

## 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.

Bewegungs-  
anweisungen

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.

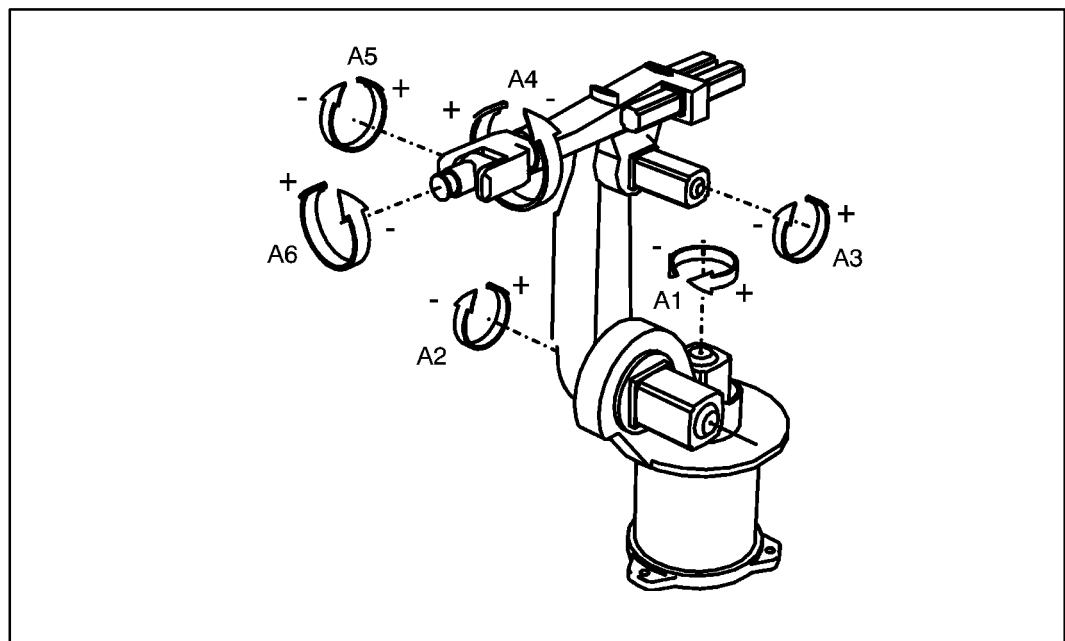
### 4.1 Verwendung verschiedener Koordinatensysteme

Koordinaten-  
system

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 Koordinatensysteme erfolgen:

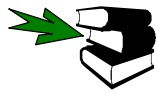
achsspezi-  
fisch

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 Roboter gelenkwinkel angegeben werden, um Lage und Orientierung eindeutig festzulegen (s. Abb. 8).



**Abb. 8** 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.

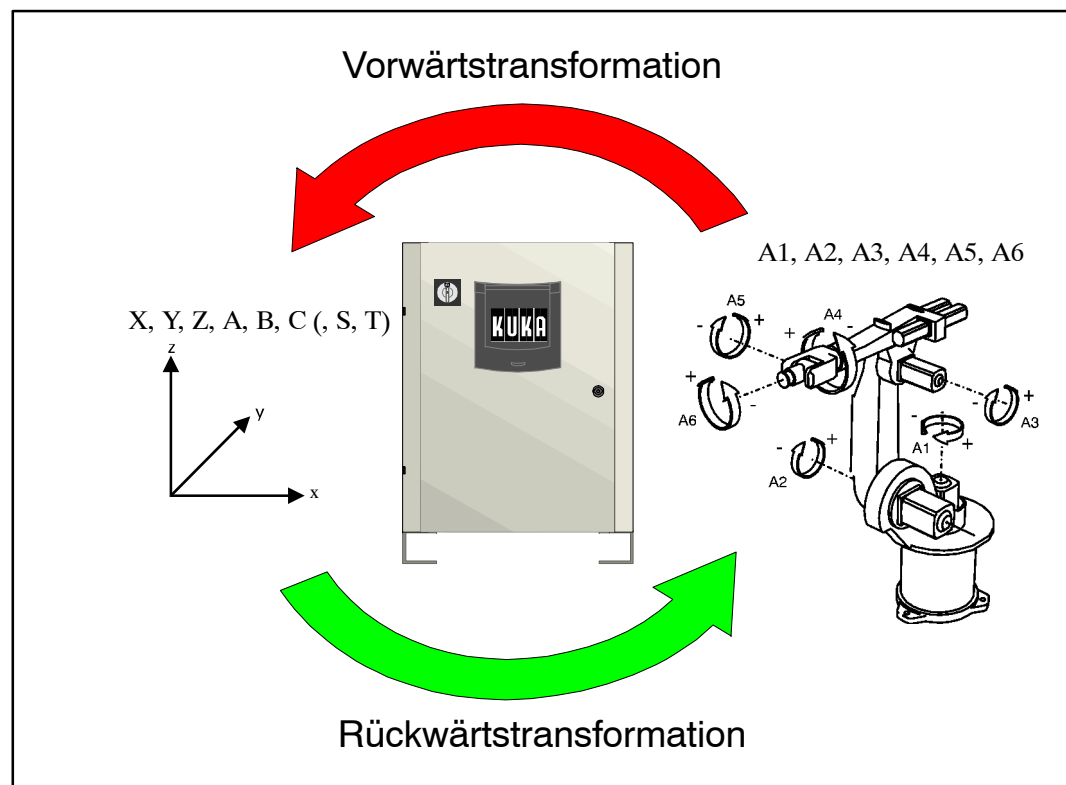


Weitere Informationen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Strukturen]**.

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.

Koordinaten-  
transformation

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. 9).

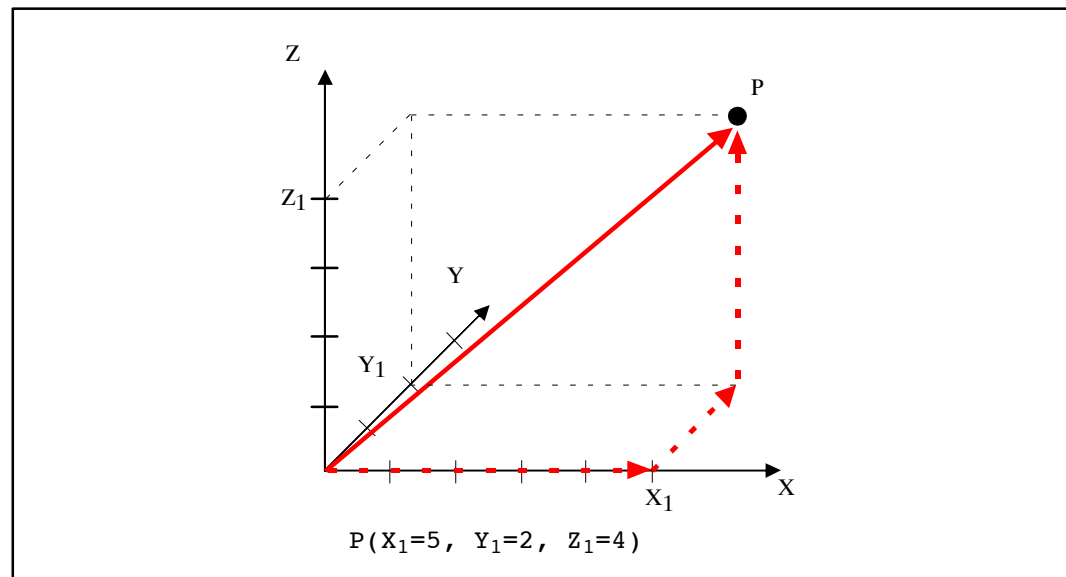


**Abb. 9** Koordinatentransformation

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. 10).



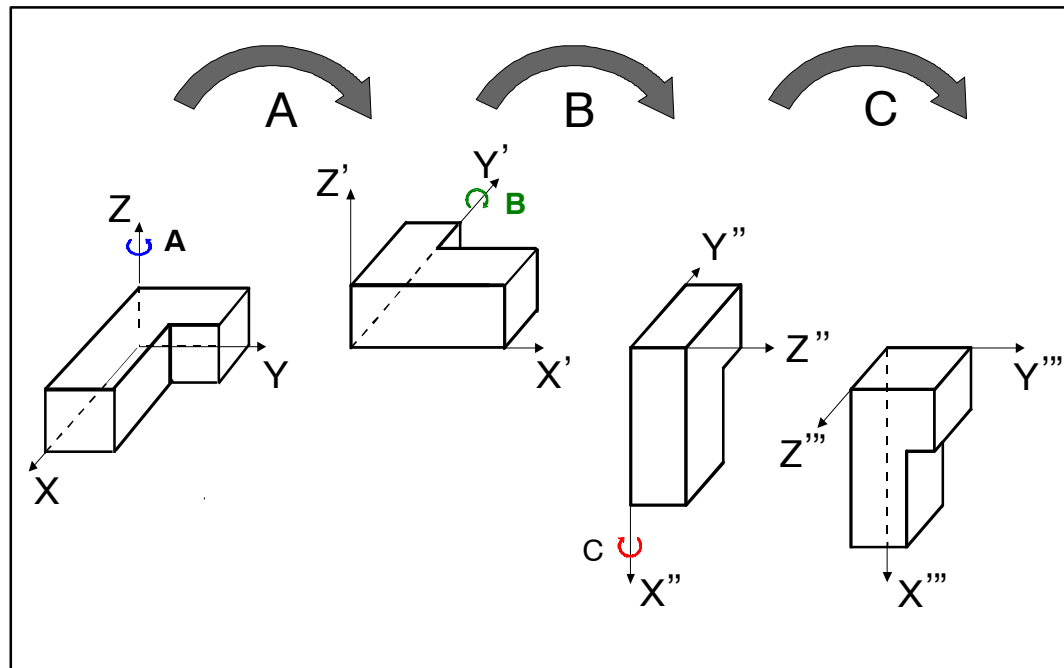
**Abb. 10** Translatorische Beschreibung der Lage eines Punktes

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 KRC1 mit A, B und C bezeichneten Winkel beschreiben Drehungen um die Koordinatenachsen  $Z$ ,  $Y$  und  $X$ . Dabei ist die Reihenfolge der Rotationen einzuhalten:

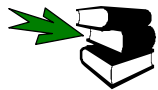
1. Drehung um die  $Z$ -Achse um den Winkel A
2. Drehung um die neue  $Y$ -Achse um den Winkel B
3. Drehung um die neue  $X$ -Achse um den Winkel C

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. 11).



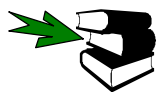
**Abb. 11** Rotatorische Beschreibung der Orientierung eines Punktes

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`.



Weitere Informationen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Strukturen]**.

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.



Weitere Informationen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Strukturen]** und Abschnitt **[Bewegungsbefehle]**.

In der KR C1 sind folgende kartesische Koordinatensysteme vordefiniert (Tab. 1 und Abb. 12):

Koordinatensystem	Systemvariable	Status
Weltkoordinatensystem	\$WORLD	schreibgeschützt
Roboterkoordinatensystem	\$ROBROOT	schreibgeschützt (in R1\MADA\\$_MASCHINE.DAT änderbar)
Werkzeugkoordinatensystem	\$TOOL	beschreibbar
Basis(Werkstück-)koordinatensystem	\$BASE	beschreibbar

Tab. 1 Vordefinierte Koordinatensysteme

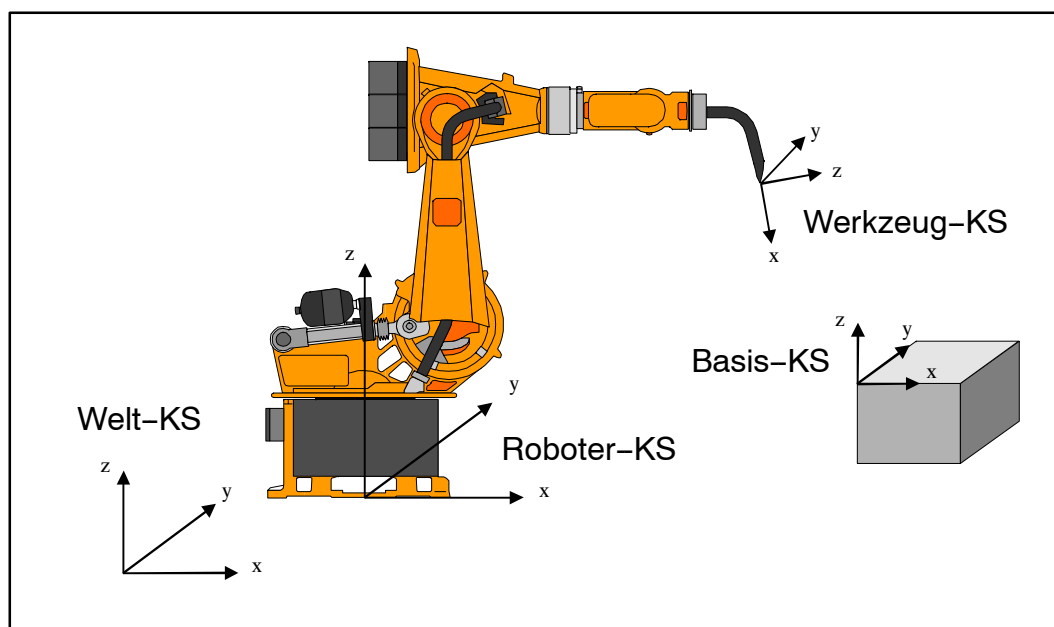


Abb. 12 Kartesische Koordinatensysteme bei Robotern

### Weltkoordinatensystem

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

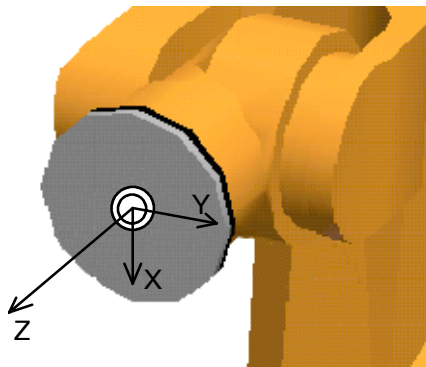
### Roboterkoordinatensystem

Dieses Koordinatensystem befindet sich im Roboterfuß und dient als Bezugskoordinatensystem für den mechanischen Roboterbau. 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.

### Werkzeugkoordinatensystem

Das Werkzeugkoordinatensystem hat seinen Ursprung in der Werkzeugspitze. Die Orientierung kann dabei so gewählt werden, daß seine X-Achse mit der Werkzeug-Stoßrichtung

identisch ist und aus dem Werkzeug heraus zeigt. Bei einer Bewegung der Werkzeugschneidspitze wird das Werkzeugkoordinatensystem mitbewegt.



Bei der Auslieferung liegt das Werkzeugkoordinatensystem im Roboterflansch (Z-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 Werkzeugschneidspitze 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  $\$BASE = \$WORLD$ .

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 (Werkzeug am Roboterflansch) die aktuelle Position ( $\$POS\_ACT$ ) bezogen auf das  $\$BASE$ -Koordinatensystem (s. Abb. 13).

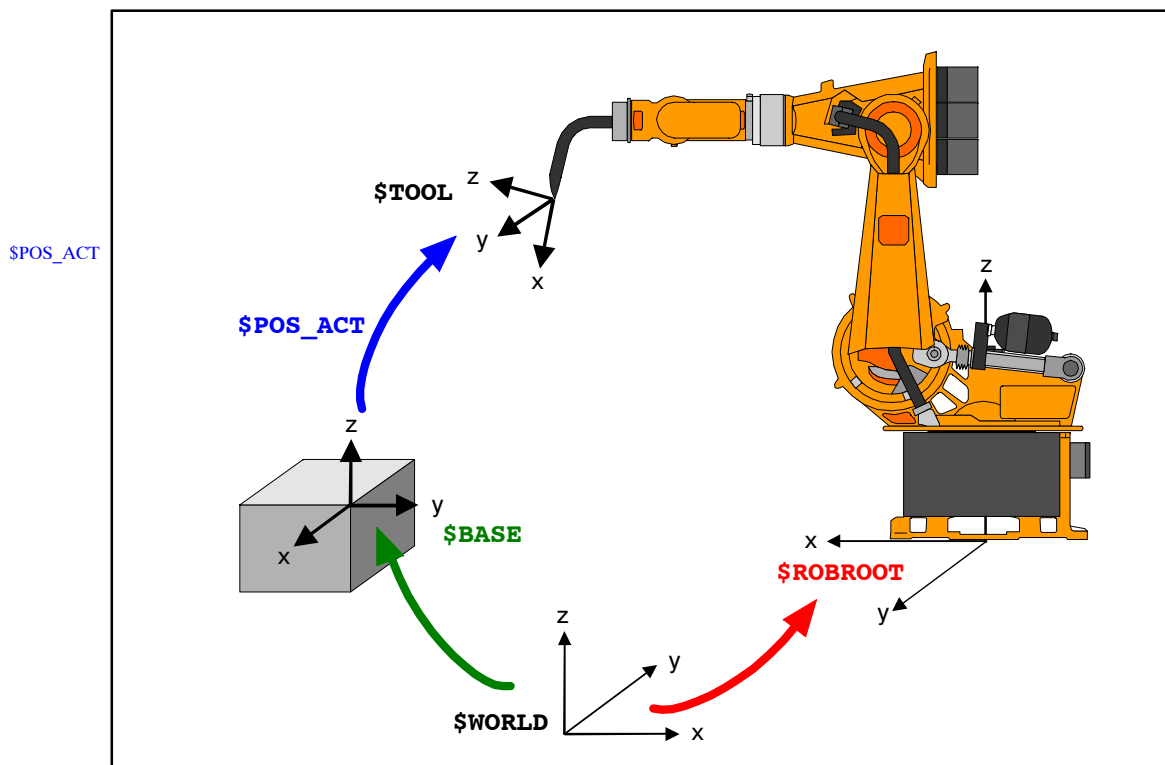


Abb. 13 Kinematische Kette bei basisbezogener Interpolation

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.



Die Variable **\$TOOL** bezieht sich immer auf das Werkzeug bzw. Werkstück, welches sich am Roboter befindet. Die Variable **\$BASE** dagegen bezieht sich immer auf das externe Werkzeug bzw. Werkstück.

greiferbezog.  
Interpolation

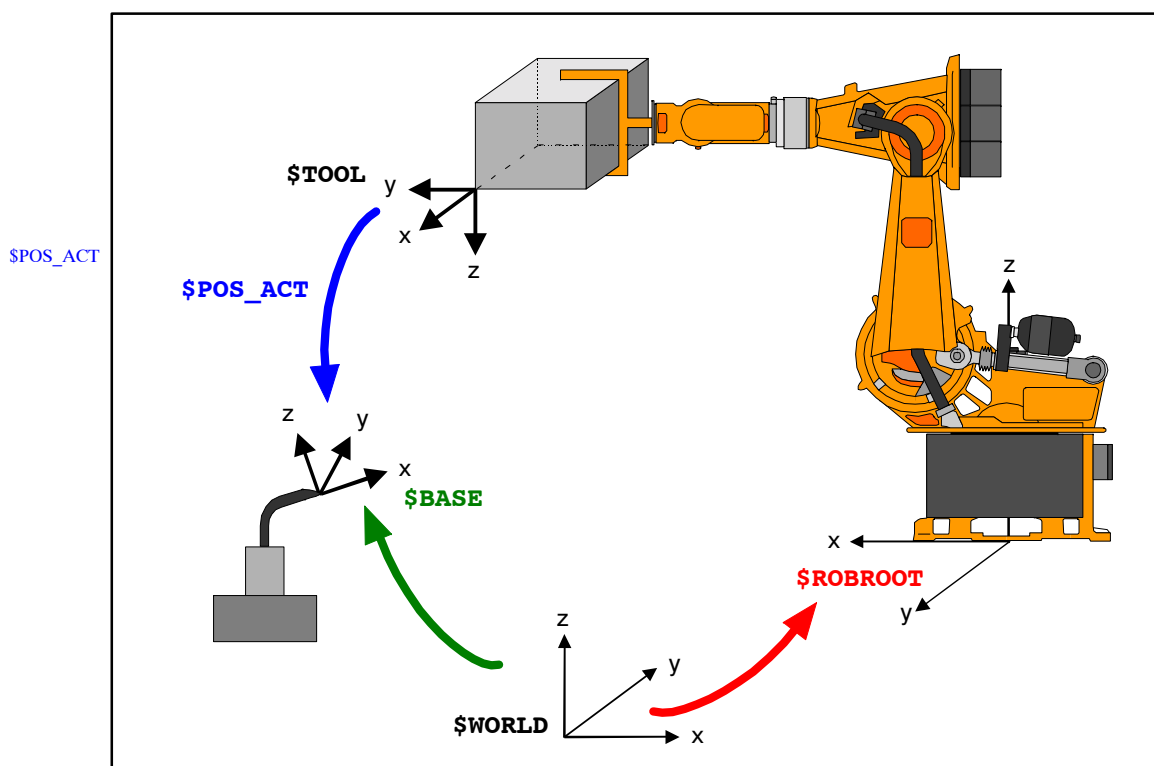
Da nun Werkstück und Werkzeug ihre Position getauscht haben, die Bewegung aber weiter bezüglich des Werkstücks erfolgen soll, muß die Interpolation der Bewegungsbahn jetzt über das **\$TOOL**-Koordinatensystem erfolgen. Diese Zuordnung der Interpolationsart erfolgt implizit beim Verwenden eines normalen oder externen TCP's. Mit der Systemvariablen **\$IPO\_MODE** können Sie diese Interpolationsart definieren. Die Programmzeile

```
$IPO_MODE = #TCP
```

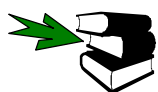
ermöglicht eine greiferbezogene Interpolation im **\$TOOL**-Koordinatensystem. Die aktuelle Position **\$POS\_ACT** wird nun also bezüglich **\$TOOL** berechnet (s. Abb. 14). Mit

```
$IPO_MODE = #BASE
```

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



**Abb. 14** Kinematische Kette bei greiferbezogener Interpolation



Ein Beispiel zur Verschiebung von Koordinatensystemen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Geometrischer Operator]**.

## 4.2 Punkt-zu-Punkt Bewegungen (PTP)

### 4.2.1 Allgemein (Synchron-PTP)

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.

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. 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.

Ist die Beschleunigungsanpassung oder das höhere Fahrprofil aktiviert (**\$ADAP\_ACC**=#STEP1, **\$OPT\_MOVE**=#STEP1) werden die Achsen phasensynchron verfahren, d.h alle Achsen befinden sich gleichzeitig in Beschleunigungs-, Konstantfahr- und Verzögerungsphase.



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.

Die zeitsynchrone Bewegungsführung vermindert die mechanische Belastung des Roboters, da Motor- und Getriebemomente für alle Achsen mit kürzeren Verfahrwegen herabgesetzt werden.

Die phasensynchrone Bewegungsführung führt (zusätzlich) zu einer Bewegungsbahn, die unabhängig von der programmierten Geschwindigkeit und der programmierten Beschleunigung im Raum stets gleich verläuft.



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Überschleifbewegungen]**.



#### **4.2.2    Höheres Fahrprofil**

Standardmäßig wird für PTP-Bewegungen das höhere Fahrprofil verwendet. Mit diesem Modell wird bei **PTP-Einzelsätzen** und **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. Die Geschwindigkeit wird hierbei immer so angepaßt, daß die Momente nicht überschritten werden.

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.

Die Geschwindigkeitszuweisungen und die Beschleunigungsgrenzwerte (Angabe in Prozent) können für jede Achse einzeln eingestellt werden. Dieser Grenzwert wirkt aber nicht unmittelbar auf die Beschleunigung, sondern auf das Beschleunigungsmoment der Achse. D.h. ein Achsbeschleunigungswert von 50% verringert die Beschleunigung nicht unbedingt um die Hälfte.

### 4.2.3 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

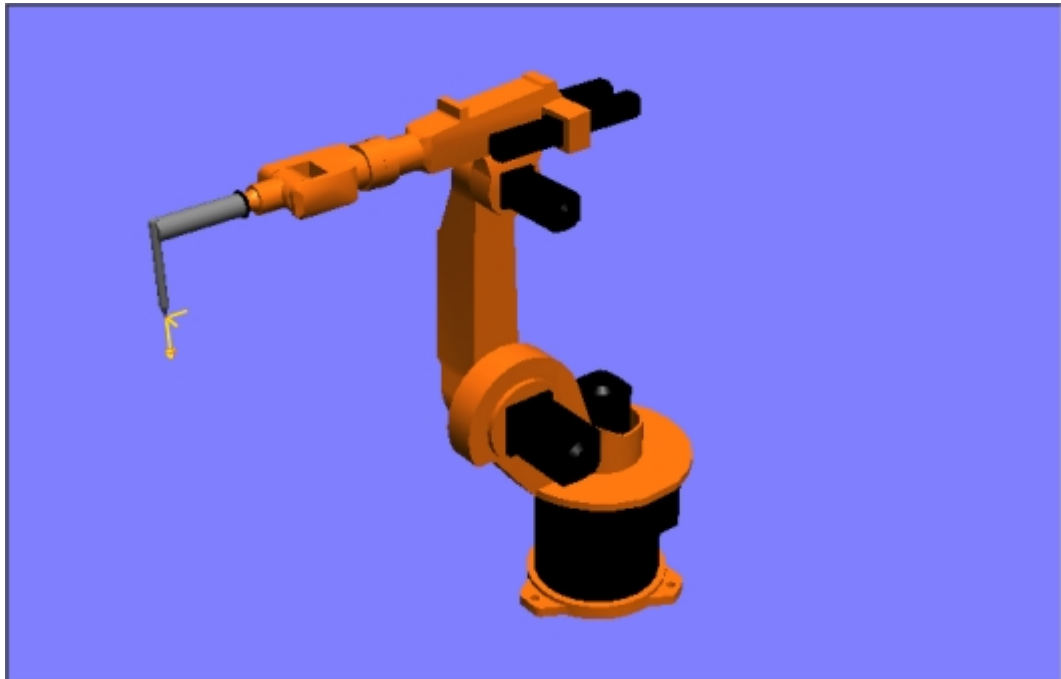
PTP {AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

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. Beispielsweise 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°.



**Abb. 15** Mechanische Nullstellung

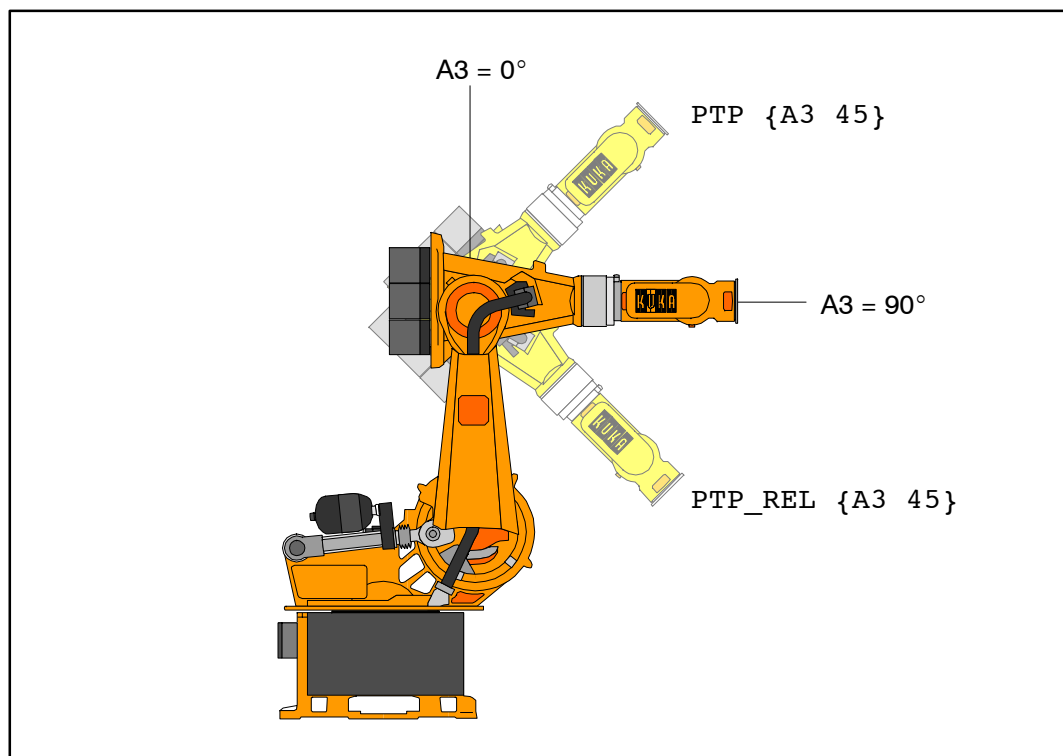
Werden bei der Angabe der Achskoordinaten einzelne Komponenten weggelassen, so verfährt der Roboter nur die angegebenen Achsen, die anderen verändern ihre Lage nicht. Mit

PTP {A3 45}

wird also z.B. nur die Achse 3 auf  $45^\circ$  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  $45^\circ$  weiter, sondern verfährt auf die absolute  $45^\circ$ -Lage der Achse.

Zum relativen Verfahren dient die Anweisung PTP\_REL. Um also z.B. die Achsen 1 und 4 um je  $35^\circ$  zu verdrehen, programmieren Sie einfach:

PTP\_REL {A1 35,A4 35}



**Abb. 16** Unterschied zwischen absoluten und relativen achsspezifischen 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 Programmart den bereits zurückgelegten Weg nicht berücksichtigen und wird die programmierte Relativedistanz 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.

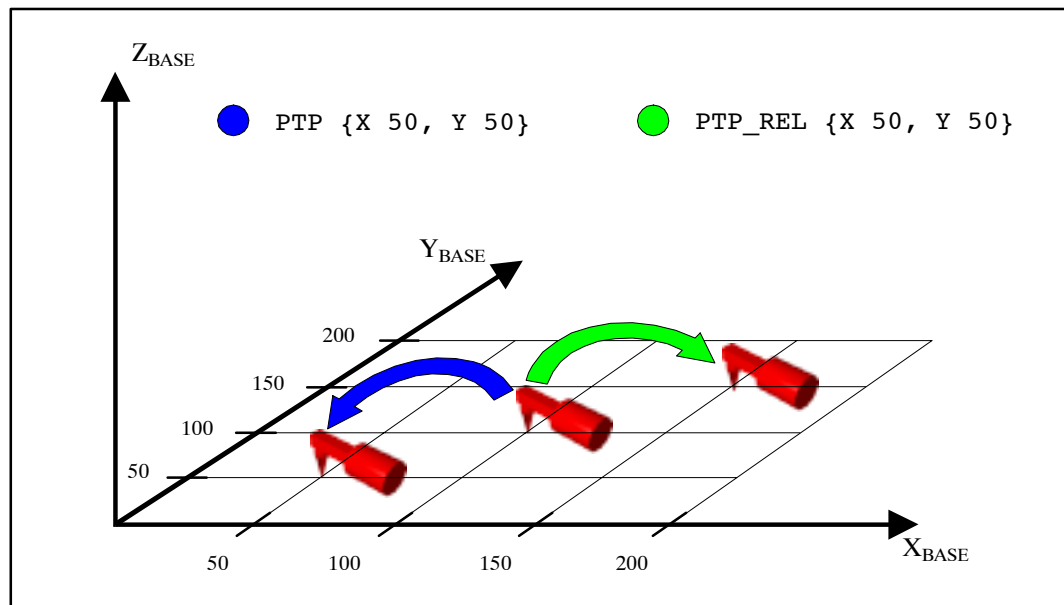
Auch bei Zielpunktangabe in kartesischen Koordinaten können wieder einzelne Komponenten der Geometrieangabe weggelassen werden. Die Anweisung

```
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. 17** Unterschied zwischen absoluten und relativen kartesischen 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.



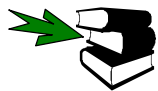
Die Basisverschiebung mit Hilfe des Doppelpunkt-Operators hat außerdem einen entscheidenden Vorteil gegenüber einer Neubesetzung von `$BASE`:

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,

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

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.



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Rechnervorlauf]**.

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.



Bei einem TouchUp mit dem standardmäßig ausgelieferten Basis-Paket werden für jeden Punkt automatisch \$BASE und \$TOOL in der Datenliste abgelegt.

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
```

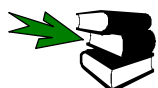


In diesem Beispiel werden außerdem die notwendigen Belegungen der Geschwindigkeiten, Beschleunigungen sowie \$BASE- und \$TOOL-Koordinatensystemen nicht mehr "von Hand" vorgenommen. Stattdessen wird das standardmäßig vorhandene "BAS.SRC" hierzu verwendet. Dazu muß es zunächst mit der EXT-Anweisung dem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

Mit der Initialisierungsanweisung

```
INI BAS (#INITMOV, 0)
```

werden schließlich alle wichtigen Systemvariablen mit Standardwerten besetzt.



Weitere Informationen finden Sie im Kapitel **[Unterprogramme und Funktionen]** Abschnitt **[Vereinbarungen]**.

**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 "Status" und "Turn" zu programmieren, damit eine eindeutige Ausgangslage definiert wird. Da bei Bahnbewegungen (s. 4.3) "S" und "T" nicht berücksichtigt werden, muß 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 behält den alten S-Wert bei und wählt den T-Wert, mit dem sich der kürzeste, fahrbare axiale Weg ergibt, welcher aufgrund der einmaligen Programmierung von "S" und "T" im ersten PTP-Satz bei verschiedenen Programmierabläufen 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  $+180^\circ$  bzw. kleiner  $-180^\circ$  sind, ohne besondere Verfahrenstrategie (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  $\geq 0^\circ$

Bit x = 1: Winkel der Achse x  $< 0^\circ$

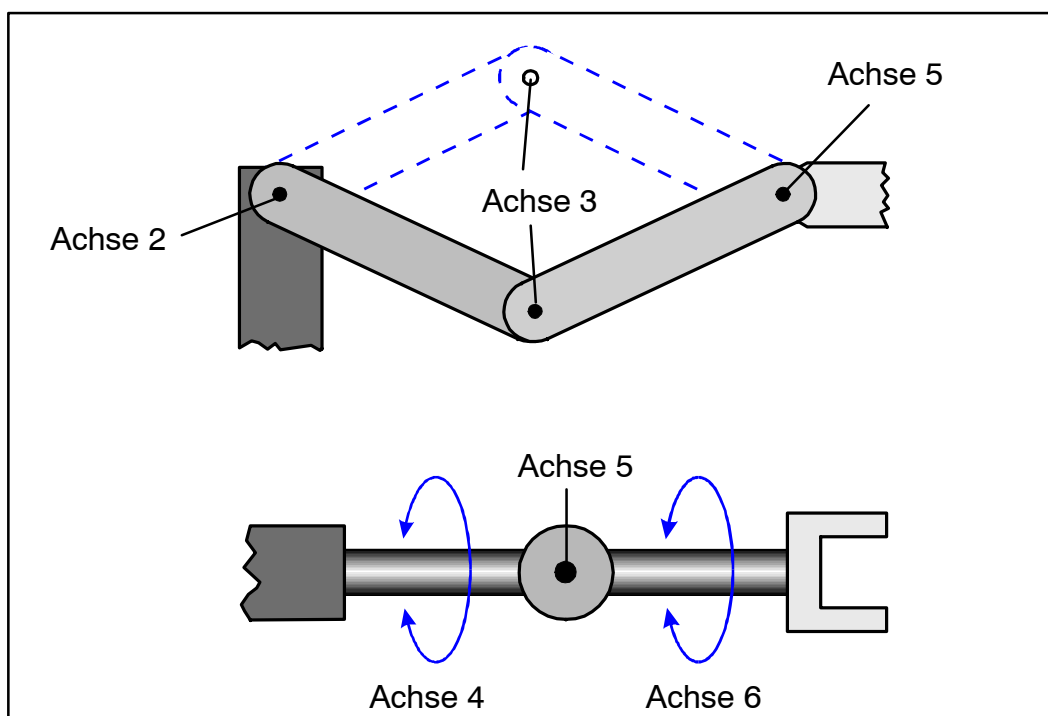
Wert	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	$A6 \geq 0^\circ$	$A5 \geq 0^\circ$	$A4 \geq 0^\circ$	$A3 \geq 0^\circ$	$A2 \geq 0^\circ$	$A1 \geq 0^\circ$
1	$A6 < 0^\circ$	$A5 < 0^\circ$	$A4 < 0^\circ$	$A3 < 0^\circ$	$A2 < 0^\circ$	$A1 < 0^\circ$

**Tab. 2** 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 weglassen werden).

**Status** Mit dem Status S werden Mehrdeutigkeiten in der Achsstellung gehandelt (s. Abb. 18). S ist daher abhängig von der jeweiligen Roboterkinematik.

Mehrdeutigkeiten



**Abb. 18** 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 6-achsigen Knickarmroboter nach folgender Tabelle gesetzt:

Wert	Bit 2	Bit 1	Bit 0
0	$0^\circ \leq A5 < 180^\circ$ $A5 < -180^\circ$	$A3 < \phi$ ( $\phi$ hängt vom Robotertyp ab)	Grundbereich
1	$-180^\circ \leq A5 < 0^\circ$ $A5 \geq 180^\circ$	$A3 \geq \phi$ ( $\phi$ hängt vom Robotertyp ab)	Überkopfbereich

**Tab. 3** Status-Bits für 6-achsige Knickarmroboter

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

Handwurzelpunkt: Schnittpunkt der Handachsen

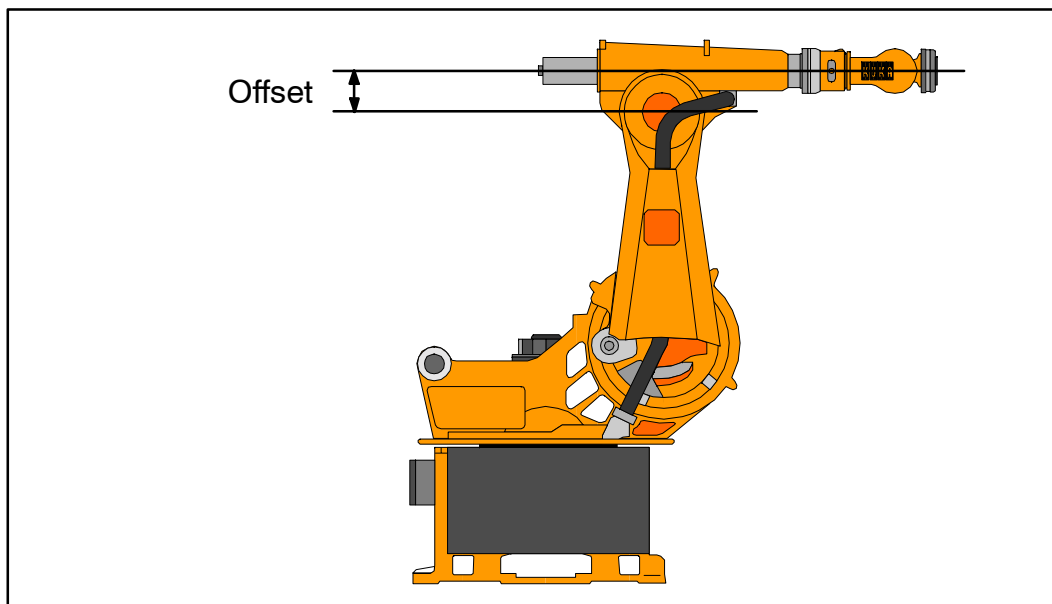
A1-Koordinatensystem: Steht die Achse 1 auf  $0^\circ$ , ist es mit dem \$ROBROOT-Koordinatensystem identisch. Bei Werten ungleich  $0^\circ$  wird es mit der Achse 1 mitbewegt.

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  $< 0^\circ$  ist, ansonsten ist Bit 1 = 1. Bei Robotern mit einem Offset zwischen Achse 3 und 4 (z.B. KR 30, s. Abb. 19), ist die Winkellage, in der sich der Wert von Bit 1 ändert, von der Größe dieses Offsets abhängig.



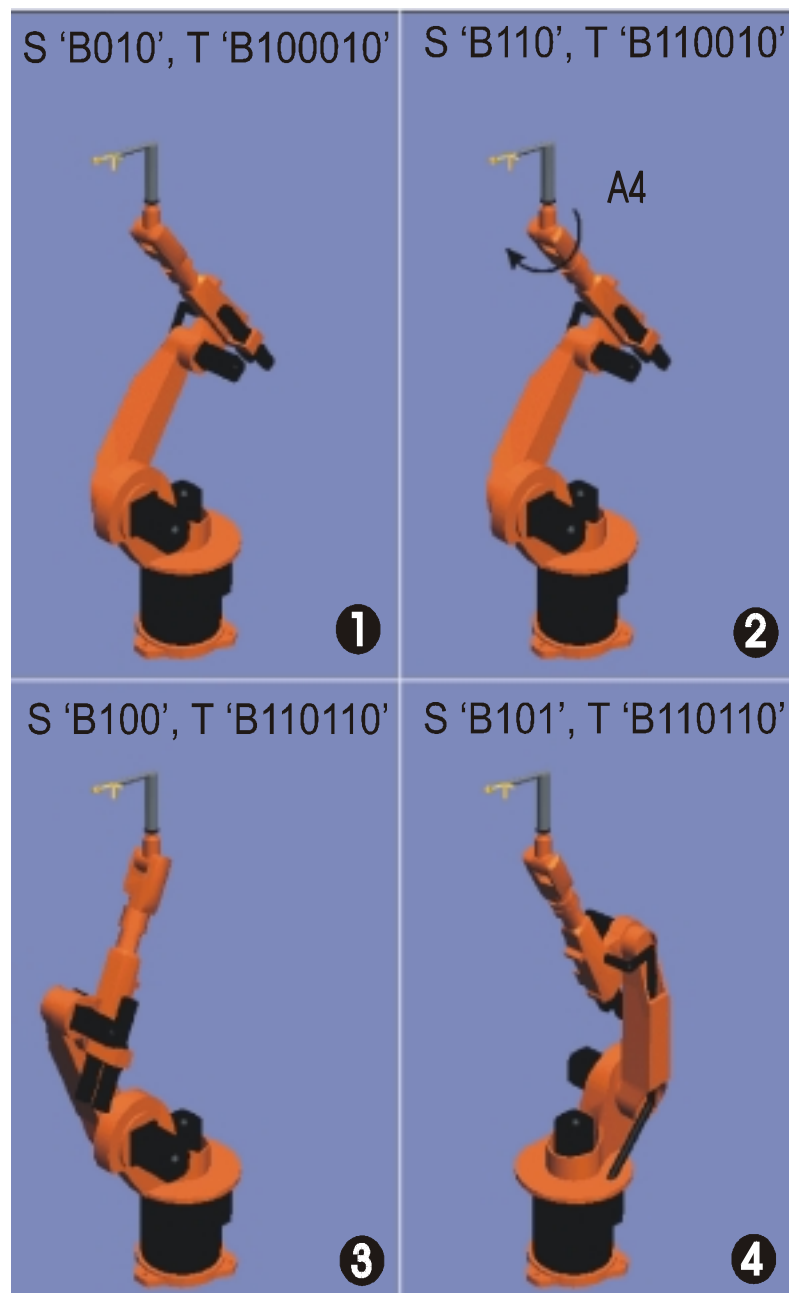
**Abb. 19** Offset zwischen Achse 3 und 4 bei einem KR 30

In Abb. 20 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.  $45^\circ$ , Achse 3 ca.  $80^\circ$ .

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

Von Stellung 2 nach 3 ändert sich nun die Armkonfiguration. Achse 3 dreht auf eine Winkellage von ca.  $-50^\circ$ , 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  $180^\circ$  gedreht. Status-Bit 0 wird zu 1.



**Abb. 20** Auswirkungen der Status-Bits auf die Roboterstellung



**NOTIZEN:**

## 4.3 Bahnbewegungen (CP–Bewegungen = Continuous Path)

### 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. 4 gibt einen Überblick über die zu programmierenden Systemvariablen und deren Einheit.

	Variablen-name	Datentyp	Einheit	Funktion
<b>Geschwindigkeiten</b>	\$VEL.CP	REAL	m/s	Bahngeschwindigkeit
	\$VEL.ORI1	REAL	°/s	Schwenkgeschwindigkeit
	\$VEL.ORI2	REAL	°/s	Drehgeschwindigkeit
<b>Beschleunigungen</b>	\$ACC.CP	REAL	m/s <sup>2</sup>	Bahnbeschleunigung
	\$ACC.ORI1	REAL	°/s <sup>2</sup>	Schwenkbeschleunigung
	\$ACC.ORI2	REAL	°/s <sup>2</sup>	Drehbeschleunigung

**Tab. 4** Systemvariablen für Geschwindigkeiten und Beschleunigungen



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

Weiterhin werden bei CP–Bewegungen die Achsgeschwindigkeit und –beschleunigung überwacht und bei Überschreiten der Überwachungsgrenzen, die in den Systemvariablen \$ACC\_ACT\_MA und \$VEL\_ACT\_MA definiert sind, eine Stoppreaktion ausgelöst und eine Fehlermeldung ausgegeben. Diese Grenzen liegen standardmäßig bei 250% der nominalen Achsbeschleunigung (Acceleration) und 110% der nominalen Achsgeschwindigkeit (Velocity). Diese Überwachungsbereiche gelten für alle Betriebsarten und das Handverfahren.

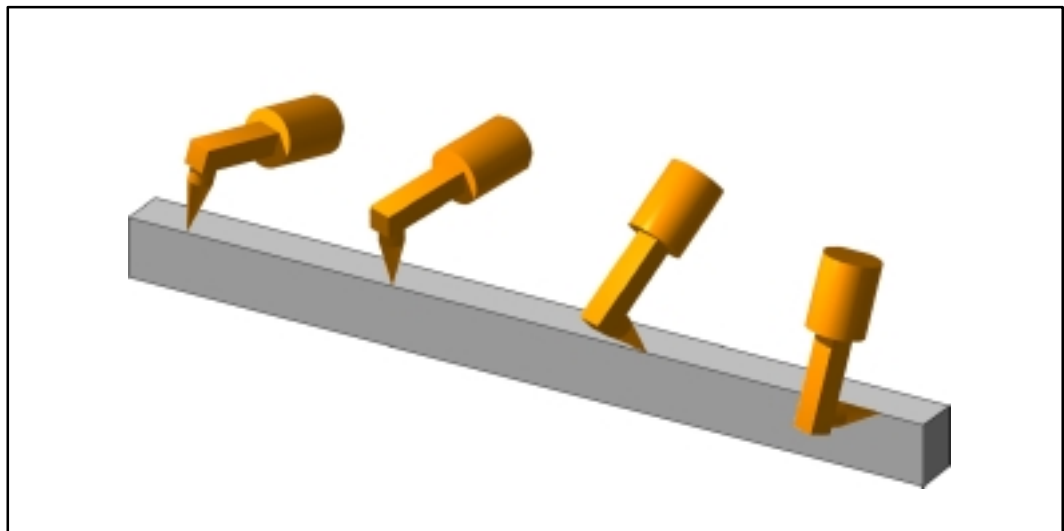
Es besteht die Möglichkeit über die Systemvariable \$CP\_VEL\_TYPE die Achsvorschübe und somit die Beschleunigung und die Geschwindigkeit zu reduzieren, um ein Ansprechen der Überwachungsgrenzen (Stoppreaktion) zu vermeiden. Als Defaulteinstellung ist die Variable mit #CONSTANT belegt, d.h. die Reduzierung ist nicht aktiv. Möchte man diese Funktion in der Betriebsart T1 setzt man #VAR\_T1 (in T1 werden niedrigere Achsgeschwindigkeiten und –beschleunigungen verwendet) und in allen Betriebsarten #VAR\_ALL. Im Handbetrieb ist die Reduzierung immer aktiv.

Ist es erwünscht, daß im Testbetrieb eine Meldung erscheint, wenn die Bahngeschwindigkeit reduziert wird, muß die Systemvariable \$CPVELREDMELD auf den Wert 1 gesetzt werden.

### 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. 21):

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



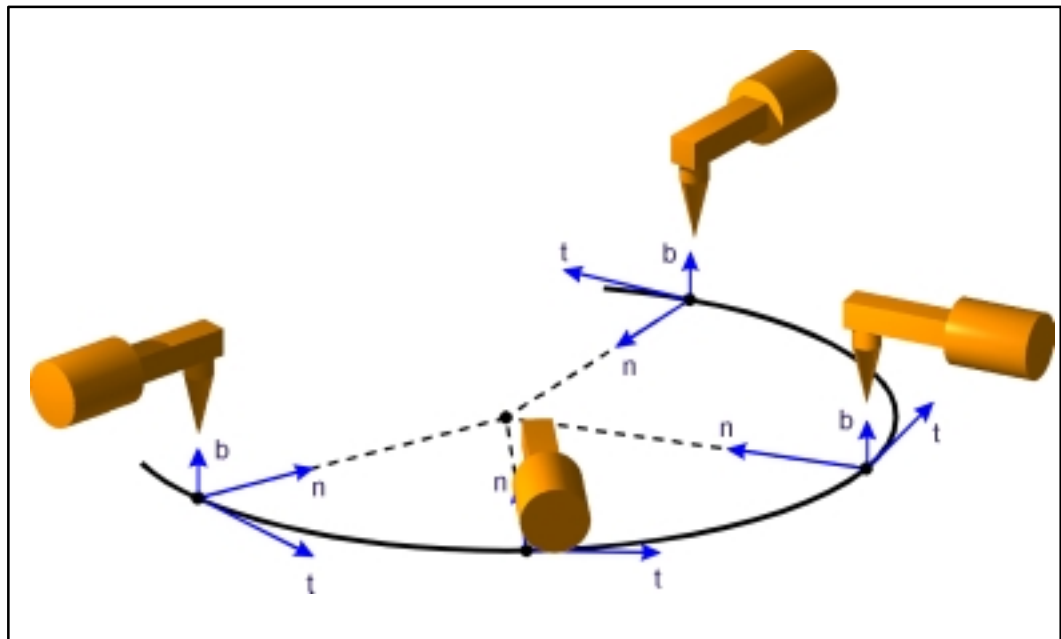
**Abb. 21** Orientierungsänderung bei einer Linearbewegung

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

- `$CIRC_TYPE = #BASE` raumbezogene Orientierungsführung während der Kreisbewegung. Dieser Wert wird auch bei der Initialisierung mittels `BAS (#INITMOV, 0)` gesetzt.
- `$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. 22).



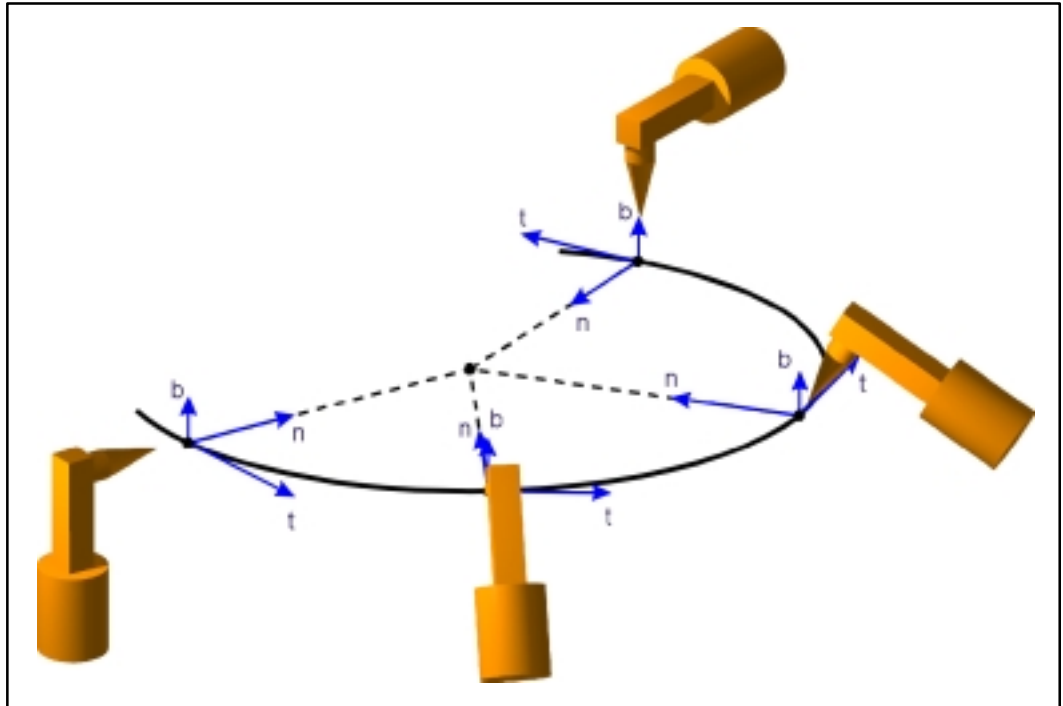
**Abb. 22** Konstante bahnbezogene Orientierungsführung bei Kreisbewegungen

Das bahnbegleitende Dreibein setzt sich aus dem Kreistangentenvektor **t**, dem Normalenvektor **n** und dem Binormalenvektor **b** zusammen. Die Werkzeugorientierung wird in Abb. 22 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.

Im gezeigten Beispiel wird die Werkzeugorientierung relativ zum bahnbegleitenden Dreibein während der Bewegung vom Start- zum Zielpunkt nicht verändert (`$ORI_TYPE=#CONST`).

variabel +  
bahnbezog.

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. 23). Die Orientierungsführung im bahnbegleitenden Dreibein ist bei Kreisbewegungen also vollkommen analog zur Orientierungsführung bei Linearbewegungen.

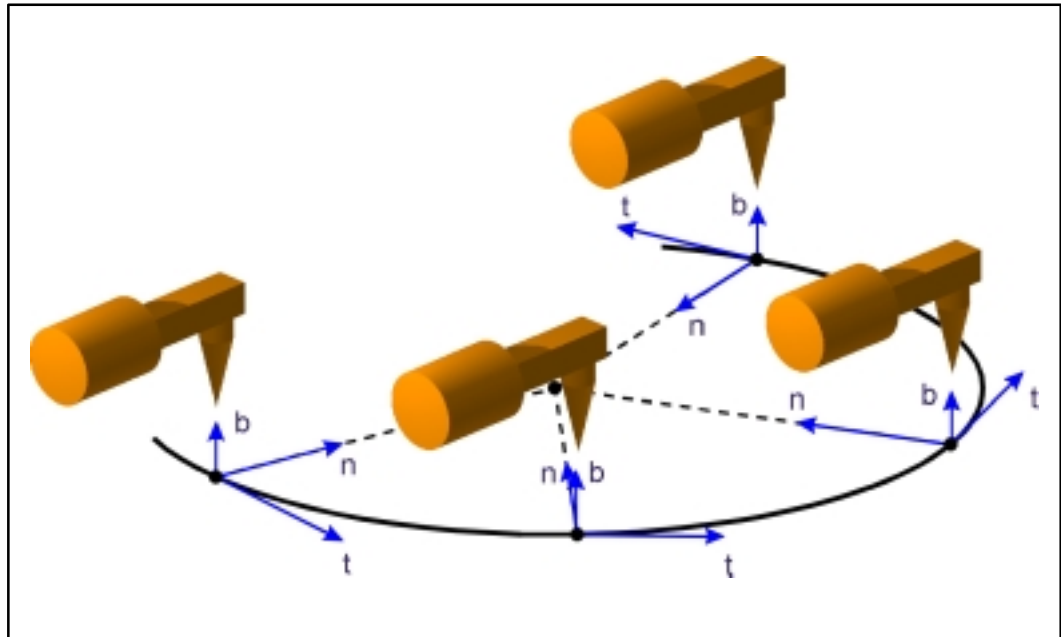


**Abb. 23** Variable bahnbezogene Orientierungsführung bei Kreisbewegungen

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. 24) während einer Kreisbewegung (z.B. Kleberauftrag mit rotationsymmetrischer Kleberdüse).

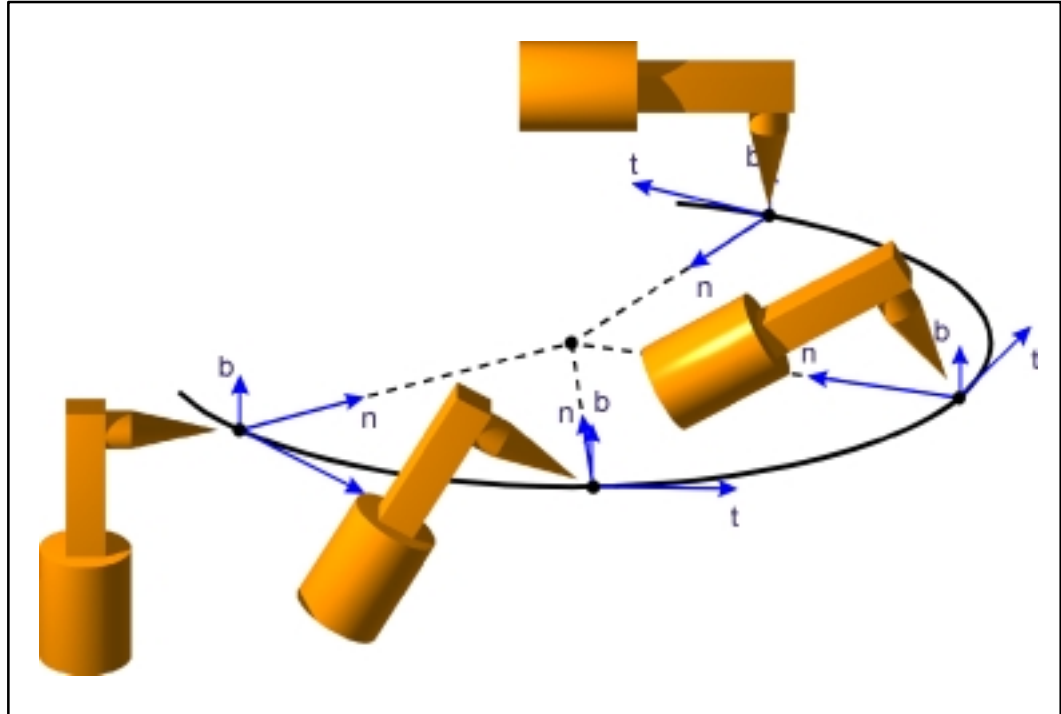


**Abb. 24** Konstante raumbezogene Orientierungsführung bei Kreisbewegungen



variabel +  
raumbezog.

Eine raumbezogene Orientierungsänderung (\$ORI\_TYPE=#VAR) zwischen Start- und Zielposition wird wieder durch Überlagerung von Schwenk- und Drehbewegungen ausgeführt (s. Abb. 25). In diesem Fall allerdings relativ zum Basiskoordinatensystem.



**Abb. 25** Variable raumbezogene Orientierungsführung bei Kreisbewegungen

In Tab. 5 sind nochmals die Voreinstellungen der Systemvariablen zur Orientierungsführung bei Bahnbewegungen aufgelistet:

	im System	durch <b>BAS</b> (#INITMOV, 0)
<b>\$ORI_TYPE</b>		#VAR
<b>\$CIRC_TYPE</b>	#PATH	#BASE

**Tab. 5** 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.

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.

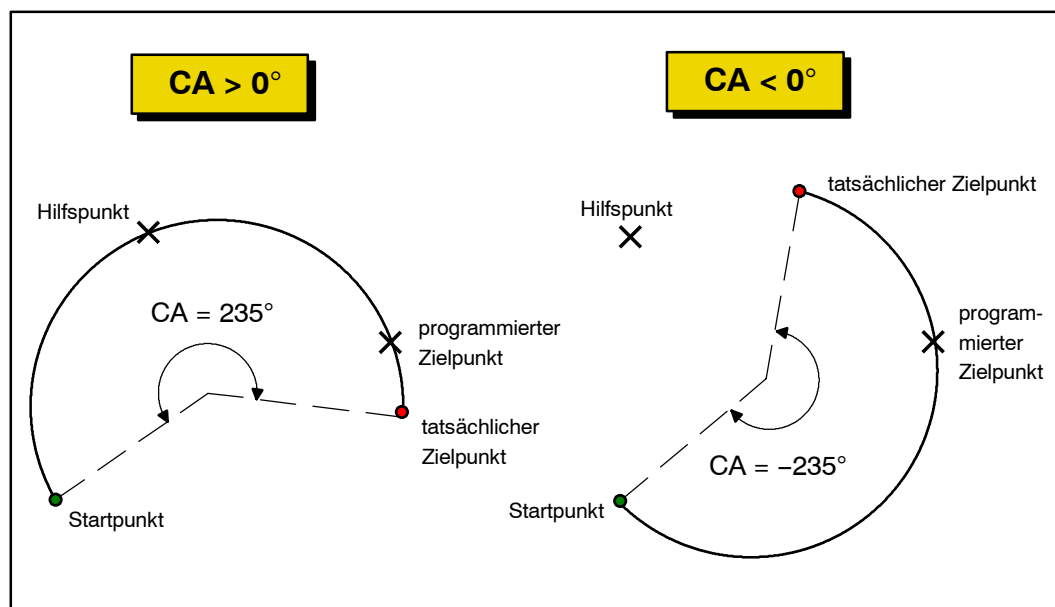
Der abzufahrende Kreisbogen kann entsprechend dem Kreiswinkel verlängert oder verkürzt werden. Die programmierte Zielorientierung wird dann im tatsächlichen Zielpunkt erreicht. Über das Vorzeichen des Kreiswinkels kann der Drehsinn, d.h. die Richtung in der der Kreisbogen abgefahren werden soll, festgelegt werden (s. Abb. 26):

$CA > 0^\circ$  im programmierten Sinn (Startpunkt → Hilfspunkt → Zielpunkt)

$CA < 0^\circ$  entgegen dem programmierten Sinn (Startpunkt → Zielpunkt → Hilfspunkt)



Der Wert des Kreiswinkels ist nicht begrenzt. Insbesondere können auch Vollkreise ( $> 360^\circ$ ) programmiert werden.



**Abb. 26** 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} ; Umoorientierung 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
    
```

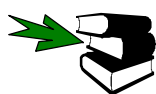
## 4.4 Rechnervorlauf

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.



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Überschleifbewegungen]**.

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 vorausseilen 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. 27).

```

13
14 $ADVANCE=1
15
16 → LIN {X 1620,Y 0,Z 1910,A 0,B 90,C 0} ← Hauptlaufzeiger
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 ← Der Vorlaufzeiger befindet sich an dieser
25                               Stelle; $ADVANCE = 1
26 SPANNUNG=110
27
28 PTP PUNKT7
29
30
31

```

Abb. 27 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, daß die Anweisungen von Zeile 16 bis 22 parallel zur Bewegungsausführung komplett abgearbeitet wurden, und die Bewegungsdaten für die PTP-Bewegungen in Zeile 24 gerade aufbereitet werden.



Um ein Überschleifen zu ermöglichen, muß mindestens ein Rechnervorlauf von 1 eingestellt sein (die Variable \$ADVANCE besitzt standardmäßig den Wert "3". Maximal sind 5 Vorlaufschritte möglich).

In einem Interrupt-Unterprogramm ist kein Rechnervorlauf möglich. Die Steuerung arbeitet Interruptprogramme immer zeilenweise ab, daher ist ein Überschleifen in Interruptprogrammen nicht möglich.

Die Voreinstellungen von \$ADVANCE :

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

automat.  
Vorlaufstop

Anweisungen und Daten, die die Peripherie beeinflussen (z.B. Ein-/Ausgabeanweisungen), oder sich auf den aktuellen Roboterzustand beziehen, lösen einen **Vorlaufstop** aus (s. Tab. 6). Dies ist notwendig, um die korrekte, zeitliche Reihenfolge der Anweisungen und der Roboterbewegungen sicherzustellen.

Anweisungen	ANOUT ON	ANOUT OFF
	ANIN ON	ANIN OFF
	DIGIN ON	DIGIN OFF
	PULSE	
	HALT	WAIT
	CREAD	CWRITE COPEN CCLOSE
	SREAD	SWRITE
	CP-PTP-Kombination ohne Überschleifen	
Anweisungen in Verbindung mit einem Interrupt	END (falls im Modul ein nicht globaler Interrupt definiert wurde)	
	INTERRUPT DECL (falls der Interrupt bereits deklariert wurde)	
	RESUME ohne BRAKE	
gebräuchliche Systemvariablen	\$ANOUT[Nr]	\$ANIN[Nr]
	\$DIGIN1	\$DIGIN2 ... \$DIGIN6
	\$OUT[Nr]	\$IN[Nr]
	\$AXIS_ACT	\$AXIS_BACK \$AXIS_FOR \$AXIS_RET
	\$POS_ACT	\$POS_BACK \$POS_FOR \$POS_RET
	\$AXIS_INC	\$AXIS_INT \$POS_ACT_MES \$POS_INT
	\$TORQUE_AXIS	\$ASYNC_AXIS
	\$TECH[X].MODE, \$TECH[X].CLASS bei gewissen Operationen	
	\$LOAD, \$LOAD_A1, \$LOAD_A2, \$LOAD_A3 (falls es sich um einen absolutgenauen Roboter mit Lastwechsel handelt)	

<b>weitere Systemvariablen</b>	\$ALARM_STOP    \$AXIS_ACTMOD    \$INHOME_POS    \$ON_PATH \$EM_STOP       \$EXTSTARTTYP    \$REVO_NUM     \$SAFETY_SW \$ACT_TOOL       \$PAL_MODE       \$ACT_BASE     \$ACT_EX_AX \$OV_PRO         \$WORKSPACE     \$IBUS_OFF     \$IBUS_ON \$ASYNC_EX_AX _DECOUPLE
<b>importierte Variablen</b>	alle, bei Zugriff
<b>Sonstiges</b>	Findet zwischen überschiffenen Sätzen ein Filterwechsel statt, wird ein Vorlaufstopp ausgelöst.

**Tab. 6** Anweisungen und Variablen die einen automatischen Vorlaufstopp auslösen

CONTINUE

In Anwendungsfällen, bei denen dieser Vorlaufstopp 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 Vorlaufstopp aus, tut aber sonst nichts:

`WAIT SEC 0`



Weitere Informationen finden Sie im Kapitel **[Programmablaufkontrolle]** Abschnitt **[Wartezeiten]**.



**NOTIZEN:**

## 4.5 Überschleifbewegungen

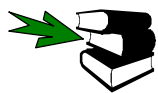
Überschleif-  
kontur

Zur Geschwindigkeitssteigerung können Punkte, die nicht genau angefahren werden müssen, überschleift werden. Der Roboter kürzt den Weg, wie in Abb. 28 gezeigt, ab.

Die Überschleifkontur wird 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.

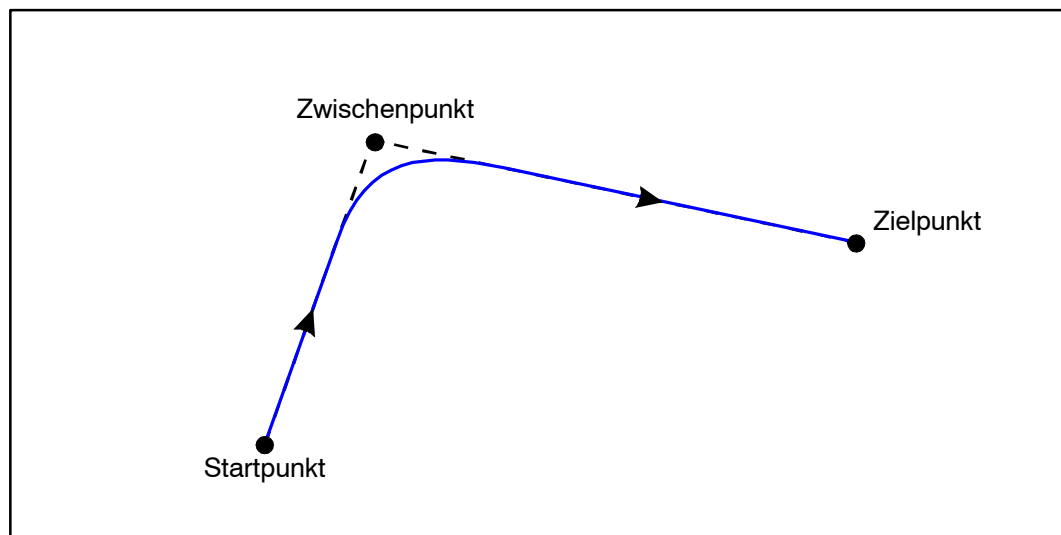
Um ein Überschleifen zu ermöglichen, muß daher der Rechnervorlauf (\$ADVANCE) mindestens auf 1 gesetzt sein. Falls der Rechnervorlauf zu klein ist, erscheint die Meldung "Überschleifen nicht möglich" und die Punkte werden genau angefahren.

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



Weitere Informationen zur TRIGGER-Anweisung als Abhilfe finden Sie im Kapitel **[Trigger- und Bahnbezogene Schaltaktionen]**.

Überschleifen



**Abb. 28** Überschleifen von Zwischenpunkten



## 4.5.1 PTP-PTP-Überschleifen

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

Überschleif-  
beginn

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 Überschleifen erfolgt genau dann, wenn die erste Achse einen Winkel von 45° vom Überschleifpunkt verfahren ist.

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 überschleifen 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 in Geschwindigkeit und Beschleunigung optimierte Bewegungsausführung.



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Höheres Fahrprofil]**.

Aus folgendem Beispiel können Sie die Wirkung der Überschleifanweisung und der Variablen \$APO.CPTP ersehen. Die abgefahrte Bahn ist in Abb. 29 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

```

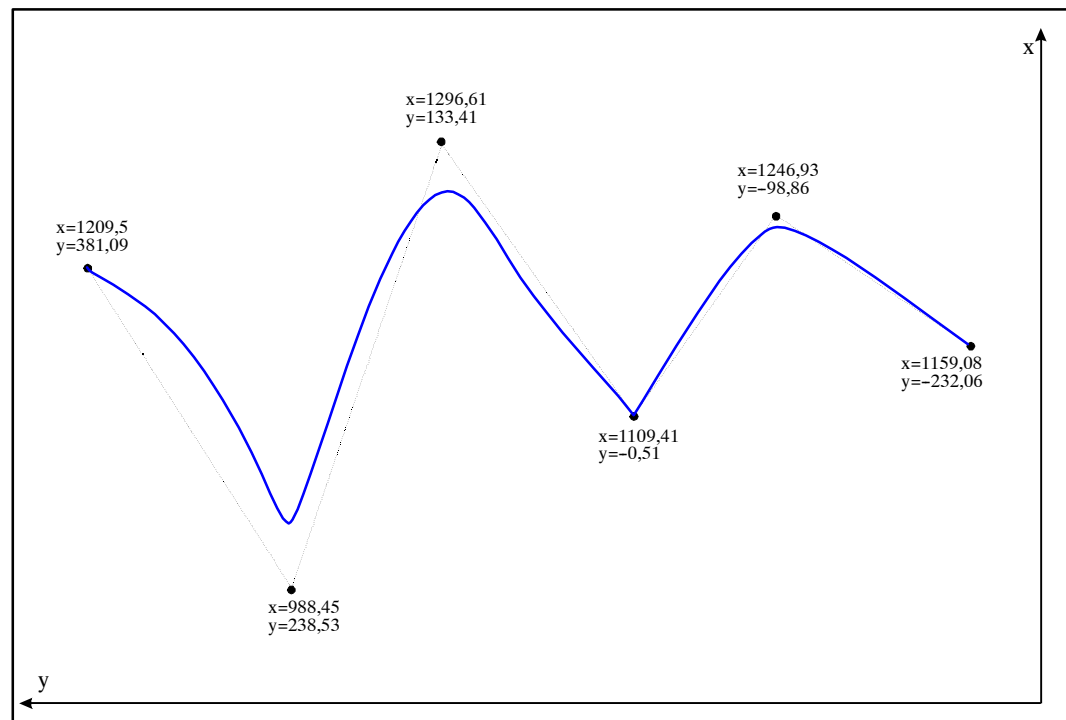


Abb. 29 Beispiel für PTP-PTP-Überschleifen



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. 29 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.

Überschleif-  
beginn

Zur Festlegung des Überschleifbeginns stehen drei vordefinierte Variablen bereit (s. Tab. 7):

Variable	Datentyp	Einheit	Bedeutung	Schlüsselwort im Befehl
<b>\$APO.CDIS</b>	REAL	mm	Translatorisches Distanzkriterium	<b>C_DIS</b>
<b>\$APO.CORI</b>	REAL	°	Orientierungsdistanz	<b>C_ORI</b>
<b>\$APO.CVEL</b>	INT	%	Geschwindigkeitskriterium	<b>C_VEL</b>

**Tab. 7** Systemvariablen zur Festlegung des Überschleifbeginns

Distanzkrite-  
rium

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

Orientierungs-  
kriterium

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

Geschwindig-  
keitskriterium

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 frühestens 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.

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

Systemintern kann unter Umständen das Überschleifen verkürzt werden (Satzmitte, Symmetriekriterium), jedoch nie vergrößert.

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. 30:



```

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

```

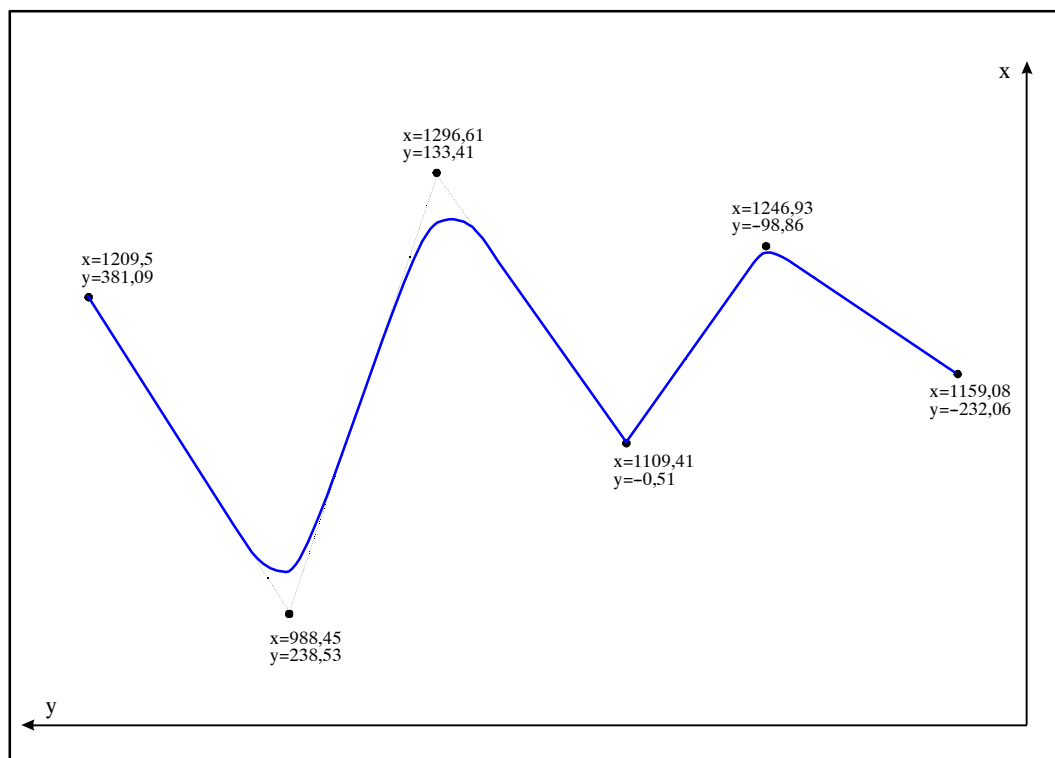


Abb. 30 Beispiel für LIN-LIN-Überschleifen



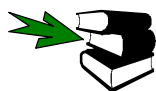
Die Position, bei der die Überschleifkontur in den nachfolgenden LIN-Satz mündet, wird von der Steuerung automatisch berechnet. Falls \$ACC und \$VEL für beide Einzelsätze identisch sind und die Satztlängen ausreichen, ist die Überschleifparabel symmetrisch zur Winkelhalbierenden zwischen den beiden Einzelsätzen. Bei kurzen Einzelsätzen wird der Überschleifbeginn auf die halbe Satztlänge begrenzt. 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.

### 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. LEERER MERKER).



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[LIN-LIN-Überschleifen]**.

Das CIRC-CIRC-Überschleifen soll wieder anhand eines Beispiels und der abgebildeten Bewegungsbahn (s. Abb. 31) 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

```

```

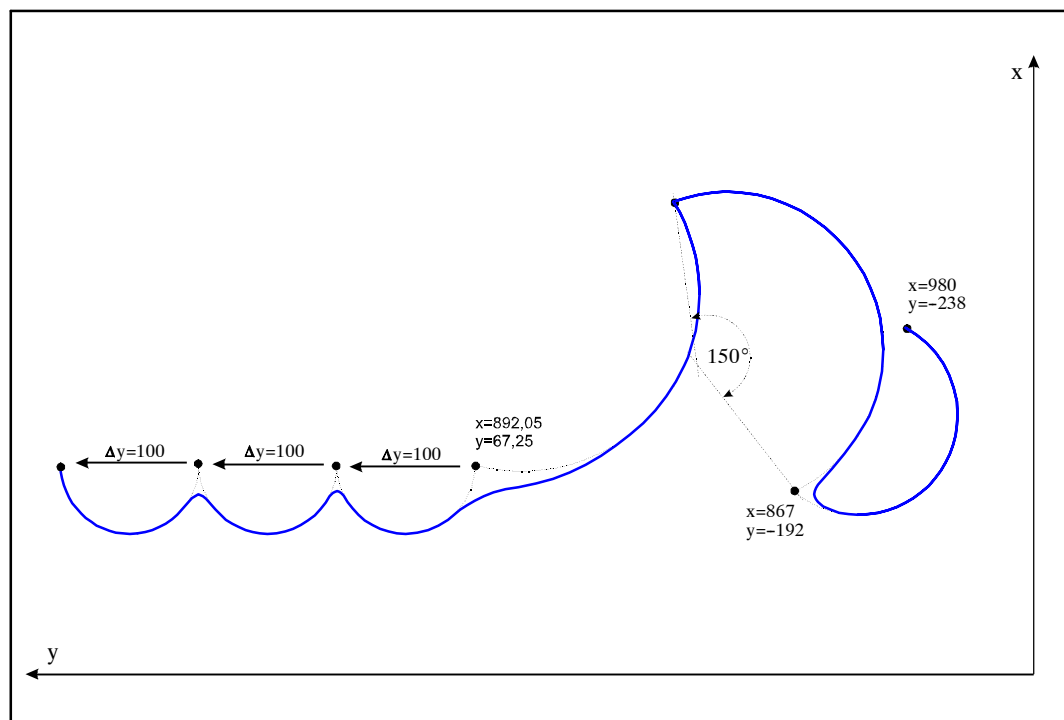
; relative Kreisbewegungen

; 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
    
```



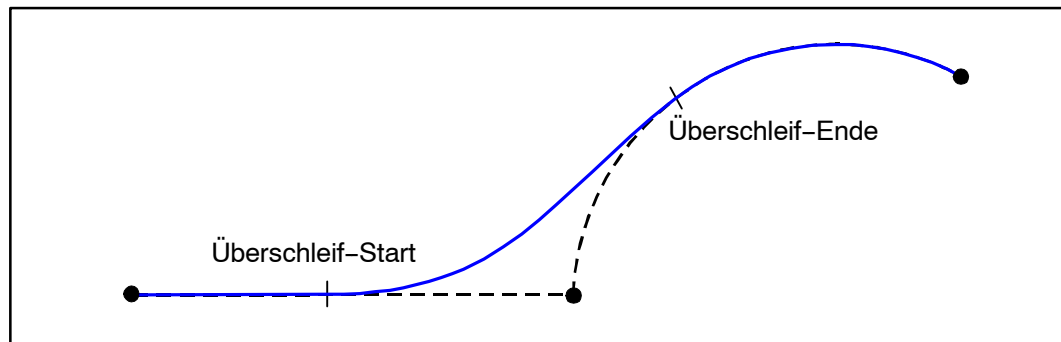
**Abb. 31** Beispiel für CIRC-CIRC-Überschleifen



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. 32).



LIN-CIRC  
Überschleif



**Abb. 32** 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 Umoorientierung im Überschleifbereich trotzdem raumbezogen.

#### 4.5.4 PTP-Bahnüberschleifen

Es besteht die Möglichkeit, achsspezifische PTP- und kartesische Bahn-Bewegungssätze zu überschleifen. Die PTP-Bewegung bietet folgende Vorteile:

- 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.



Die Bahn einer PTP-Bewegung kann nicht exakt vorherbestimmt werden.

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 Bahn-Bewegungssatz 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.



Die Überschleifkontur eines PTP-Bahn- bzw. Bahn-PTP-Überschleifs ist eine PTP-Bewegung. Eine genaue Bestimmung der Überschleifbahn ist daher nicht möglich.

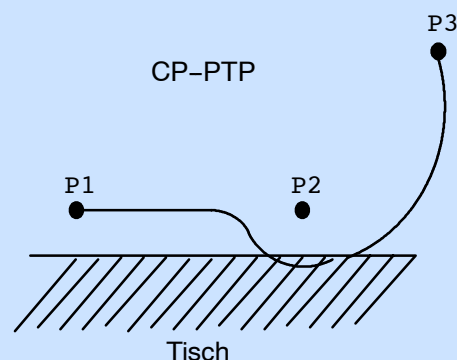
Bsp.:

Ist eine genaue Bahn notwendig, z.B. eine CP-Bewegung parallel zum Tisch, so ist beim Verlassen dieser Bahn mit der Kombination CP-PTP beim Überschleifen Vorsicht geboten. Gleiches gilt für eine PTP-CP-Bewegung um auf diese Bahn zu gelangen.

Programm:

```
...
LIN P1
LIN P2 C_DIS
PTP P3
...
```

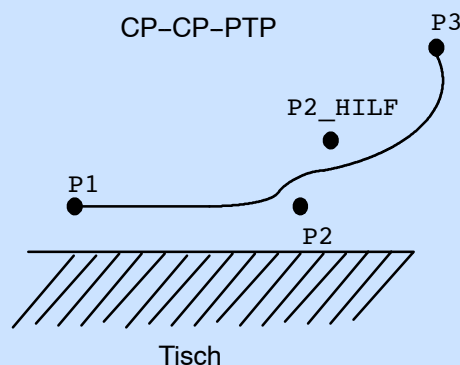
Hierbei entsteht das Problem, daß die Überschleifbewegung nicht voraussehbar ist. Eine Kollision mit dem Tisch wäre denkbar.



Um dem oben entstandenen Problem entgegenzutreten, und einen Genauhalt zu vermeiden, muß eine weitere CP-Bewegung (P2\_HILF) eingefügt werden.

Programm:

```
...
LIN P1
LIN P2 C_DIS
LIN P2_HILF C_DIS
PTP P3
...
```



### 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 Stellungenänderungen 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.



weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Bewegungsbefehle]**.

Im folgenden Beispiel wurde ein PTP-LIN-Überschleif, ein LIN-CIRC-Überschleif und ein CIRC-PTP-Überschleif programmiert (s. Abb. 33):



```

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
    
```

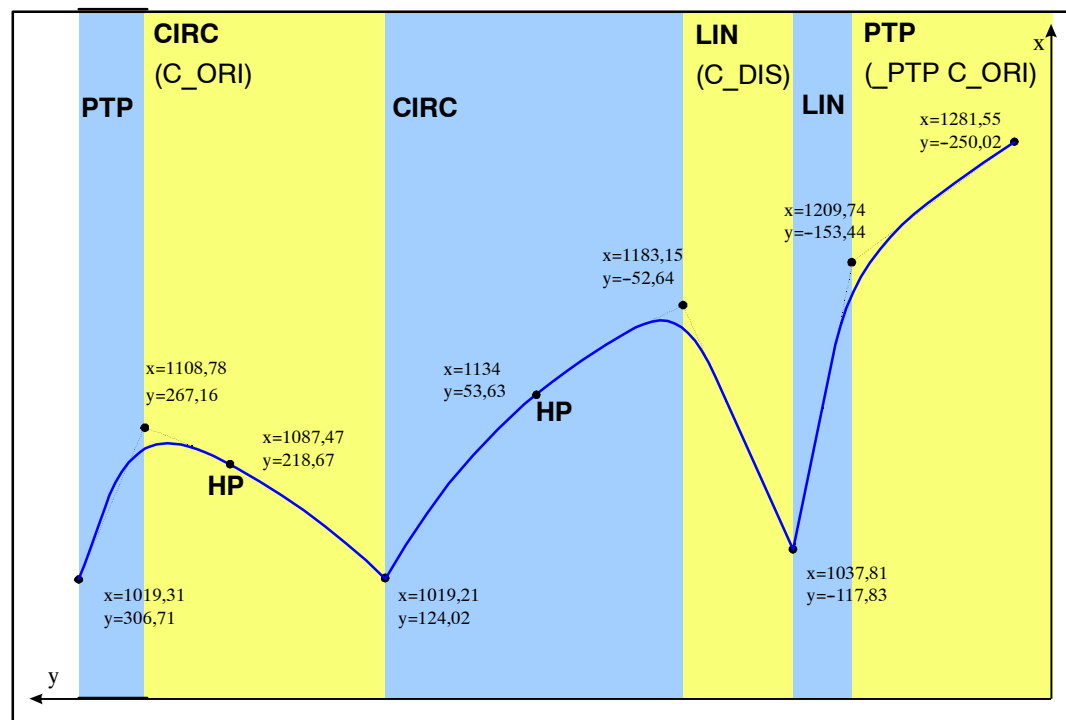


Abb. 33 PTP-Bahnüberschleifen und Bahn-Bahnüberschleifen

#### 4.5.5 Werkzeugwechsel beim Überschleifen

Diese Funktionalität steht für alle Kombinationen von PTP-, LIN- und CIRC-Einzelsätzen zur Verfügung.

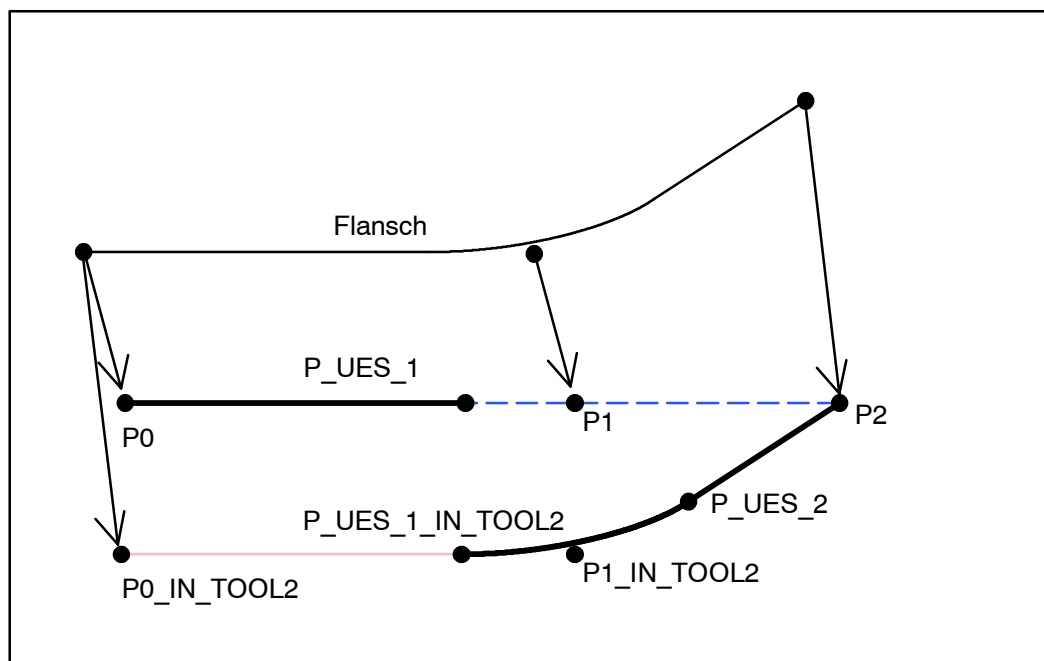
Es ist möglich ein Werkzeug auch während des Überschleifens virtuell zu wechseln. D.h. ein Werkzeug wird zweimal unterschiedlich gelernt. Z.B. ist bei einer Spritzpistole der Abstand zum Werkstück bei "TOOL1" 5cm und bei "TOOL2" 10cm.



##### Beispiel

In diesem Beispiel ist es erwünscht, daß am Anfang der Überschleifbewegung von P1 zu P2 das Werkzeug TOOL\_DATA[1] gegen TOOL\_DATA[2] ausgetauscht wird.

```
$TOOL=TOOL_DATA[1]
PTP P0_TOOL1;           Punkt wird mit dem Werkzeug TOOL_DATA[1] geteacht
LIN P1_TOOL1 C_DIS;     Punkt wird mit dem Werkzeug TOOL_DATA[1] geteacht
$TOOL=TOOL_DATA[2]
LIN P2_TOOL2;           Punkt wird mit dem Werkzeug TOOL_DATA[2] geteacht
```



Der Werkzeugwechsel findet in diesem Beispiel beim Überschleifen statt. D.h. die kartesische Position springt beim Eintritt in den Zwischensatz von P\_UES\_1 auf P\_UES\_1\_IN\_TOOL2; die Achswinkel sowie die kartesische Position und Geschwindigkeit des Flansches werden während der gesamten Bewegung stetig geführt. Der Sprung in der kartesischen TCP-Position wird erst während der Bewegung von P\_UES\_2 nach P2 abgebaut, so daß die Bahn nicht auf der räumlichen linearen Verbindung von P1 nach P2 verläuft, sondern auf einem Teil der mit Genauhalt programmierten Bahn.

## 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-File 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 entsprechende Fehlermeldung.



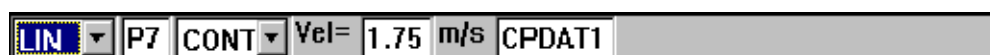
Weitere Informationen zum Thema “Datenlisten” finden Sie im Kapitel **[Datenlisten]**.



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!!