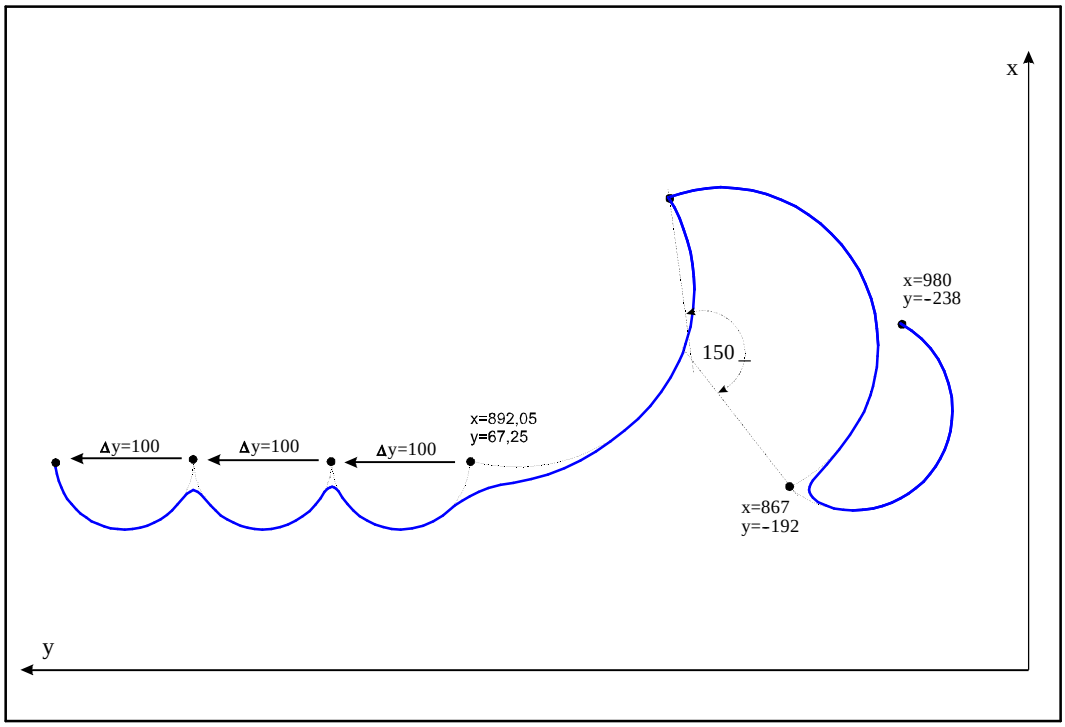


Abb. 41 Beispiel für CIRC-CIRC-Überschleifen

Wegen der Forderung nach tangentialen Übergängen kann beim Überschleifen mit CIRC-Sätzen im allgemeinen keine symmetrische Überschleifkontur berechnet werden. Die Überschleifbahn besteht daher aus zwei Parabelsegmenten, die tangential ineinander übergehen und die gleichzeitig tangential in die Einzelsätze einmünden (s. Abb. 42).

LIN-CIRC Überschleif

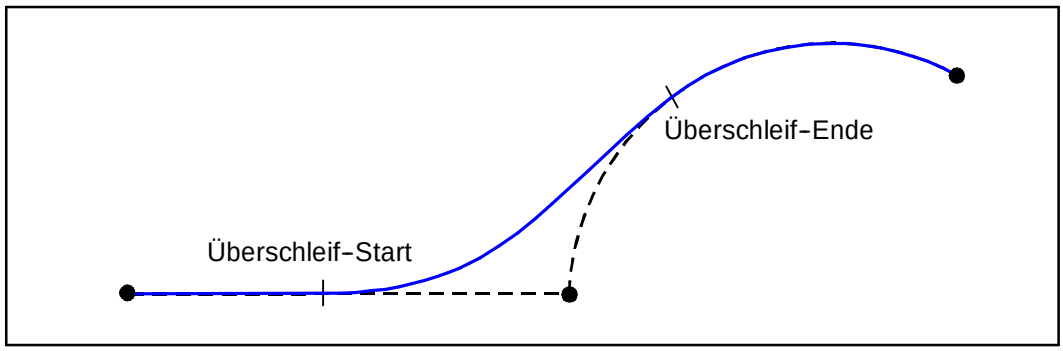


Abb. 42 Überschleifkontur bei LIN-CIRC-Sätzen

Beim CIRC-Überschleifen wird grundsätzlich raumbezogen interpoliert. Startorientierung ist immer die Orientierung, welche im Überschleifpunkt erreicht würde. Werden zwei Überschleifsätze mit bahnbezogener Orientierung abgefahren, so verhält sich die Umorientierung im Überschleifbereich trotzdem raumbezogen.

4.5.4 PTP-Bahnüberschleifen

Die Möglichkeit zwischen achsspezifischen PTP- und kartesischen Bahn-Bewegungssätzen überzuschleifen, ist vor allem für Anwendungen in den Bereichen Handhabung und Montage von Interesse. Die eigentlichen Arbeitsvorgänge (Fügen, Palettieren, Aufspannen von Werkzeugen oder Werkstücken, etc.) werden in vergleichsweise kurzen Zyklen ausgeführt, die sich aus kontrollierten Bahnbewegungen zusammensetzen. Dazwischen fallen Transportaufgaben an, bei denen vor allem Schnelligkeit gefragt ist, während die Einhaltung einer bestimmten Bahn von untergeordnetem Interesse ist. Man wird deshalb für diese Zwecke bevorzugt auf die Achsinterpolation zurückgreifen:

- Sie ist grundsätzlich schneller als ihr kartesisches Gegenstück, insbesondere in der Nähe singulärer Stellungen.
- Sie ermöglicht im Gegensatz zur kartesischen Interpolation einen Konfigurationswechsel, z.B. einen Übergang vom Grund- in den Überkopfbereich oder ein Durchschwenken durch die gestreckte Handstellung.
- Konturinvarianz gegen Override-Änderung ist gewährleistet.

Die Vorteile der achsspezifischen Interpolation können aber erst dann voll ausgeschöpft werden, wenn ein kontinuierlicher Übergang zwischen achsspezifischen und kartesischen Sätzen möglich ist, denn bei einem Genauhalt geht die anderweitig gewonnene Zeit zum großen Teil wieder verloren.

Die Programmierung des PTP-Bahnüberschleifens erfolgt vollkommen analog zu den bisher vorgestellten Verfahren. Der Überschleifbereich wird wie folgt festgelegt:

PTP ® Bahnüberschleifen

Überschleif PTP® Bahn

Der Beginn des Überschleifens wird durch das PTP-Kriterium `$APO.CPTP` in gewohnter Weise (s. 4.5.1) ermittelt. Für den nachfolgenden Bahnbewegungssatz kann explizit ein Überschleifkriterium (`C_DIS, C_ORI, C_VEL`), das den Eintritt in den CP-Satz festlegt, angegeben werden.

Dies geschieht durch die Anweisungsfolge:

```
PTP PUNKT1 C_PTP C_DIS
LIN PUNKT2
```

Fehlt im PTP-Satz eine Angabe für das im CP-Satz gewünschte Überschleifkriterium, so wird `C_DIS` als Default-Wert für die Ermittlung des Eintritts in den CP-Satz herangezogen.

Bahn ® PTP-Überschleifen

Überschleif Bahn® PTP

Für den CP-Satz zählt das programmierte Überschleifkriterium, für den PTP-Satz wird auf $APO.CPTP zurückgegriffen.

Eine Anweisungsfolge könnte also so aussehen:

```
CIRC HILF PUNKT1 C_VEL
PTP PUNKT2
```

Das CP-PTP-Überschleifen kann allerdings nur garantiert werden, wenn keine der Roboterachsen im Bahnsatz mehr als 180 _dreht und der Status S nicht wechselt, da diese Stellungsänderungen bei der Planung der Überschleifkontur nicht vorhersehbar sind. Tritt ein solcher Konfigurationswechsel im Bahnsatz vor dem Überschleifen auf (Änderung von S oder T), wird der Bahnsatz wie ein Einzelsatz zum programmierten Zielpunkt zu Ende gefahren und die quittierbare Fehlermeldung "Überschleifen CP/PTP nicht durchführbar" ausgegeben. Der Anwender sollte dann den CP-Satz in mehrere Einzelsätze zerlegen, so daß der Einzelsatz vor dem CP-PTP-Überschleifen hinreichend kurz ist, um einen Wechsel von S oder T in allen Roboterachsen ausschließen zu können.

Abschnitt 4.2.2

Im folgenden Beispiel wurde ein PTP-LIN-Überschleif, ein LIN-CIRC-Überschleif und ein CIRC-PTP-Überschleif programmiert (s. Abb. 43):


```
DEF  UEBERB_P ( )

;--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL AXIS HOME

;---------- Initialisierung ---------
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

;------------- Hauptteil ------------
PTP  HOME ; SAK-Fahrt
PTP  {POS: X 1281.55,Y -250.02,Z 716,A 79.11,B 68.13,C 79.73,S
6,T 50}

PTP  {POS: X 1209.74,Y -153.44,Z 716,A 79.11,B 68.13,C 79.73,S
6,T 50} C_PTP C_ORI
LIN  {X 1037.81,Y -117.83,Z 716,A 79.11,B 68.13,C 79.73}

$APO.CDIS=25
LIN  {X 1183.15,Y -52.64,Z 716,A 79.11,B 68.13,C 79.73} C_DIS
CIRC {POS: X 1134,Y 53.63,Z 716},{X 1019.21,Y 124.02,Z 716,A
79.11,B 68.12,C 79.73}

CIRC {POS: X 1087.47,Y 218.67,Z 716},{X 1108.78,Y 267.16,Z 716,A
79.11,B 68.12,C 79.73} C_ORI
PTP  {POS: X 1019.31,Y 306.71,Z 716,A 80.8,B 68,C 81.74,S 6,T
59}

PTP  HOME
END
```

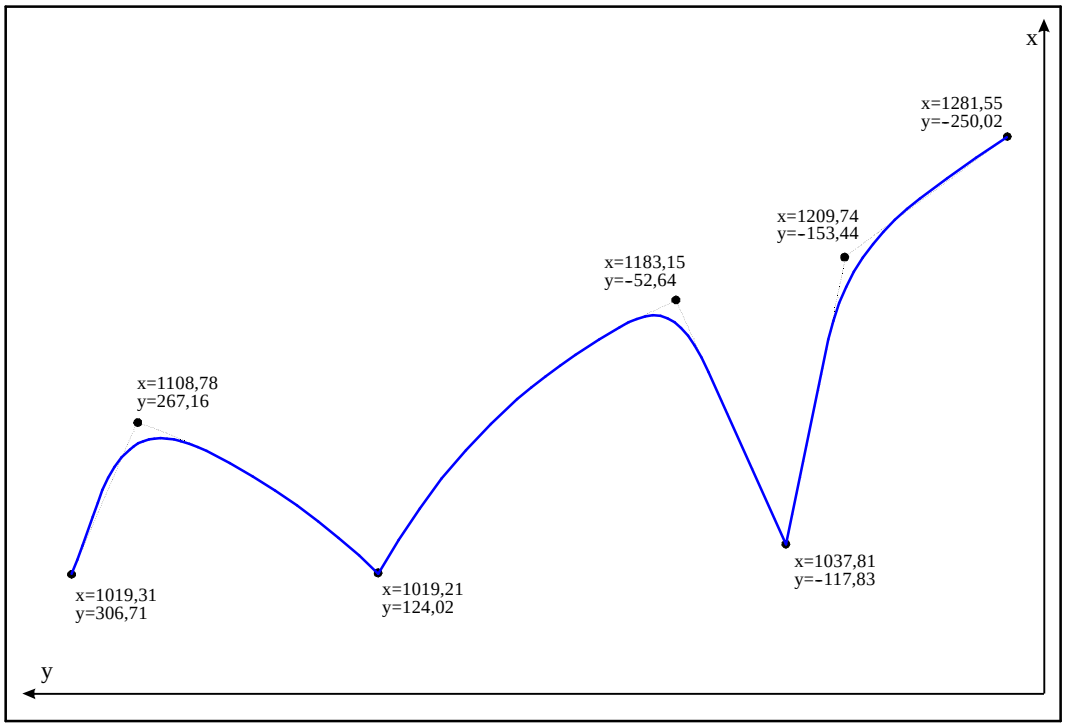


Abb. 43 PTP-Bahnüberschleifen und Bahn-Bahnüberschleifen

NOTIZEN:

4.6 Teachen von Punkten

Die Integration des Teach-In Verfahrens ist ein wesentliches Qualitätsmerkmal einer Roboterprogrammiersprache.

!-Zeichen

In KRL programmieren Sie dazu einfach ein **!**-Zeichen als Platzhalter für die später zu teachenden Koordinaten:

```
PTP !
LIN ! C_DIS
CIRC ! ,CA 135.0
```

An der Steuerung können dann die betreffenden Koordinaten nach Drücken des "Ändern"-Softkeys über den "Touch Up"-Softkey in das Programm übernommen werden. Die aktuellen Koordinaten werden direkt in der gewählten Struktur in den SRC-Code geschrieben, z.B.:

```
PTP {POS:X 145.25,Y 42.46,Z 200.5,A -35.56,B 0.0,C 176.87,S 2,T 2}
LIN {X -23.55,Y 0.0,Z 713.56,A 0.0,B 34.55,C -90.0} C_DIS
CIRC {X -56.5,Y -34.5,Z 45.56,A 35.3,B 0.0,C 90.0},{X 3.5,Y 20.30,
     Z 45.56,A 0.0,B 0.0,C 0.0}, CA 135.0
```

Beim Teachen von kartesischen Koordinaten wird immer das derzeit im System gültige Basiskoordinatensystem ($BASE) als Bezugssystem zugrunde gelegt. Versichern Sie sich daher, daß beim Teachen immer das für die spätere Bewegung verwendete Basiskoordinatensystem eingestellt ist.

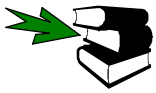
Die KR C1 erlaubt noch eine weitere Variante des Teachens: Programmieren Sie eine Bewegungsanweisung mit einer Variablen, die Sie NICHT im Vereinbarungsteil deklarieren, z.B.:

```
PTP STARTPUNKT
```

Nach Drücken der Softkeys "Ändern" und "Var" werden Sie nun aufgefordert die gewünschte Struktur auszuwählen. Ist dies geschehen, wird in der zugehörigen **Datenliste** automatisch eine Variable STARTPUNKT deklariert und mit den aktuellen Ist-Koordinaten bezüglich des aktuellen $BASE besetzt, z.B.:

```
DECL FRAME STARTPUNKT={X 15.2,Y 2.46,Z 20.5,A -35.5,B 9.0,C 16.87}
```

Ist die Datenliste nicht angelegt worden, erscheint die Fehlermeldung: *Die Variable "STARTPUNKT" konnte nicht angelegt werden*.



Dokumentation zum KRL-Assistant
Abschnitt 11 (Datenlisten)

Sofern Sie eine Bewegungsanweisung über die Inline-Formulare angelegt haben, können Sie die über das Formular geteachten Punkte später auch in einer KRL-Bewegungsanweisung verwenden:

Die Punkte werden in der dazugehörigen Datenliste mit dem im Formular angegebenen Namen und vorangestelltem X abgelegt (daher sind für Punktnamen in Inline-Formularen auch höchstens 11 statt 12 Zeichen zulässig).

Sie können den Punkt `P7` **im Inline-Formular**

also später als `XP7` in einer KRL-Anweisung ansprechen:

```
LIN XP7
```

Auch hier ist zu beachten, daß in beiden Fällen das gleiche Basiskoordinatensystem verwendet werden muß, damit der gleiche Punkt angefahren wird!!

NOTIZEN: