





**SOFTWARE**

**KR C1**

**Programmierung Experte**

**Release 3.2**

© Copyright **KUKA Roboter GmbH**

Diese Dokumentation darf – auch auszugsweise – nur mit ausdrücklicher Genehmigung des Herausgebers vervielfältigt oder Dritten zugänglich gemacht werden.

Es können weitere, in dieser Dokumentation nicht beschriebene Funktionen in der Steuerung lauffähig sein. Es besteht jedoch kein Anspruch auf diese Funktionen bei Neulieferung bzw. im Servicefall.

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so daß wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden jedoch regelmäßig überprüft, und notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Technische Änderungen ohne Beeinflussung der Funktion vorbehalten.

KUKA Interleaf

---

# Inhaltsverzeichnis

<b>1</b>	<b>Benutzergruppe Experte .....</b>	<b>7</b>
<b>2</b>	<b>Allgemeines zu KRL-Programmen .....</b>	<b>9</b>
2.1	Aufbau und Struktur von Programmen .....	9
2.1.1	Programmoberfläche .....	9
2.1.2	Dateikonzept .....	12
2.1.3	Dateistruktur .....	12
2.2	Programme erstellen und editieren .....	14
2.2.1	Erstellen eines neuen Programms .....	14
2.2.2	Programm editieren, kompilieren und binden .....	15
2.3	Ändern von Programmen .....	17
2.3.1	Programm-Korrektur .....	17
2.3.2	Editor .....	17
2.3.2.1	Blockfunktionen .....	17
2.3.2.2	Kopieren (CTRL-C) .....	17
2.3.2.3	Einfügen (CTRL-V) .....	18
2.3.2.4	Ausschneiden (CTRL-X) .....	18
2.3.2.5	Löschen .....	18
2.3.2.6	Suchen .....	19
2.3.2.7	Ersetzen .....	19
2.4	Verstecken von Programmteilen .....	22
2.4.1	FOLD .....	22
2.4.1.1	Beispielprogramm .....	23
2.4.2	Beschränkung der Informationsmenge (Detailansicht) .....	24
2.4.3	DEF-Zeile sichtbar/unsichtbar .....	26
2.5	Programmablaufarten .....	27
2.6	Fehlerbehandlung .....	29
2.7	Kommentare .....	32
<b>3</b>	<b>Variablen und Vereinbarungen .....</b>	<b>33</b>
3.1	Variablen und Namen .....	33
3.2	Datenobjekte .....	35
3.2.1	Vereinbarung und Initialisierung von Datenobjekten .....	35
3.2.2	Einfache Datentypen .....	37
3.2.3	Felder .....	39
3.2.4	Zeichenketten .....	42
3.2.5	Strukturen .....	42
3.2.6	Aufzählungstypen .....	44
3.3	Datenmanipulation .....	46
3.3.1	Operatoren .....	46
3.3.1.1	Arithmetische Operatoren .....	46
3.3.1.2	Geometrischer Operator .....	47
3.3.1.3	Vergleichsoperatoren .....	51
3.3.1.4	Logische Operatoren .....	52
3.3.1.5	Bit-Operatoren .....	53

3.3.1.6	Prioritäten von Operatoren .....	55
3.3.2	Standardfunktionen .....	56
3.4	Systemvariablen und Systemdateien .....	58
<b>4</b>	<b>Bewegungsprogrammierung .....</b>	<b>63</b>
4.1	Verwendung verschiedener Koordinatensysteme .....	63
4.2	Punkt-zu-Punkt Bewegungen (PTP) .....	70
4.2.1	Allgemein (Synchron-PTP) .....	70
4.2.2	Höheres Fahrprofil .....	71
4.2.3	Bewegungsbefehle .....	72
4.3	Bahnbewegungen (CP-Bewegungen = Continuous Path) .....	82
4.3.1	Geschwindigkeit und Beschleunigung .....	82
4.3.2	Orientierungsführung .....	83
4.3.3	Linearbewegungen .....	88
4.3.4	Kreisbewegungen .....	89
4.4	Rechnervorlauf .....	91
4.5	Überschleifbewegungen .....	94
4.5.1	PTP-PTP-Überschleifen .....	95
4.5.2	LIN-LIN-Überschleifen .....	98
4.5.3	CIRC-CIRC-Überschleifen und CIRC-LIN-Überschleifen .....	101
4.5.4	PTP-Bahnüberschleifen .....	104
4.5.5	Werkzeugwechsel beim Überschleifen .....	107
4.6	Teachen von Punkten .....	108
<b>5</b>	<b>KRL-Assistent .....</b>	<b>109</b>
5.1	Positionsangaben .....	110
5.2	[PTP] Positionierung .....	113
5.3	[LIN] Geradlinige Bewegung .....	115
5.4	[CIRC] Kreisbahnbewegung .....	117
<b>6</b>	<b>Programmablaufkontrolle .....</b>	<b>119</b>
6.1	Programmverzweigungen .....	119
6.1.1	Sprunganweisung .....	119
6.1.2	Bedingte Verzweigung .....	120
6.1.3	Verteiler .....	121
6.2	Schleifen .....	122
6.2.1	Zählschleife .....	122
6.2.2	Abweisende Schleife .....	124
6.2.3	Nicht abweisende Schleife .....	125
6.2.4	Endlosschleife .....	127
6.2.5	Vorzeitige Beendigung von Schleifendurchläufen .....	127
6.3	Warteanweisungen .....	128
6.3.1	Warten auf ein Ereignis .....	128
6.3.2	Wartezeiten .....	128
6.4	Anhalten des Programms .....	129
6.5	Quittieren von Meldungen .....	130
<b>7</b>	<b>Ein-/Ausgabeeinweisungen .....</b>	<b>131</b>
7.1	Allgemeines .....	131

7.2	Binäre Ein-/Ausgänge .....	132
7.3	Digitale Ein-/Ausgänge .....	135
7.4	Impulsausgänge .....	137
7.5	Analoge Ein-/Ausgänge .....	139
7.5.1	Analoge Ausgänge .....	139
7.5.2	Analoge Eingänge .....	141
7.6	Vordefinierte Digitaleingänge .....	143
<b>8</b>	<b>Unterprogramme und Funktionen .....</b>	<b>145</b>
8.1	Vereinbarung .....	145
8.2	Aufruf und Parameterübergabe .....	148
<b>9</b>	<b>Interrupt-Behandlung .....</b>	<b>153</b>
9.1	Deklaration .....	154
9.2	Aktivieren von Interrupts .....	156
9.3	Laufende Bewegungen anhalten .....	160
9.4	Abbrechen von Interrupt-Routinen .....	161
9.5	Verwendung zyklischer Flags .....	164
<b>10</b>	<b>Trigger – Bahnbezogene Schaltaktionen .....</b>	<b>165</b>
10.1	Schaltaktion am Start- oder Zielpunkt der Bahn .....	165
10.2	Schaltaktion beliebig auf der Bahn .....	169
<b>11</b>	<b>Datenlisten .....</b>	<b>175</b>
11.1	Lokale Datenlisten .....	175
11.2	Globale Datenlisten .....	176
<b>12</b>	<b>Externer Editor .....</b>	<b>179</b>
12.1	Starten des externen Editors .....	180
12.2	Menü "Datei" .....	183
12.2.1	Öffnen .....	183
12.2.2	Speichern .....	183
12.2.3	Datei schließen .....	183
12.2.4	Beenden .....	183
12.3	Menü "Bearbeiten" .....	185
12.3.1	Ausschneiden .....	185
12.3.2	Kopieren .....	185
12.3.3	Einfügen als ... ..	185
12.3.3.1	ursprüngliche Bahn .....	185
12.3.3.2	Rückwärtsbahn .....	185
12.3.4	Löschen .....	186
12.3.5	Alles markieren .....	186
12.4	Menü "Extras" .....	187
12.4.1	Achsspiegeln .....	187
12.4.2	Manuelles Verschieben .....	188

12.4.3	Verschieben .....	189
12.4.4	Blockweises Ändern .....	190
12.4.5	Datenliste reinigen .....	190
12.4.6	TCP-, bzw. BASE-Anpassung .....	191
12.4.7	Setzen der Softwareendschalter .....	192
12.5	Menü "HotEdit" .....	193
12.5.1	Limits .....	193
12.5.2	Tool, Base und World .....	193
12.6	Menü "Hilfe" .....	195
12.6.1	Version .....	195
12.6.2	Immer im Vordergrund .....	195



# 1 Benutzergruppe Experte

Die Software der KR C1 Steuerung unterscheidet standardmäßig zwischen dem Anwender und dem Experten. Der **Anwender** benötigt keine Programmiersyntaxkenntnisse, da er menügeführt Programme erstellt. Bei Neustart des Systems ist standardmäßig automatisch die Anwenderebene angewählt.

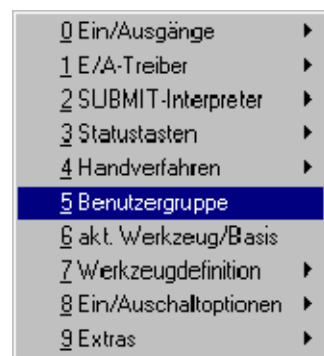
Reichen die Funktionen der Anwenderebene nicht aus, kann auf die Expertenebene gewechselt werden. Der **Experte** kann dann über die ASCII-Tastatur in der Roboterprogrammiersprache KRL (KUKA Robot Language) programmieren sowie System-, bzw. Initialisierungsdateien (Bussysteme) editieren.

KRL ist eine Hochsprache, die PASCAL-ähnlich und somit auch für die Programmierung von komplexen Aufgaben geeignet ist.

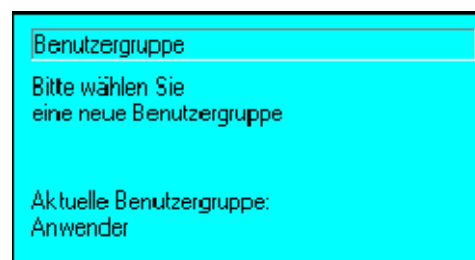
Paßwort

Der Zugang zur Expertenebene ist durch eine Paßwort geschützt. Zum Wechseln in die Expertenebene öffnen Sie über den Menükey "Konfigurier." ein Auswahlmennü, in dem der Menüpunkt "Benutzergruppe" enthalten ist.

Konfig.



Wenn Sie den Menüpunkt "Benutzergruppe" anwählen, öffnet sich eine Dialogbox, um sich als Experte oder Anwender anmelden zu können.

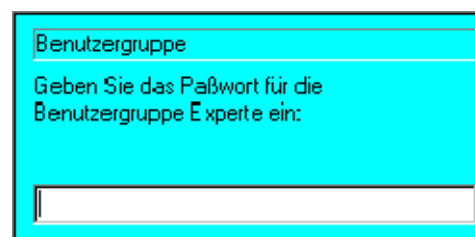


Anwender Experte

Softkeys zur Auswahl der Benutzergruppe

Befinden Sie sich auf Anwenderebene, so werden Sie nach Drücken des Softkeys "Experte" aufgefordert, das Paßwort für die Expertenebene einzugeben.

Experte



Weiter

Nach Eingabe des Paßwortes (Kleinbuchstaben, Caps-Log inaktiv) und Bestätigung mit der Return-Taste bzw. Drücken des Softkeys "Weiter", befinden Sie sich in der Expertenebene.

Anwender

Zur Rückkehr in den Anwendermodus drücken Sie den Softkey "Anwender" in der Dialogbox Benutzergruppe.

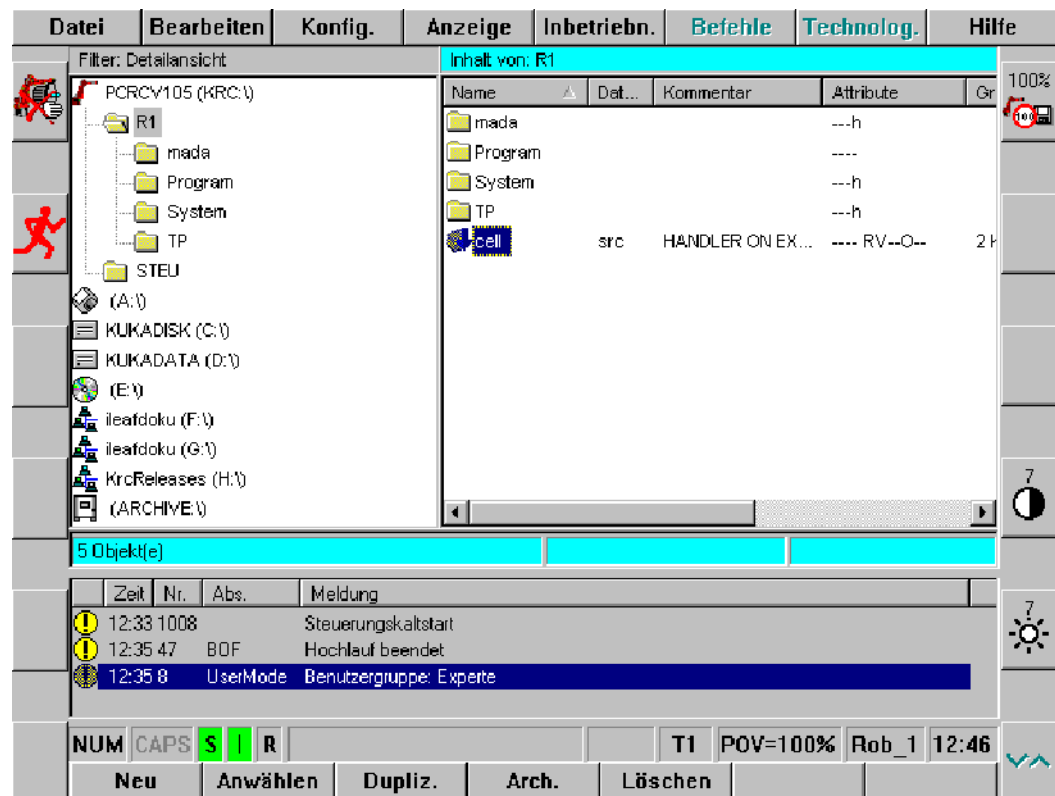


## 2 Allgemeines zu KRL-Programmen

### 2.1 Aufbau und Struktur von Programmen

#### 2.1.1 Programmoberfläche

Sobald Sie auf die Expertenebene gewechselt haben, verändert sich die Bedienoberfläche wie nachfolgend dargestellt:



Während für den Anwender alle Systemdateien unsichtbar sind, kann der Experte diese im Programmfenster sehen und auch editieren. Ferner werden neben den Dateinamen und Kommentaren auf Expertenebene auch die Dateiextension, -attribute und -größen angezeigt.

In vorstehender Abbildung werden im Programmfenster die Dateien und Verzeichnisse des Pfades "R1" angezeigt.

Standardmäßig sind nach der Installation der KRC1-Software im Verzeichnis "KRC:\R1\MADA\" die nachfolgend aufgeführten Dateien vorhanden:

Datei	Bedeutung
\$MASCHINE.DAT	Systemdatenliste mit Systemvariablen zur Anpassung von Steuerung und Roboter
\$ROBCOR.DAT	Systemdatenliste mit Daten für das Dynamikmodell des Roboters
MACHINE.UPG	Systemfile für zukünftige Upgrades
ROBCOR.UPG	Systemfile für zukünftige Upgrades

Im Verzeichnis "KRC:\R1\SYSTEM\" befinden sich folgende Dateien:

Datei	Bedeutung
\$CONFIG.DAT	Systemdatenliste mit allgemeinen Konfigurationsdaten
BAS.SRC	Basis-Paket für die Bewegungssteuerung
IR_STOPM.SRC	Programm zur Fehlerbehandlung beim Auftreten von Störungen
SPS.SUB	Submitfile für die parallele Überwachung

Im Verzeichnis "KRC:\R1\TP\" befinden sich folgende Dateien:

Datei	Bedeutung
A10.DAT A10.SRC	Technologiepaket zum Bahnschweißen mit analogen Leitspannungen
A10_INI.DAT A10_INI.SRC	Technologiepaket zur Initialisierung des Bahnschweißens mit analogen Leitspannungen
A20.DAT A20.SRC	Technologiepaket zum Bahnschweißen mit digitalen Programmnummern
A50.DAT A50.SRC	Technologiepaket für die Verwendung des Libo (Lichtbogen) – Sensors
ARC_MSG.SRC	Programm zur Programmierung von Meldungen für das Schutzgasschweißen.
ARCSPS.SUB	Submitfile für Bahnschweißen
BOSCH.SRC	Programm für Punktschweißen mit serieller Schnittstelle zum Bosch Punktschweißtimer PSS5200.521C
COR_T1.SRC	Programm zur Werkzeugkorrektur (alte Version)
CORRTOOL.DAT CORRTOOL.SRC	Programm zur Werkzeugkorrektur
FLT_SERV.DAT FLT_SERV.SRC	Programm zur benutzerdefinierten Fehlerbehandlung beim Bahnschweißen
H50.SRC	Greifer-Paket
H70.SRC	Touchsensor-Paket
MSG_DEMO.SRC	Programm mit Beispielen für Benutzermeldungen
NEW_SERV.SRC	Programm zur Änderung von Fehlerreaktionen für FLT_SERV
P00.DAT P00.SRC	Programmpaket zur Kopplung an eine SPS

PERCEPT.SRC	Programm zum Aufruf des PERCEPTRON-Protokolls
USER_GRP.DAT USER_GRP.SRC	Programm zur benutzerdefinierten Greiferansteuerung
USERSPOT.DAT USERSPOT.SRC	Programmpaket zum benutzerdefinierten Punktschweißen
WEAV_DEF.SRC	Programm für Pendelbewegungen beim Schutzgasschweißen

Im Verzeichnis "KRC:\R1\" befindet sich folgende Datei:

CELL.SRC	Programm zur Steuerung von Robotern über eine zentrale SPS. Hierbei wird entsprechend einer Programmnummer ein Bauteileprogramm ausgewählt
----------	---

## 2.1.2 Dateikonzept

Ein KRL-Programm kann aus SRC- und DAT-File bestehen.



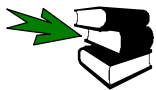
Weitere Informationen zur Programmerstellung finden Sie in diesem Kapitel, Abschnitt **[Programme erstellen und editieren]**.

Das "SRC"-File ist hierbei die Datei mit dem eigentlichen Programmcode. Dabei gibt es die Varianten DEF und DEFFCT (mit Rückgabewert). Das "DAT"-File enthält dagegen die spezifischen Programmdaten. Diese Teilung beruht auf dem Dateikonzept von KRL: Das Programm beinhaltet neben dem Bearbeitungsablauf verschiedene Aktionen, die der Industrieroboter ausführen soll. Dies können bestimmte Bewegungsabläufe sein, das Öffnen oder Schließen eines Greiferwerkzeugs, bis hin zu komplexen Abläufen, wie beispielsweise die Steuerung einer Schweißzange unter Berücksichtigung der Randbedingungen.

Zum Testen von Programmen ist es hilfreich bzw. erforderlich, Teilaufgaben einzeln zum Ablauf bringen zu können. Das in KRL realisierte Dateikonzept wird den speziellen Bedürfnissen der Roboterprogrammierung gerecht.

## 2.1.3 Dateistruktur

Eine Datei ist die Einheit, welche der Programmierer erstellt und entspricht damit einer Datei auf der Festplatte oder im Speicher (RAM). Jedes Programm in KRL kann aus einem oder mehreren Dateien bestehen. Einfache Programme umfassen genau eine Datei. Komplexere Aufgaben löst man besser mit einem Programm, das aus mehreren Dateien besteht.



Detaillierte Informationen über Unterprogramme und Funktionen finden Sie im Kapitel **[Unterprogramme und Funktionen]**



Der innere Aufbau einer KRL-Datei besteht aus Vereinbarungsteil, Anweisungsteil sowie bis zu 255 lokalen Unterprogrammen und Funktionen.

**DEF** Der Objektname ohne Erweiterung ist zugleich der Name der Datei und wird deshalb in der Vereinbarung mit "DEF" angeführt. Der Name kann aus maximal 24 Zeichen bestehen und darf kein Schlüsselwort sein (siehe Kapitel **[Variablen und Vereinbarungen]**). Jede Datei beginnt mit der Vereinbarung "DEF" und endet mit "END".

```
DEF NAME (X1 : IN)
Vereinbarungen
Anweisungen
END
```

**Vereinbarung** Vereinbarungen werden bereits vor der Programmbearbeitung, d.h. während des Übersetzens, ausgewertet. Im Vereinbarungsteil darf daher keine Anweisung stehen. Die erste Anweisung ist zugleich der Beginn des Anweisungsteils.

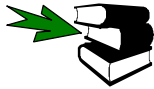
**Anweisung** Anweisungen haben im Gegensatz zu Vereinbarungen einen dynamischen Charakter: Sie werden bei der Programmbearbeitung ausgeführt.

**Datenliste** Ein Roboterprogramm kann aus einer Programmdatei alleine oder aus einer Programmdatei mit zugehöriger Datenliste bestehen. Datenliste und Datei werden über den gemeinsamen Namen als zusammengehörig gekennzeichnet. Die Namen unterscheiden sich nur in der Extension, z.B.:

Datei:       PROG1.SRC  
Datenliste:  PROG1.DAT



**In Datenlisten sind nur Wertzuweisungen mit “=” zulässig.** Wenn die Datenliste und die Datei den gleichen Namen besitzen, können Variablen, die in der Datenliste vereinbart sind, auf die gleiche Weise verwendet werden wie Variablen, die in der SRC-Datei vereinbart sind.



Detaillierte Informationen zum Thema finden Sie im Kapitel **[Datenlisten]**.



---

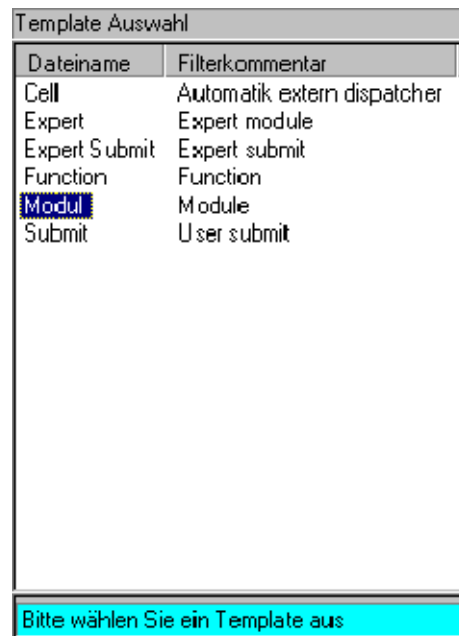
### NOTIZEN:

## 2.2 Programme erstellen und editieren

### 2.2.1 Erstellen eines neuen Programms

Neu

Da ein Roboterprogramm auch ohne Datenliste geschrieben werden kann, werden Datei und Datenliste auf Expertenebene nicht automatisch gleichzeitig angelegt. Um ein Programm anzulegen, betätigen Sie den Softkey "Neu". Es erscheint folgendes Fenster:

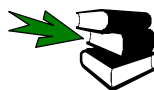


OK



Sie werden aufgefordert ein Template auszuwählen. Verwenden Sie dazu die Pfeiltasten und bestätigen Sie mit dem Softkey "OK" oder mit der Return Taste.

Die verfügbaren Templates können nicht in allen Verzeichnissen angelegt werden.



Weitere Informationen über Templates finden Sie im **Bedienhandbuch** in der Dokumentation **Bedienung**, Kapitel **[Navigator]**, Abschnitt **[Anhang]**.

Die verschiedenen Templates im einzelnen:

#### **Modul:**

Es wird ein SRC- und DAT-File angelegt, das ein Rumpfprogramm enthält.

#### **Expert:**

Es wird ein SRC- und DAT-File angelegt, das lediglich die Kopfzeile DEF... und END enthält.

#### **Cell:**

Hierbei wird nur ein SRC-File angelegt, das ein Rumpfprogramm enthält. Dieses Programm dient zur Steuerung des Roboters über eine zentrale SPS.

#### **Function:**

Hier wird eine Funktion (SRC-File) angelegt, die die Kopfzeile DEF... und END enthält.

#### **Submit:**

Es wird ein SUB-File mit Rumpfprogramm angelegt. Die Submitdatei enthält Anweisungen und kann z.B. zur zyklischen Überwachung (Greifer, etc.) genutzt werden. Die Submitdatei arbeitet parallel zum Roboterbetrieb und wird vom Steuerungsinterpret abgearbeitet.

#### **Expert Submit:**

Wie beim Template Submit wird ein SUB-File angelegt, das jedoch nur die Kopfzeile DEF... und END enthält.



Die Kopfzeile DEF... und END und die Rumpfprogramme der einzelnen Templates sind z.B. für das Template Cell unter "C:\KRC\ROBOTER\TEMPLATE\CellVorgabe.src" zu finden.



Wenn Sie das entsprechende Template ausgewählt haben, werden Sie aufgefordert einen Namen für das erzeugte File einzugeben.

Name	D...	Kommentar	Attribute	Grö...
<div style="border: 1px solid black; width: 100%; height: 100%;"></div>	---		---	---

Dateiname  
(max. 8 Zeichen)

Dateiextension  
(SRC, DAT oder SUB)

Kommentar

Der Dateiname ist als einziges zwingend erforderlich und darf max. 24 Zeichen betragen. Die Dateiextension wird automatisch ergänzt. Wenn Sie einen Kommentar einfügen wollen, müssen Sie mit der Pfeiltaste rechts den Cursor auf das entsprechende Feld bewegen, und den gewünschten Text eingeben.

OK

Drücken Sie den Softkey "OK" oder die Returnntaste um diese Eingaben zu quittieren.



Die Datenliste ist zwingend erforderlich, wenn Sie in ihrer SRC-Datei auch menügeführte Befehle einfügen wollen.

### 2.2.2 Programm editieren, kompilieren und binden

Haben Sie sich eine Datei oder eine Datenliste mit "Neu" erstellt, so können Sie sie mit dem Editor bearbeiten. Hierzu dient der Softkey "Öffnen". Beim Schließen des Editors wird der komplette Programmcode kompiliert, d.h. der textuelle KRL-Code wird in eine für die Steuerung verständliche Maschinensprache übersetzt.



Damit das Programm übersichtlich bleibt, ist es nötig z.B. bei Verzweigungen diese auf mehreren Ebenen einzurücken. Im Editor kann das durch Leerzeichen geschehen.

Compiler

Der Compiler überprüft den Code dabei auf syntaktische und semantische Richtigkeit. Falls Fehler vorhanden sind, wird eine entsprechende Meldung ausgegeben und eine Fehlerdatei mit der Dateierweiterung ".ERR" erzeugt.



Nur fehlerfreie Programme können angewählt und ausgeführt werden.



Weitere Informationen zur Behandlung von Editier-Fehlern finden Sie im Abschnitt **[Fehlerbehandlung]**.

Binder

Beim Laden eines Programms über den Softkey "Anwählen" werden alle benötigten Dateien und Datenlisten zu einem Programm zusammengebunden. Beim Binden wird überprüft, ob alle Module vorhanden, analysiert und fehlerfrei sind. Außerdem überprüft der Binder bei einer Parameterübergabe die Typverträglichkeit zwischen den Übergabeparametern. Treten beim Binden Fehler auf, so wird – wie beim Kompilieren – ein Error-File mit der Erweiterung ".ERR" erzeugt.



Sie können ein KRL-Programm auch mit jedem üblichen Text-Editor schreiben und anschließend mit dem Softkey "Laden" in den Systemspeicher laden. In diesem Fall müssen Sie allerdings selbst darauf achten, daß alle notwendigen Initialisierungen (z.B. Achsgeschwindigkeiten) vorgenommen werden.

Nachfolgend ein einfaches Programmbeispiel zur Festlegung von Achsgeschwindigkeiten und Achsbeschleunigungen:



```

DEF PROG1()

;----- Vereinbarungsteil -----
INT J

;----- Anweisungsteil -----
$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 {A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

FOR J=1 TO 5
    PTP {A1 4}
    PTP {A2 -7,A3 5}
    PTP {A1 0,A2 -9,A3 9}
ENDFOR

PTP {A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
END
    
```



#### NOTIZEN:

## 2.3 Ändern von Programmen

Grundsätzlich gibt es zwei Möglichkeiten, ein Programm auf der Expertenebene der Bedienoberfläche zu ändern:

- Programm-Korrektur (PROKOR)
- Editor

### 2.3.1 Programm-Korrektur

Die Programm-Korrektur ist eine Standard-Methode. Wird ein Programm angewählt, oder ein laufendes Programm gestoppt, befindet man sich automatisch im PROKOR-Modus.

Hier kann man Befehle anhand des Inlineformulars bzw. ASCII-Code (in der Expertenebene) eingeben oder bearbeiten, die sich nur auf **eine** Programmzeile auswirken – also keine Kontrollstrukturen (Schleifen etc.) oder Variablendeklarationen.



Im angewählten Zustand werden fehlerhafte Eingaben werden nach verlassen der Programmzeile sofort gelöscht und es erscheint eine Fehlermeldung im Meldungsfenster.

### 2.3.2 Editor

Will man also bestimmte KRL-Befehle oder Programmstrukturen bearbeiten oder einfügen, so wird man dies über den Editor tun. Da bei Schließen des Editors der komplette Code kompiliert wird, können auch Fehler erkannt werden, die erst im Zusammenhang mehrerer Zeilen auftreten (z.B. falsch vereinbarte Variablen).

#### 2.3.2.1 Blockfunktionen



Die Blockfunktionen stehen nur im Editor ab der Benutzerebene "Experte" zur Verfügung. Sie müssen ein Programm, dessen Inhalt Sie mit Hilfe der Blockfunktionen ändern wollen, mit dem Softkey "Edit" öffnen. Wie Sie zuvor in die Benutzerebene "Experte" wechseln, wird im Kapitel **[System konfigurieren]**, Abschnitt **(Benutzergruppe)** beschrieben.

Setzen Sie zunächst den blinkenden Editiercursor an den Anfang oder das Ende des zu verschiebenden Programmteils. Halten Sie dann die "Shift"-Taste auf der Tastatur gedrückt, während Sie den Cursor nach unten, bzw. oben bewegen. Auf diese Weise markieren Sie einen Programmteil, der im nächsten Arbeitsschritt mit den Blockfunktionen bearbeitet werden soll. Den bereits markierten Teil erkennen Sie an der farblichen Hervorhebung.

**Bearbeiten**

Betätigen Sie den Menükey "Bearbeiten" und wählen Sie aus dem sich öffnenden Menü die gewünschte Funktion aus.



Werden für die Blockfunktionen der Nummernblock und das Tastaturfeld verwendet, muß die NUM-Funktion ausgeschaltet sein. Drücken Sie in diesem Fall die "Num"-Taste auf dem Tastaturfeld. Die entsprechende Anzeige in der Statuszeile ist dann ausgeschaltet.

#### 2.3.2.2 Kopieren (CTRL-C)

Der markierte Programmteil wird zur weiteren Verarbeitung zwischengespeichert. Er kann anschließend an anderer Stelle eingefügt werden.

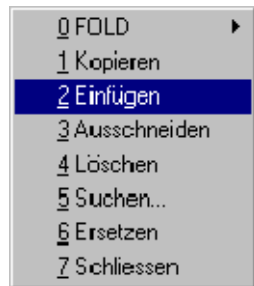




Alternativ können Sie die CTRL-Taste auf dem Nummernblock gedrückt halten und auf der Tastatur die C-Taste drücken. Lassen Sie anschließend beide Tasten los.

### 2.3.2.3 Einfügen (CTRL-V)

Setzen Sie den Editiercursor an die Stelle, an welcher der zuvor "herausgeschnittene" oder "kopierte" Programmteil wieder eingefügt werden soll.



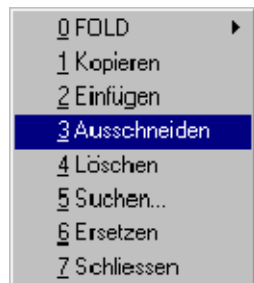
Wählen Sie jetzt die Option "Block einfügen" aus. Der vorher markierte Programmteil wird unterhalb des Editier-Cursors wieder eingefügt.



Alternativ können Sie die CTRL-Taste auf dem Nummernblock gedrückt halten und auf der Tastatur die V-Taste drücken. Lassen Sie anschließend beide Tasten los.

### 2.3.2.4 Ausschneiden (CTRL-X)

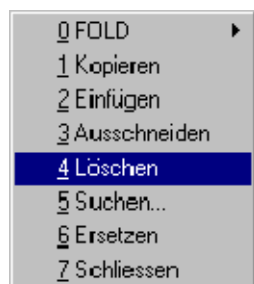
Wählen Sie aus dem Menü die Option "Block ausschneiden" aus, wird der markierte Programmteil zwischengespeichert und aus dem Programmlisting gelöscht.



Alternativ können Sie die CTRL-Taste auf dem Nummernblock gedrückt halten und auf der Tastatur die X-Taste drücken. Lassen Sie anschließend beide Tasten los.

### 2.3.2.5 Löschen

Der markierte Bereich kann aus dem Programm entfernt werden. In diesem Fall erfolgt keine Zwischenspeicherung. Der entfernte Programmteil ist damit unwiderruflich verloren.



Aus diesem Grund erfolgt eine Sicherheitsabfrage im Meldungsfenster, die über die Softkey-leiste beantwortet werden muß.



- **Abbrechen** Die Aktion "Löschen" wird abgebrochen.
- **Ja** Der markierte Bereich wird unwiderruflich gelöscht;
- **Nein** Die Funktion "Löschen" wird abgebrochen;



Wählen Sie aus dem Menü die Option "Löschen" aus, wird der markierte Programmteil ohne Zwischenspeicherung aus dem Programmlisting gelöscht.

### 2.3.2.6 Suchen

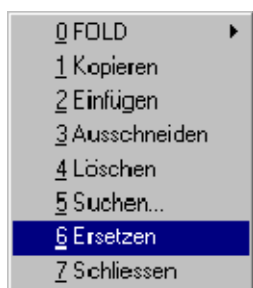


Weitere Informationen finden Sie im **Bedienhandbuch** in der Dokumentation **Prog. Anwender**, Kapitel **[Programmbearbeitung]**, Abschnitt **[Arbeiten mit dem Programmierer]**.

### 2.3.2.7 Ersetzen

Die Funktion "Suchen und ersetzen" steht nur in der Expertenebene und dort ausschließlich im Editor zur Verfügung. Diese Anwendung durchsucht das Programm nach einer vorgegebenen Zeichenfolge im sichtbaren Bereich (nicht Fold-Zeilen oder geöffnete Folds) und ermöglicht die Ersetzung mit einer definierten Zeichenfolge.

Wählen Sie dazu im Menü "Bearbeiten" die Option "Ersetzen".



Es öffnet sich folgendes Fenster:

Suchen	<input type="text"/>
Ersetzen	<input type="text"/>

```

1  DEF ERROR( )
2  int i
3  INI
4  PTP HOME  Ue1= 100 % DEFAULT
5
6  FOR i=1 TO 4
7  $OUT[I]=TRUE
8  ENDFOR
9
10 I=I+1
11
12 PTP HOME  Ue1= 100 % DEFAULT
13 END
  
```

KRC:\R1\PROGRAM\ERROR.SRC      Ln 6, Col 10

Die Softkeyleiste ändert sich.

Suchen	ersetzen	Alle ers.			Abbrechen
--------	----------	-----------	--	--	-----------

Geben Sie eine Zeichenfolge in der Suchzeile ein und wechseln mit der Pfeiltaste nach unten in die Ersetzenzeile. Dort tragen Sie den Begriff ein, der den Gesuchten ersetzen soll.

Suchen	FOR
Ersetzen	IF

```

1  DEF ERROR( )
2  int i
3  INI
4  PTP HOME  Ue1= 100 % DEFAULT
5
6  FOR i=1 TO 4
7  $OUT[I]=TRUE
8  ENDFOR
9
10 I=I+1
11
12 PTP HOME  Ue1= 100 % DEFAULT
13 END
  
```

KRC:\R1\PROGRAM\ERROR.SRC      Ln 1, Col 0

Suchen

ersetzen

Tritt der gesuchte Begriff öfter in dem Dokument auf und Sie wollen ihn nur an einer best. Stelle ersetzen, betätigen Sie den Softkey "Suchen" so oft, bis Sie die gewünschte Stelle gefunden haben. Anschließend drücken Sie "ersetzen". Der gesuchte Begriff wird gegen den Angegebenen ausgetauscht.

Möchten Sie den gesuchten Begriff an allen Stellen bzw. in einem vorher markierten Bereich im Programm ersetzen, so betätigen Sie nach der obigen Eingabe im Such-/Ersetzenformular den Softkey "Alle ers."

**Abbrechen**

Sie erhalten im Meldungsfenster die Nachricht, "Der angegebene bzw. markierte Bereich wurde durchsucht" (Bestätigung, daß das gesamte Programm bzw. die Markierung durchsucht wurde). Nach betätigen des Softkeys "Abbrechen" verlassen Sie den Ersetzenmodus und erhalten im Meldungsfenster die Anzahl der gemachten Ersetzungen seit Aktivierung dieser Funktion.

	Zeit	Nr.	Abs.	Meldung
!	12:10 40	BOF		Der angegebene bzw. markierte Bereich wurde durchsucht.
!	12:13 39	BOF		2 Ersetzungen gemacht



**NOTIZEN:**

## 2.4 Verstecken von Programmteilen

Anders als bei normalen Editoren gestattet der KCP-Editor eine anforderungsspezifische Anzeige der Programminhalte. So sieht zum Beispiel der Anwender nur die wesentlichen Inhalte eines Programms, während auf der Expertenebene das gesamte Programm einsehbar ist.

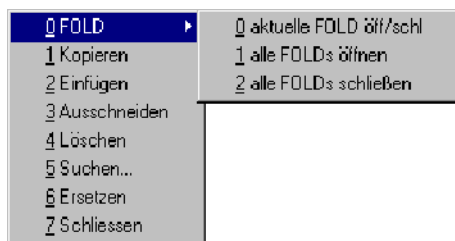
### 2.4.1 FOLD

Die KUKA-Bedienoberfläche benutzt eine besondere Technik zur übersichtlichen Darstellung eines Programmes. Als KRL-Kommentare markierte Anweisungen erlauben es, die Anzeige von nachfolgenden Teilen des Programmes zu unterdrücken. Das Programm wird so in sinnvolle Abschnitte unterteilt, die entsprechend ihrem Ordner-Charakter "FOLDS" genannt werden.

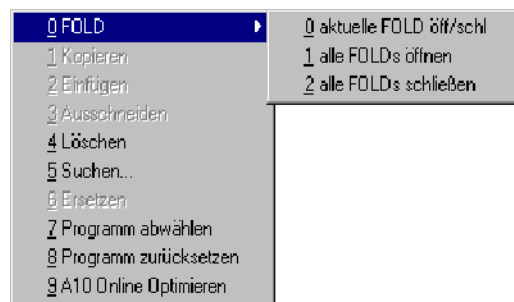
"FOLDS" sind standardmäßig "geschlossen" und können nur auf der Expertenebene "geöffnet" werden. Sie enthalten Informationen die für den Anwender auf der KUKA-Bedienoberfläche (KUKA-BOF) unsichtbar sind. Auf der Expertenebene haben Sie die Möglichkeit einen KRL-Block für den Anwender unsichtbar zu machen. Hierfür werden die betreffenden Vereinbarungen oder Anweisungen durch die Bezeichnungen "; FOLD" und "; ENDFOLD" eingeschlossen.

#### Bearbeiten

Folds in einem Programm können angezeigt oder versteckt werden, wenn Sie den Menükey "Bearbeiten" drücken und anschließend "FOLD" sowie das gewünschte Kommando anwählen.



Programm im Editor



Programm Angewählt

Folgende Optionen stehen zur Auswahl:

- **aktuelle FOLD öff/schl** öffnet bzw. schließt den FOLD der Zeile, in der sich der Editier-Cursor befindet
- **alle FOLDS öffnen** öffnet alle FOLDS des Programms
- **alle FOLDS schließen** schließt alle FOLDS des Programms



Wird ein angewähltes Programm, in dem Folds geöffnet sind, zurückgesetzt, so werden diese Folds automatisch geschlossen.

Von der Sequenz...

```
;FOLD RESET OUT

FOR I=1 TO 16
    $OUT[I]=FALSE
ENDFOR

;ENDFOLD
```



...sind bei geschlossenen Folds auf der Bedienoberfläche nur die Worte "**RESET OUT**" zu sehen. Mit diesem Befehl können Sie beispielsweise Deklarations- und Initialisierungsteil für den Anwender unsichtbar machen.

#### 2.4.1.1 Beispielprogramm



```
DEF FOLDS()

;FOLD DECLARATION;% weitere Informationen
;----- Deklarationsteil -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I
;ENDFOLD

;FOLD INITIALISATION
;----- Initialisierung -----
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
;Beschleunigungen, $BASE, $TOOL, etc.
FOR I=1 TO 16
  $OUT[I]=FALSE
ENDFOR
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
;ENDFOLD

;----- Hauptteil -----
PTP HOME ;SAK-Fahrt
LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
PTP HOME

END
```

Das Beispielprogramm sieht auf der Bedienoberfläche folgendermaßen aus:

```
1
2  DECLARATION
3
4  INITIALISATION
5
6  ;----- Hauptteil -----
```

Das gleiche Programm mit geöffneten Folds:

```

1
2  DECLARATION
3  ;----- Deklarationsteil -----
4  EXT BAS (BAS_COMMAND :IN,REAL :IN )
5  DECL AXIS HOME
6  INT I
7
8  INITIALISATION
9  ;----- Initialisierung -----
10 INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
11 INTERRUPT ON 3
12 BAS (#INITNOV,0 ) ;Initialisierung von Geschwindigkeiten,
13 ;Beschleunigungen, $BASE, $TOOL, etc.
14 FOR I=1 TO 16
15 $OUT[I]=FALSE
16 ENDFOR
17 HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
18
19 ;----- Hauptteil -----

```

Im geschlossenen FOLD ist nur der Ausdruck nach dem Schlüsselwort "FOLD" sichtbar. Im geöffneten FOLD sind dagegen alle Anweisungen und Vereinbarungen zu sehen.

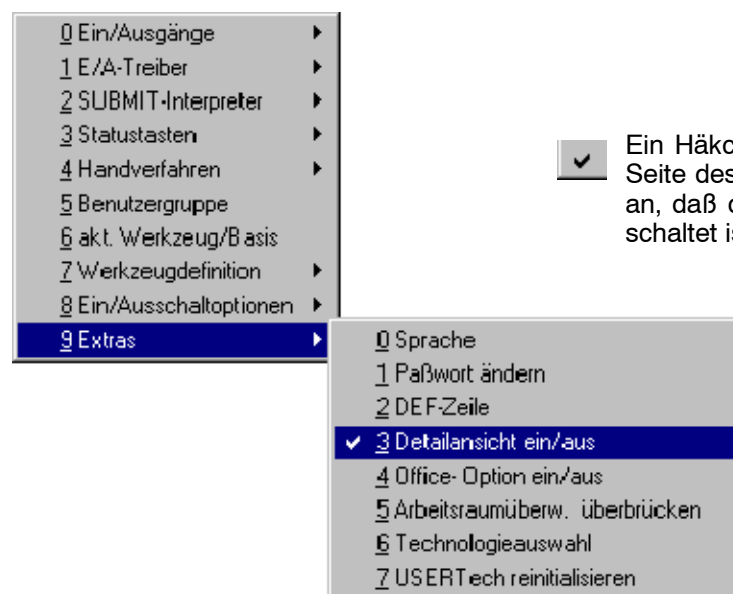


"FOLD" ist lediglich eine Anweisung für den Editor. Der Compiler interpretiert die FOLD-Anweisungen aufgrund des vorangestellten Semikolons als normalen Kommentar.

## 2.4.2 Beschränkung der Informationsmenge (Detailansicht)

**Konfig.**

Ein weiteres Hilfsmittel um die Informationsmenge auf der Bedienoberfläche möglichst gering zu halten, ist die Funktion "Detailansicht ein/aus". Durch Drücken des Menükeys "Konfig." und wählen der Option "Extras" -> "Detailansicht ein/aus" gelangen Sie zur gewünschten Funktion.



Ein Häkchen auf der linken Seite des Menüpunkts zeigt an, daß die Funktion eingeschaltet ist.

"Detailansicht ein/aus" ist standardmäßig aktiviert und kann nur auf der Expertenebene ausgeschaltet werden. "Detailansicht ein/aus" unterdrückt z.B. alle Texte in einer FOLD-Zeile, welche nach den Zeichen ";%" geschrieben werden. Diese Informationen werden aber zur Anzeige eines Inline-Formulars benötigt.

✓ 3 Detailansicht ein/aus

```

1
2  DECLARATION
3
4  INITIALISATION
5
6  ;----- Hauptteil -----
7  PTP HOME ;SAK-FAHRT
8  LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
9  PTP HOME
10

```

3 Detailansicht ein/aus

```

1  &ACCESS RU0
2  DEF FOLDS ( )
3
4  ;FOLD DECLARATION;% weitere Informationen
5
6  ;FOLD INITIALISATION
7
8  ;----- Hauptteil -----
9  PTP HOME ;SAK-FAHRT
10 LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
11 PTP HOME
12
13 END

```



Erst wenn alle FOLDS geöffnet sind **und** "Detailansicht ein/aus" ausgeschaltet ist, sind dem Programmierer alle vorhandenen Programmzeilen verfügbar. Die Darstellung auf der Bedienoberfläche entspricht dann der Darstellung in einem normalen Texteditor.

```

1  &ACCESS RU0
2  DEF FOLDS ( )
3
4  ;FOLD DECLARATION;% weitere Informationen
5  ;----- Deklarationsteil -----
6  EXT BAS (BAS_COMMAND :IN,REAL :IN )
7  DECL AXIS HOME
8  INT I
9  ;ENDFOLD
10
11 ;FOLD INITIALISATION
12 ;----- Initialisierung -----
13 INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
14 INTERRUPT ON 3
15 BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
16 ;Beschleunigungen, $BASE, $TOOL, etc.
17 FOR I=1 TO 16
18 $OUT[I]=FALSE
19 ENDFOR

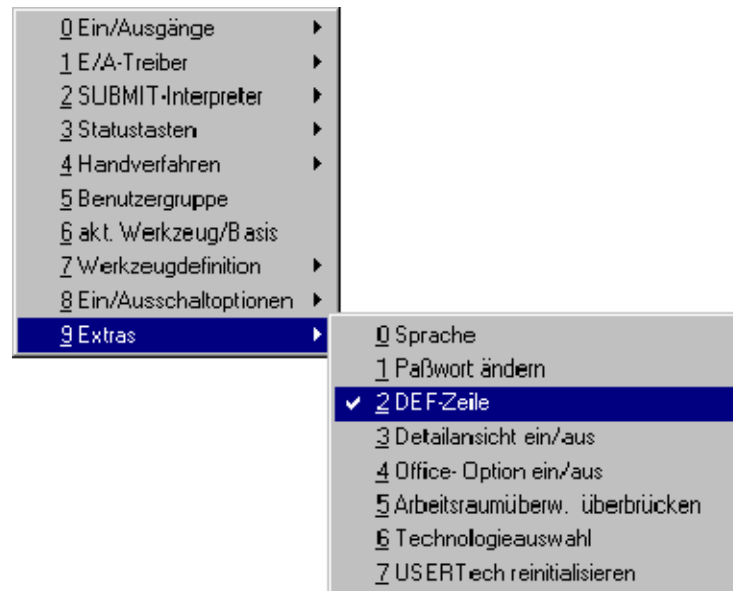
```

/R1/FOLDS.SRC Ln 3, Col 0

### 2.4.3 DEF-Zeile sichtbar/unsichtbar

**Konfig.**

Ebenfalls hilfreich ist die Funktion "DEF-Zeile". Diese Option blendet die "DEF"- und "END"-Zeilen ein. Drücken Sie hierzu den Menükey "Konfig." und wählen die Option "Extras" -> "DEF-Zeile" aus.



Mit dem Beispielprogramm aus dem Abschnitt (**Verstecken von Programmteilen**) sieht der Inhalt des Programmfensters folgendermaßen aus:

```

2 DEF-Zeile
1
2  DECLARATION
3
4  INITIALISATION
5
6  ;----- Hauptteil -----
7  PTP HOME ;SAK-FAHRT
8  LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
9  PTP HOME
10
    
```

```

✓ 2 DEF-Zeile
1  DEF FOLDS ( )
2
3  DECLARATION
4
5  INITIALISATION
6
7  ;----- Hauptteil -----
8  PTP HOME ;SAK-FAHRT
9  LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
10 PTP HOME
11
12 END
    
```






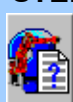

Erst wenn die DEF-Zeile sichtbar ist, können Deklarationen vorgenommen werden. Nach einem Neustart des Systems oder dem Wechsel auf die Benutzergruppe "Anwender" wird diese Funktion automatisch deaktiviert.

## 2.5 Programmablaufarten

Die Programmablaufart legt fest, ob die Programmbearbeitung

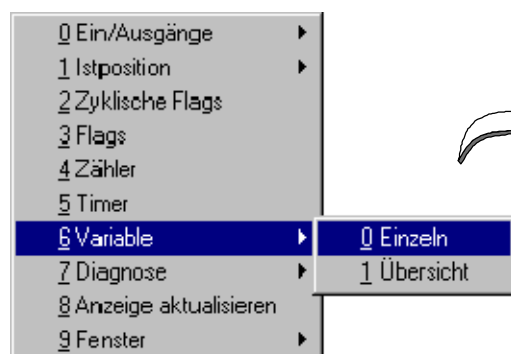
- ohne Programmstop,
- schrittweise oder
- satzweise

erfolgen soll. In nachfolgender Tabelle sind alle Programmablaufarten beschrieben.

Ablaufart	Beschreibung
<b>GO</b> 	Alle Anweisungen werden im Programm ohne STOP bis zum Programmende bearbeitet.
<b>MSTEP</b> 	Motion Step (Bewegungssatz) Das Programm wird schrittweise, d.h. mit einem STOP nach jedem Bewegungssatz ausgeführt. Das Programm wird ohne Vorlauf bearbeitet.
<b>ISTEP</b> 	Inkremental Step (Einzelsatz) Das Programm wird satzweise, d.h. mit einem STOP nach jeder Anweisung (auch Leerzeile) ausgeführt. Das Programm wird ohne Vorlauf bearbeitet.
<b>PSTEP</b> 	Program Step (Programmschritt) Unterprogramme werden komplett abgefahren. Das Programm wird ohne Vorlauf bearbeitet.
<b>CSTEP</b> 	Continuous Step (Bewegungssatz) Das Programm wird schrittweise, d.h. mit einem STOP nach jedem Bewegungssatz mit Genauhalt ausgeführt. Das Programm wird mit Vorlauf abgearbeitet, d.h. Punkte werden überschiffen.

Die Programmablaufarten GO, MSTEP und ISTEP können am KCP über einen Statuskey oder die Variable "\$PRO\_MODE" angewählt werden. PSTEP und CSTEP hingegen sind nur über die Variable "\$PRO\_MODE" einstellbar. Zum Ändern des jeweiligen Zustands dieser Variable rufen Sie die Menüfunktion "Anzeige" -> "Variable" -> "Einzeln" auf. Geben Sie anschließend im Eingabefeld "Name" die Variable "\$PRO\_MODE" und im Feld "Neuer Wert" den gewünschten Wert ein.

### Anzeige



Name:

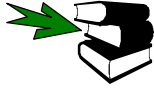
Aktueller Wert:

Neuer Wert:

Modul:



Die Programmablaufarten “#PSTEP” und “#CSTEP” können nur über die Variablenkorrektur und nicht per Statuskey ausgewählt werden.



Näheres finden Sie im Kapitel **[Variablen und Vereinbarungen]**, Abschnitt **[Datenobjekte]** unter **(Aufzählungstypen)**.



NOTIZEN:

## 2.6 Fehlerbehandlung



Tritt ein Fehler beim Kompilieren oder Binden auf, so wird im Meldungsfenster eine Fehlermeldung ausgegeben und die fehlerhafte Datei im Navigator kenntlich gemacht.



Als Beispiel dient die Datei "ERROR.SRC", die (fehlerhaft) erstellt wurde:

```

1  DEF  ERROR ( )
2  INI
3  PTP HOME  Ve1= 100 % DEFAULT
4
5  FOR I=1 TO 4
6  $OUT[I]=TRUE
7  ENDFOR
8
9  I == I + 1
10
11 PTP HOME  Ve1= 100 % DEFAULT
12 END

```

Nach Schließen des Editors erscheint nun im Meldungsfenster eine Hinweismeldung über die Anzahl der Fehler.

Zeit	Nr.	Abs.	Meldung
11:17:0			UserMode1 Benutzergruppe: Experte
11:35:1326	KCP	/R1/ERROR : 3 Fehler bei Analyse	

Gleichzeitig werden bei diesem Vorgang die betroffenen Dateien mit einem roten Kreuz gekennzeichnet.

Name	△	Datei-...	Kommentar	Attribu
error		src		-a-- R
error		dat		-a-- R

Ihnen steht die folgende Softkeyleiste zur Verfügung:

Neu	Fehlerliste	Öffnen	Datenliste	Löschen		
-----	-------------	--------	------------	---------	--	--

Der Softkey "Öffnen" lädt die Datei in den Editor und bei Betätigung des Softkeys "Datenliste" wird das Dat-File mit dem Editor geöffnet. Wollen Sie die fehlerhaften Dateien löschen, so drücken Sie "Löschen" und über "Neu" können sie eine neue Datei erstellen.

## Fehlerliste

Durch Drücken des Softkeys "Fehlerliste" wird diese geöffnet.

Fehleranzeige(error.SRC)				Titelzeile mit dem Namen der Datei
Zeile	Spalte	Fehler...	Beschreibung	
51	5	2263	Typ der Laufsc...	Kurzbeschreibung
52	6	2249	Ausdruck ungle...	
55	5	2309	'(' erwartet	
*1				Fehlernummer
				Nummer der fehlerhaften Zeile und Spalte

Damit wechselt die Softkeyleiste:

				->Editor	Aktual.	Schliessen
--	--	--	--	----------	---------	------------



### HINWEIS \*1

Die angezeigten Zeilennummern entsprechen den absoluten Zeilennummern im Programm, wie sie ein normaler ASCII-Editor anzeigen würde. Um eine Übereinstimmung zwischen den Zeilennummern in der Fehleranzeige und den Zeilennummern im KCP zu erzielen, müßten alle Folds geöffnet, die Detailansicht und die DEF-Zeile aktiv sein. Diese Darstellung ist aber etwas unübersichtlich, da alle Informationen zur Verfügung gestellt werden, obwohl sie nicht benötigt werden. Weiter Informationen über die Detailansicht und die DEF-Zeile finden Sie im Abschnitt **[Verstecken von Programmteilen]**.

Aus der Fehleranzeige ist ersichtlich, daß folgende Fehler aufgetreten sind:

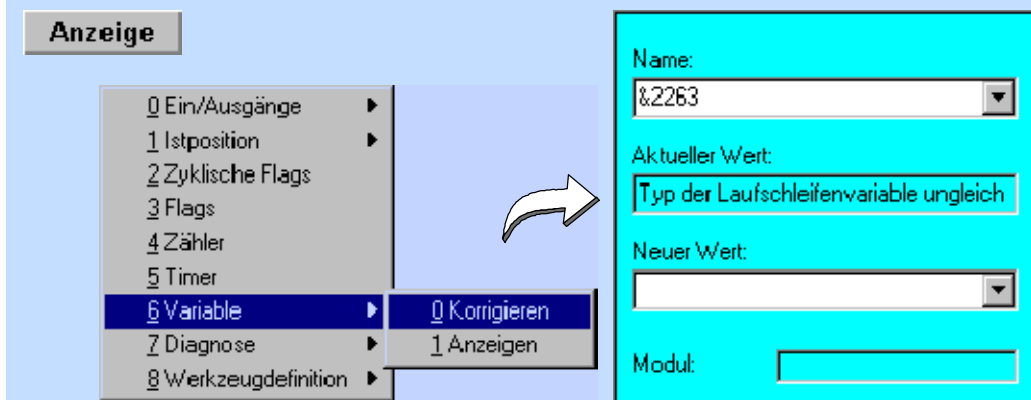
- 3 Zeilen im SRC-File sind fehlerhaft;
- die Zeilennummern der fehlerhaften Zeilen lauten 51, 52 und 55;
- in Zeile 51 die Fehlernummern
  - 2263: Typ der Schleifenvariable ungleich INT;
- in Zeile 52 die Fehlernummer
  - 2249: Ausdruck ungleich INT;
- in Zeile 55 die Fehlermeldung
  - 2309: das Zeichen "(" erwartet;

Aus den Fehlermeldungen "2263" ergibt sich sehr schnell, daß die Variable I nicht als Integer deklariert wurde. Die Fehlermeldung 2249 resultiert ebenfalls aus der fehlenden Deklaration, da bei einer Zählschleife der Zähler immer vom Typ INT sein muß. Die Meldung "2309" bedeutet: Der Compiler interpretiert die Zeile als Unterprogrammaufruf, bei dem allerdings die Klammern fehlen.





Die Bedeutung der Fehlernummern können Sie Online mit Hilfe der Menüfunktion "Anzeige" -> "Variable" -> "Korrigieren" anzeigen lassen. Geben Sie hierzu im Zustandsfenster im Eingabefeld "Name" das Zeichen "&" gefolgt von der Fehlernummer ein. In diesem Fall beispielsweise "&2263" und betätigen die Eingabe-Taste.



->Editor

Wenn Sie nun die SRC-Datei (in diesem Falls "ERROR.SRC") in den Editor laden, können Sie die entsprechenden Korrekturen vornehmen. Zur Erleichterung positioniert sich der Cursor blinkend in der ersten fehlerhaften Zeile. Beachten Sie, daß die begrenzte Sichtbarkeit ausgeschaltet sowie die DEF-Zeile sichtbar ist. Näheres finden Sie in Abschnitt **[Verstecken von Programmteilen]**.

Im vorliegenden Beispiel brauchen die Folds nicht geöffnet zu werden. Wird dies gewünscht, verwenden Sie den Menübefehl "Bearbeiten" -> "Folds" -> "alle FOLDS öffnen".



Die im ursprünglich erstellten Programm fehlende Zeile "INT I" muß **vor** der Zeile "INI" eingefügt werden. Dies ist nur möglich, wenn die Zeile "DEF ERROR ( )" sichtbar ist.

Fügen Sie also die Zeile

**INT I**

vor der INI-Zeile ein, und streichen Sie in Zeile 62 ein Gleichheitszeichen.

**I = I + 1**

<pre> 1  DEF ERROR( ) 2  INT I 3  INI 4  PTP HOME  Vel= 100 % DEFAULT 5 6  FOR I=1 TO 4 7  \$OUT[I]=TRUE 8  ENDFOR 9 10 I=I+1 11 12 PTP HOME  Vel= 100 % DEFAULT 13  END </pre>	<p>_____ Diese Zeile hier einfügen</p> <p>_____ Ein Gleichheitszeichen entfernen</p>
---	--

**Aktual.**

Nach Schließen des Editors und dem Abspeichern der korrigierten Datei können Sie bei der Fehlerliste den Softkey "Aktual." betätigen, und die Fehlerliste verschwindet wenn alle Fehler behoben sind.

## 2.7 Kommentare

Kommentare sind ein wichtiger Bestandteil jedes Computerprogrammes. Dadurch können Sie Ihr Programm übersichtlich und auch für andere verständlich machen. Die Bearbeitungsgeschwindigkeit des Programmes wird durch Kommentare nicht beeinflusst.

Kommentare können Sie an jeder Stelle eines Programmes einfügen. Sie werden stets mit einem Strichpunkt ";" eingeleitet, z.B.:

```
...
PTP P1                ;Bewegung zum Ausgangspunkt
...
;--- Ausgaenge zuruecksetzen ---
FOR I = 1 TO 16
    $OUT[I] = FALSE
ENDFOR
...
```



NOTIZEN:

## 3 Variablen und Vereinbarungen

### 3.1 Variablen und Namen

Neben der Verwendung von Konstanten, also der direkten Wertangabe in Form von Zahlen, Zeichen, etc., können in KRL auch Variablen und andere Formen von Daten im Programm benutzt werden.

Bei der Programmierung von Industrierobotern sind Variablen beispielsweise für die Sensorverarbeitung notwendig. Sie erlauben es, den vom Sensor eingelesenen Wert zu speichern und an verschiedenen Stellen im Programm auszuwerten. Außerdem können arithmetische Operationen durchgeführt werden, um etwa eine neue Position zu berechnen.

Eine Variable wird im Programm durch einen Namen dargestellt, wobei die Bezeichnung des Namens in gewissen Grenzen frei wählbar ist.

Namen

#### Namen in KRL

- dürfen maximal 24 Zeichen lang sein,
- dürfen Buchstaben (A–Z), Ziffern (0–9) sowie die Zeichen '\_' und '\$' enthalten,
- dürfen nicht mit Ziffern beginnen,
- dürfen keine Schlüsselwörter sein.



Da alle Systemvariablen (s. Abschnitt 3.4) mit dem '\$'–Zeichen beginnen, sollten Sie dieses Zeichen nicht als erstes Zeichen in selbstdefinierten Namen verwenden.

Gültige KRL–Namen sind z.B.

SENSOR\_1

KLEBEDUESE13

P1\_BIS\_P12

Eine Variable ist als fester Speicherbereich anzusehen, dessen Inhalt über den Variablennamen ansprechbar ist. Die Variable ist daher zur Laufzeit des Programms durch einen Speicherplatz (Ort) und einen Speicherinhalt (Wert) realisiert.

Wertzuweisung

Mit dem Gleichheitszeichen (=) werden den Variablen Werte zugewiesen. Die Anweisung

ANZAHL = 5

bedeutet also, daß im Speicherbereich mit der Adresse von ANZAHL der Wert 5 eingetragen wird. Wie die Adresse genau aussieht, ist für den Programmierer uninteressant, sie wird daher selbständig vom Compiler vergeben. Wichtig ist nur, daß der Speicherinhalt mit Hilfe seines Namens jederzeit im Programm ansprechbar ist.

Da unterschiedliche Datenobjekte (s. Abschnitt 3.2) auch unterschiedlichen Speicherbedarf haben, muß der Datentyp einer Variablen vor der Verwendung vereinbart (deklariert) werden (s. Abschnitt 3.2.1).

**Lebensdauer** Die Lebensdauer einer Variablen ist die Zeit, während der der Variablen Speicherplatz zugewiesen ist. Sie ist abhängig davon, ob die Variable in einer SRC-Datei oder einer Datenliste deklariert ist:

- **Variable in einer SRC-Datei deklariert**

Die Lebensdauer beschränkt sich auf die Laufzeit des Programms. Nach Beendigung der Abarbeitung wird der Speicherbereich wieder freigegeben. Der Wert der Variablen geht somit verloren.

- **Variable in einer Datenliste (s. Kapitel Datenlisten) deklariert**

Die Lebensdauer ist unabhängig von der Laufzeit des Programms. Die Variable existiert so lange, wie die Datenliste existiert. Solche Variablen sind also permanent (bis zum nächsten Ein-/Ausschalten).



### NOTIZEN:

## 3.2 Datenobjekte

Unter Datenobjekten sind benennbare Speichereinheiten eines bestimmten Datentyps zu verstehen. Die Speichereinheiten können dabei aus unterschiedlich vielen Speicherzellen (Bytes, Worte, etc.) bestehen. Wird ein solches Datenobjekt vom Programmierer unter einem Namen vereinbart, erhält man eine Variable. Die Variable belegt nun eine oder mehrere Speicherzellen, in denen Daten durch das Programm geschrieben und gelesen werden können. Durch die symbolische Benennung der Speicherzellen mit frei wählbaren Namen wird die Programmierung einfacher und übersichtlicher, das Programm besser lesbar.

Zur Klärung des Begriffes Datentyp diene folgendes Beispiel: In einer Speicherzelle mit 8 Bit befinde sich die Bitkombination

00110101

Wie ist diese Bitkombination zu interpretieren? Handelt es sich um die binäre Darstellung der Zahl 53 oder um das ASCII-Zeichen "5", was mit dem gleichen Bitmuster dargestellt wird?

**Datentyp** Zur eindeutigen Beantwortung dieser Frage fehlt noch eine wichtige Information, nämlich die Angabe des Datentyps eines Datenobjekts. Im obigen Fall könnte das beispielsweise der Typ "ganze Zahl" (INTEGER) oder "Zeichen" (CHARACTER) sein.

Neben diesem computertechnischen Grund für die Einführung von Datentypen, ist die Verwendung von Datentypen auch benutzerfreundlicher, da man mit genau den Typen arbeiten kann, die für die spezielle Anwendung besonders gut geeignet sind.

### 3.2.1 Vereinbarung und Initialisierung von Datenobjekten

**DECL** Die Zuordnung eines Variablennamens zu einem Datentyp und die Reservierung des Speicherplatzes geschieht in KRL mit Hilfe der DECL-Vereinbarung. Mit

```
DECL INT ANZAHL, NUMMER
```

vereinbaren Sie z.B. zwei Variablen ANZAHL und NUMMER vom Datentyp "ganze Zahl" (Integer).

Damit kennt der Compiler diese beiden Variablen sowie den zugehörigen Datentyp und kann bei Benutzung der Variablen überprüfen, ob dieser Datentyp die beabsichtigte Operation überhaupt erlaubt.

Die Deklaration beginnt, wie im Beispiel gezeigt, mit dem Schlüsselwort DECL, gefolgt vom Datentyp und der Liste von Variablen, die diesen Datentyp erhalten sollen.



Bei der Deklaration von Variablen und Feldern eines vordefinierten Datentyps kann das Schlüsselwort DECL entfallen. Neben den einfachen Datentypen INT, REAL, CHAR und BOOL (s. Abschnitt 3.2.2) sind unter anderem die Strukturdatentypen POS, E6POS, FRAME, AXIS und E6AXIS (s. Abschnitt 3.2.5) vordefiniert. Für Variablen (keine Felder!) des Datentyps POS kann die Deklaration komplett entfallen. Der Datentyp POS gilt als Standarddatentyp für Variablen. Unverzichtbar ist das Schlüsselwort DECL bei der Vereinbarung frei definierbarer Struktur- oder Aufzählungstypen (s. Abschnitt 3.2.5 und 3.2.6).

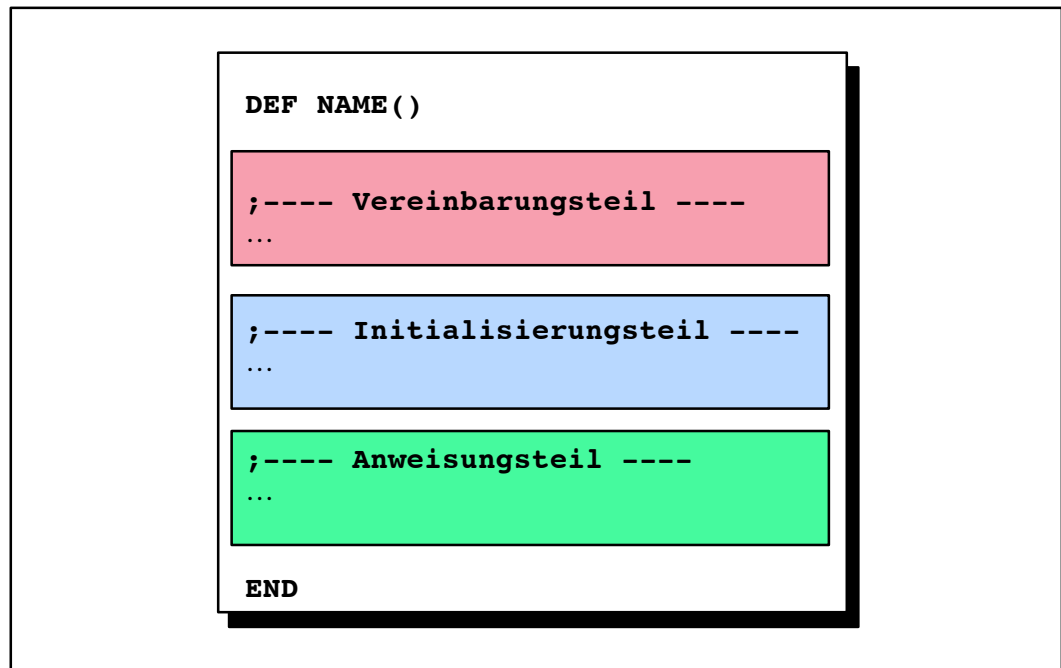
**Initialisierung** Nach der Vereinbarung einer Variablen ist deren Wert zunächst auf ungültig gesetzt, da er sonst von der zufälligen Speicherbelegung abhängen würde. Um mit der Variablen arbeiten zu können, muß sie daher mit einem bestimmten Wert vorbelegt werden. Diese 1. Wertzuweisung an eine Variable nennt man Initialisierung.



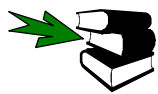
Bei der Erstellung neuer Dateien über den Softkey "Neu" auf der KUKA-Bedienoberfläche wird automatisch auch eine INI-Sequenz erzeugt. Die Vereinbarung von Variablen muß stets vor dieser Sequenz erfolgen.

Eine Wertzuweisung an eine Variable ist eine Anweisung, und darf daher grundsätzlich nicht im Vereinbarungsteil stehen. Die Initialisierung kann aber jederzeit im Anweisungsteil erfolgen. Alle vereinbarten Variablen sollten jedoch zweckmäßigerweise in einem Initialisierungsabschnitt direkt nach dem Deklarationsteil initialisiert werden (s. Abb. 1).

Nur in Datenlisten ist es zulässig, Variablen direkt in der Deklarationszeile zu initialisieren.



**Abb. 1** Grundaufbau eines Roboterprogramms



Weitere Informationen finden Sie im Kapitel **[Datenlisten]**.

### 3.2.2 Einfache Datentypen

Unter den einfachen Datentypen versteht man einige grundsätzliche Datentypen, die in den meisten Programmiersprachen vorhanden sind. Einfache Datentypen enthalten im Gegensatz zu den strukturierten Datentypen (s. Abschnitt 3.2.3–3.2.6) nur einen einzigen Wert. Die in KRL bekannten Datentypen sind zusammen mit ihrem jeweiligen Wertebereich in Tab. 1 aufgelistet.

Datentyp	Integer	Real	Boolean	Character
Schlüsselwort	<b>INT</b>	<b>REAL</b>	<b>BOOL</b>	<b>CHAR</b>
Bedeutung	ganze Zahl	Gleitkomma- zahl	logischer Zu- stand	1 Zeichen
Wertebereich	$-2^{31} \dots 2^{31}-1$	$\pm 1.1\text{E}-38 \dots$ $\pm 3.4\text{E}+38$	TRUE, FALSE	ASCII-Zeichen

**Tab. 1** Einfacher Datentyp

**INT** Der Datentyp Integer ist eine Teilmenge aus der Menge der ganzen Zahlen. Eine Teilmenge kann es deshalb nur sein, weil kein Rechner die theoretisch unendliche Menge der ganzen Zahlen darstellen kann. Die in der KR C1 für Integer-Typen vorgesehenen 32 Bit ergeben daher  $2^{31}$  ganze Zahlen plus Vorzeichen. Die Zahl 0 wird dabei zu den positiven Zahlen gezählt.  
Mit

```
NUMMER = -23456
```

wird der Variablen NUMMER der Wert -23456 zugewiesen.

Weisen Sie einer INTEGER-Variablen einen REAL-Wert zu, so wird der Wert nach den allgemeinen Regeln gerundet (x.0 bis x.49 abrunden, x.5 bis x.99 aufrunden). Durch die Anweisung

```
NUMMER = 45.78
```

erhält die INTEGER-Variable NUMMER den Wert 46.



Ausnahme: Bei Integer-Division wird die Nachkommastelle abgeschnitten, z.B.:  
 $7/4 = 1$

Binärsystem  
Hexadezimal-  
system

Während der Mensch im Dezimalsystem rechnet und denkt, kennt ein Computer nur Nullen und Einsen, welche die beiden Zustände Aus und Ein repräsentieren. Ein Zustand (Aus oder Ein) wird also mit einem Bit dargestellt. Aus Geschwindigkeitsgründen greift der Rechner im allgemeinen auf ein ganzes Paket solcher Nullen und Einsen zu. Typische Paketgrößen sind 8 Bit (=1 Byte), 16 Bit oder 32 Bit. Bei maschinennahen Operationen ist daher oftmals die Darstellung im Binärsystem (2er-System: Ziffern 0 und 1) oder im Hexadezimalsystem (16er-System: Ziffern 0–9 und A–F) hilfreich. Binäre oder hexadezimale Integerwerte können Sie in KRL mit Hilfe des Hochkommas (') und der Angabe von B für Binärdarstellung oder H für Hexadezimaldarstellung angeben.

D	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

**Tab. 2** Die ersten 17 Zahlen im Dezimal- und Hexadezimalsystem

Die Zahl 90 können Sie also in KRL auf drei verschiedene Arten einer Integervariablen zuweisen:

```
INTZAHL = 90 ;Dezimalsystem
INTZAHL = 'B1011010' ;Binaersystem
INTZAHL = 'H5A' ;Hexadezimalsystem
```

Bin → Dez Die Umrechnung von Binärzahlen in das Dezimalsystem gestaltet sich wie folgt:

1	0	1	1	0	1	0	$= 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 90$
$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	

Hex → Dez Zum Transfer von Zahlen aus dem Hexadezimalsystem in das Dezimalsystem gehen Sie folgendermaßen vor:

5	A	$= 5 \cdot 16^1 + 10 \cdot 16^0 = 90$
$16^1$	$16^0$	

REAL Bei dem Begriff der Gleitkommadarstellung handelt es sich um die Aufteilung einer Zahl in Mantisse und Exponent und deren Darstellung in normierter Form. So wird z.B.

```
5.3 als 0.53000000 E+01
-100 als -0.10000000 E+03
0.0513 als 0.51300000 E-01
```

dargestellt.

Beim Rechnen mit Realwerten muß wegen der begrenzten Anzahl der Gleitkommastellen und der damit einhergehenden Ungenauigkeit beachtet werden, daß die gewohnten algebraischen Gesetze nicht mehr in allen Fällen gültig sind. So gilt beispielsweise in der Algebra:

$$\frac{1}{3} \times 3 = 1$$

Läßt man dies einen Computer nachrechnen, so kann sich ergeben, daß das Ergebnis nur 0.99999999 E+00 ist. Ein logischer Vergleich dieser Zahl mit der Zahl 1 ergäbe den Wert FALSE. Für praktische Anwendungen im Roboterbereich reicht diese Genauigkeit jedoch im allgemeinen aus, wenn man beachtet, daß der logische Test auf Gleichheit bei Realgrößen nur innerhalb eines kleinen Toleranzbereichs sinnvoll durchgeführt werden kann.

Zulässige Zuweisungen an Real-Variablen sind z.B.:

```
REALZAHL1 = -13.653
REALZAHL2 = 10
REALZAHL3 = 34.56 E-12
```



Wird einer REAL-Variablen ein INTEGER-Wert zugewiesen, so wird eine automatische Typumwandlung nach REAL vorgenommen. Die Variable REALZAHL2 hat also nach der obigen Zuweisung den Wert 10.0!

BOOL Die boolschen Variablen dienen zur Beschreibung von logischen Zuständen (z.B. Ein-/Ausgängen). Sie können nur die Werte TRUE (wahr) und FALSE (falsch) annehmen:

```
ZUSTAND1 = TRUE
ZUSTAND2 = FALSE
```



**CHAR** Character-Variablen können genau 1 Zeichen aus dem ASCII-Zeichensatz darstellen. Bei der Zuweisung eines ASCII-Zeichens zu einer CHAR-Variablen muß das zugewiesene Zeichen in Anführungszeichen (") eingeschlossen sein:

```
ZEICHEN1 = "G"
```

```
ZEICHEN2 = "?"
```

Zur Speicherung ganzer Zeichenfolgen siehe Abschnitt 3.2.4.

### 3.2.3 Felder

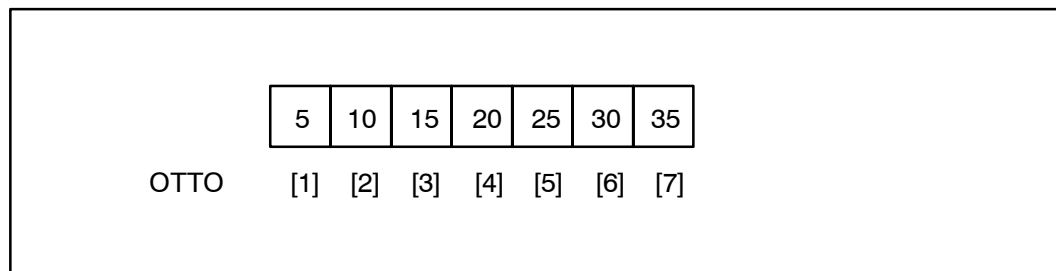
Unter Feldern versteht man die Zusammenfassung von Objekten gleichen Datentyps zu einem Datenobjekt, wobei die einzelnen Komponenten eines Feldes über Indizes angesprochen werden können. Durch die Vereinbarung

```
DECL INT OTTO[ 7 ]
```

**Feldindex** können Sie z.B. 7 verschiedene Integer-Zahlen in dem Feld OTTO[ ] ablegen. Auf jede einzelne Komponente des Feldes können Sie durch Angabe des zugehörigen Index zugreifen (erster Index ist immer die 1):

```
OTTO[ 1 ] = 5 ; dem 1. Element wird die Zahl 5 zugewiesen
OTTO[ 2 ] = 10 ; dem 2. Element wird die Zahl 10 zugewiesen
OTTO[ 3 ] = 15 ; dem 3. Element wird die Zahl 15 zugewiesen
OTTO[ 4 ] = 20 ; dem 4. Element wird die Zahl 20 zugewiesen
OTTO[ 5 ] = 25 ; dem 5. Element wird die Zahl 25 zugewiesen
OTTO[ 6 ] = 30 ; dem 6. Element wird die Zahl 30 zugewiesen
OTTO[ 7 ] = 35 ; dem 7. Element wird die Zahl 35 zugewiesen
```

Anschaulich kann man sich das Feld mit dem Namen OTTO[ ] als Regal mit 7 Fächern vorstellen. Die Fächerbelegung würde nach den vorangegangenen Zuweisungen nun so aussehen:



**Abb. 2** Darstellung eines eindimensionalen Feldes

Sollen nun alle Elemente eines Feldes mit der gleichen Zahl, z.B. 0, initialisiert werden, so müssen Sie nicht jede Zuweisung explizit programmieren, sondern Sie können die Vorbesetzung mit Hilfe einer Schleife und einer Zählvariablen "automatisieren":

```
FOR I = 1 TO 7
    OTTO[ I ] = 0
ENDFOR
```



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

Die Zählvariable ist in diesem Fall die Integervariable I. Sie muß vor der Verwendung als Integer deklariert worden sein.



- Der Datentyp eines Feldes ist beliebig. Somit können die einzelnen Elemente wiederum aus zusammengesetzten Datentypen bestehen (z.B.: Feld aus Feldern)
- Für die Index sind nur Integer-Datentypen zulässig
- Neben Konstanten und Variablen sind auch arithmetische Ausdrücke für den Index zulässig (s. Abschnitt 3.3.1)
- Der Index zählt immer ab 1

2-dimens.

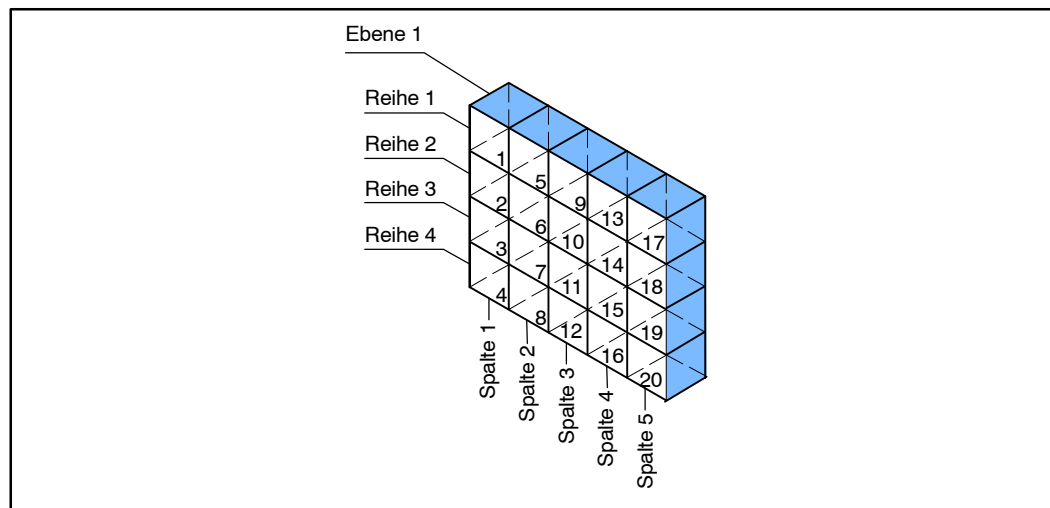
Neben den bisher besprochenen eindimensionalen Feldern, d.h. Feldern mit nur einem Index, können Sie in KRL auch zwei- oder dreidimensionale Felder verwenden. Mit

```
DECL REAL MATRIX[ 5, 4 ]
```

vereinbaren Sie ein zweidimensionales  $5 \times 4$  Feld mit  $5 \times 4 = 20$  REAL-Elementen. Anschaulich läßt sich dieses Feld als Matrix mit 5 Spalten und 4 Reihen darstellen. Mit der Programmsequenz

```
I[ 3 ] = 0
FOR SPALTE = 1 TO 5
  FOR REIHE = 1 TO 4
    I[ 3 ] = I[ 3 ] + 1
    MATRIX[ SPALTE, REIHE ] = I[ 3 ]
  ENDFOR
ENDFOR
```

werden die Elemente der Matrix mit einem Wert entsprechend der Reihenfolge ihrer Belegung besetzt. Man erhält daher folgende Matrixbelegung:



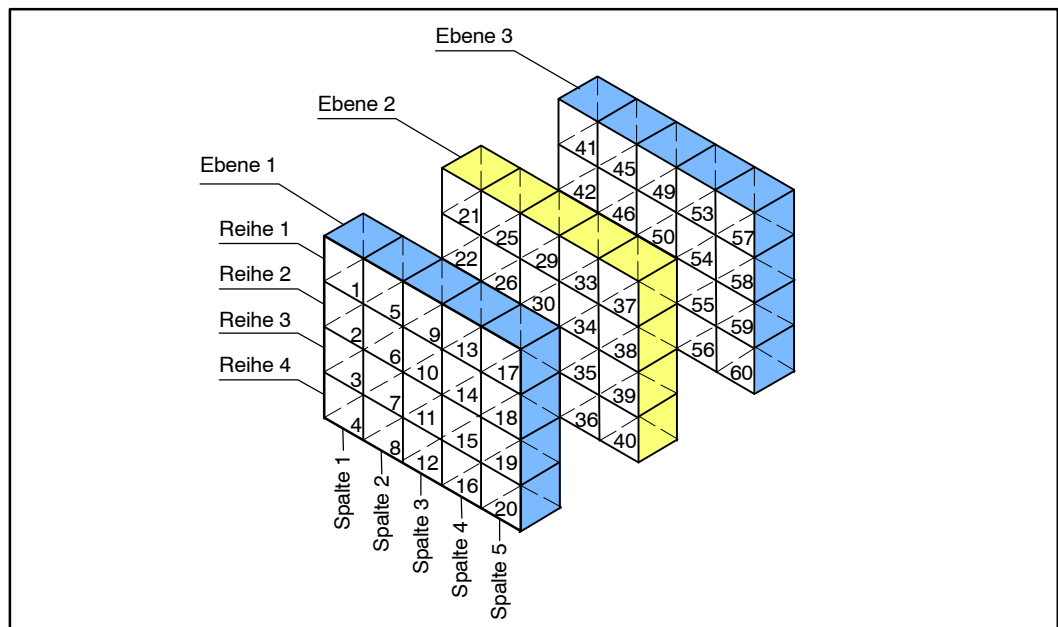
**Abb. 3** Darstellung eines zweidimensionalen Feldes

3-dimens. Dreidimensionale Felder kann man sich schließlich so vorstellen, daß mehrere zweidimensionale Matrizen hintereinander liegen. Die dritte Dimension gibt also sozusagen die Ebene an, in der die Matrix liegt (s. Abb. 4). Vereinfacht wird ein dreidimensionales Feld analog zu den ein- oder zweidimensionalen Feldern, z.B.:

```
DECL BOOL FELD_3D[ 3, 5, 4 ]
```

Die Initialisierungssequenz könnte so aussehen:

```
FOR EBENE = 1 TO 3
  FOR SPALTE = 1 TO 5
    FOR REIHE = 1 TO 4
      FELD_3D[EBENE,SPALTE,REIHE] = FALSE
    ENDFOR
  ENDFOR
ENDFOR
```



**Abb. 4** Darstellung eines dreidimensionalen Feldes

## 3.2.4 Zeichenketten

Mit dem Datentyp `CHAR` können Sie, wie beschrieben, nur einzelne Zeichen abspeichern. Zur Verwendung ganzer Zeichenketten, also z.B. von Wörtern, definiert man einfach ein eindimensionales Feld vom Typ `CHAR`:

```
DECL CHAR NAME[ 8 ]
```

Sie können jetzt wie bisher üblich jedes einzelne Element des Feldes `NAME[ ]` ansprechen, z.B.:

```
NAME[ 3 ] = "G"
```



Sie können aber auch gleich ganze Zeichenketten eingeben:

```
NAME[ ] = "ABCDEFGG"
```

belegt die ersten sieben Elemente des Feldes `NAME[ ]` mit den Buchstaben A, B, C, D, E, F und G:



## 3.2.5 Strukturen

STRUC

Sollen verschiedene Datentypen zusammengefaßt werden, dann ist das Feld ungeeignet und man muß auf die allgemeinere Form des Verbundes zurückgreifen. Mit der Vereinbarungsanweisung `STRUC` können unterschiedliche Datentypen, die zuvor definiert wurden bzw. vordefinierte Datentypen sind, zu einem neuen Verbunddatentyp zusammengefaßt werden. Insbesondere können auch andere Verbunde und Felder Bestandteil eines Verbundes sein.

Typisches Beispiel für die Verwendung von Verbunden ist der Standarddatentyp `POS`. Er besteht aus 6 `REAL`-Werten und 2 `INT`-Werten und wurde in der Datei `$OPERATE.SRC` folgendermaßen definiert:

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

Punkt-Separator

Verwenden Sie jetzt z.B. eine Variable `POSITION` vom Strukturdatentyp `POS`, so können Sie die Elemente entweder einzeln mit Hilfe des Punkt-Separators, z.B.:

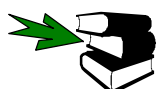
```
POSITION.X = 34.4
POSITION.Y = -23.2
POSITION.Z = 100.0
POSITION.A = 90
POSITION.B = 29.5
POSITION.C = 3.5
POSITION.S = 2
POSITION.T = 6
```

Aggregat

oder gemeinsam mittels eines sogenannten Aggregates

```
POSITION={X 34.4,Y -23.2,Z 100.0,A 90,B 29.5,C 3.5,S 2,T 6}
```

besetzen.



Weitere Informationen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Vereinbarung und Initialisierung von Datenobjekten]**.

Für Aggregate gelten folgende Bestimmungen:



- Die Werte eines Aggregats können einfache Konstanten oder selbst Aggregate sein.
- In einem Aggregat müssen nicht sämtliche Komponenten der Struktur angegeben werden.
- Die Komponenten brauchen nicht in der Reihenfolge angegeben zu werden, in der diese definiert wurden.
- Jede Komponente darf in einem Aggregat nur einmal enthalten sein.
- Bei Feldern aus Strukturen beschreibt ein Aggregat den Wert eines einzelnen Feldelementes.
- Am Anfang eines Aggregates kann – durch Doppelpunkt abgetrennt – der Name des Strukturtyps angegeben sein.

Folgende Zuweisungen sind für POS-Variablen also beispielsweise auch zulässig:

```
POSITION={B 100.0,X 29.5,T 6}
POSITION={A 54.6,B -125.64,C 245.6}
POSITION={POS: X 230,Y 0.0,Z 342.5}
```

Bei POS, E6POS, AXIS, E6AXIS und FRAME-Strukturen werden fehlende Komponenten nicht verändert. Bei allen sonstigen Aggregaten werden nicht vorhandene Komponenten auf ungültig gesetzt.

Das Vorgehen beim Erstellen eigener Strukturvariablen sei an folgendem Beispiel erläutert:

In ein Unterprogramm zum Lichtbogenschweißen soll in einer Variablen S\_PARA folgende Information übergeben werden:

REAL	V_DRAHT	Drahtgeschwindigkeit
INT	KENNL	Kennlinie 0...100%
BOOL	LIBO	mit/ohne Lichtbogen (für Simulation)

Die Variable S\_PARA muß aus 3 Elementen unterschiedlichen Datentyps bestehen. Zunächst muß ein neuer Datentyp definiert werden, der diese Forderungen erfüllt:

```
STRUC SCHWEISSTYP REAL V_DRAHT, INT KENNL, BOOL LIBO
```

Hiermit ist ein neuer Datentyp mit der Bezeichnung SCHWEISSTYP entstanden (SCHWEISSTYP ist keine Variable!). SCHWEISSTYP besteht aus den 3 Komponenten V\_DRAHT, KENNL und LIBO. Nun können Sie eine beliebige Variable des neuen Datentyps deklarieren, z.B.:

```
DECL SCHWEISSTYP S_PARA
```

Damit haben Sie eine Variable S\_PARA des Datentyps SCHWEISSTYP geschaffen. Die einzelnen Elemente lassen sich – wie schon beschrieben – mit Hilfe des Punkt-Separators oder des Aggregats ansprechen:

```
S_PARA.V_DRAHT = 10.2
S_PARA.KENNL = 66
S_PARA.LIBO = TRUE
```

oder

```
S_PARA = {V_DRAHT 10.2, KENNL 50, LIBO TRUE}
```



Um selbstdefinierte Datentypen von Variablen besser unterscheiden zu können, sollten die Namen der neuen Datentypen mit ...TYP enden.

vordefinierte  
Strukturen

In der Datei \$OPERATE.SRC sind folgende Strukturen vordefiniert:

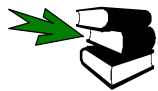
```
STRUC AXIS      REAL  A1,A2,A3,A4,A5,A6
STRUC E6AXIS    REAL  A1,A2,A3,A4,A5,A6,E1,E2,E3,E4,E5,E6
STRUC FRAME     REAL  X,Y,Z,A,B,C
STRUC POS       REAL  X,Y,Z,A,B,C, INT S,T
STRUC E6POS     REAL  X,Y,Z,A,B,C,E1,E2,E3,E4,E5,E6, INT S,T
```

Die Komponenten A1...A6 der Struktur **AXIS** sind Winkelwerte (rotatorische Achsen) oder Translationswerte (translatorische Achsen) zum achsspezifischen Verfahren der Roboterachsen 1...6.

Mit den zusätzlichen Komponenten E1...E6 in der Struktur **E6AXIS** können Zusatzachsen angesprochen werden.

In der Struktur **FRAME** können Sie 3 Positionswerte im Raum (X,Y und Z) sowie 3 Orientierungen im Raum (A, B und C) festlegen. Ein Punkt im Raum ist damit nach Lage und Orientierung eindeutig definiert.

Da es Roboter gibt, die ein und den selben Punkt im Raum mit mehreren Achsstellungen anfahren können, dienen die Integervariablen S und T in der Struktur **POS** zum Festlegen einer eindeutigen Achsstellung.

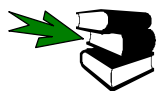


Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Bewegungsbefehle]** Status (S) und Turn (T).

Mit den Komponenten E1...E6 in der Struktur **E6POS** können wieder Zusatzachsen angesprochen werden.

geometrische  
Datentypen

Die Typen **AXIS**, **E6AXIS**, **POS**, **E6POS** und **FRAME** nennt man auch geometrische Datentypen, da der Programmierer mit ihnen auf einfache Weise geometrische Beziehungen beschreiben kann.



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Verwendung verschiedener Koordinatensysteme]**.

## 3.2.6 Aufzählungstypen

Ein Aufzählungsdattentyp ist ein Datentyp, der sich aus einer begrenzten Menge von Konstanten zusammensetzt. Die Konstanten sind frei definierbare Namen und können vom Benutzer festgelegt werden. Eine Variable dieses Datentyps (Aufzählungsvariable) kann nur den Wert einer dieser Konstanten annehmen.

Zur Erläuterung diene die Systemvariable \$MODE\_OP. In ihr wird abgespeichert welche Betriebsart gerade angewählt ist. Zur Auswahl stehen die Betriebsarten T1, T2, AUT und EX.

Man könnte nun \$MODE\_OP als Integervariable deklarieren, jeder Betriebsart eine Zahl zuordnen und diese dann in \$MODE\_OP abspeichern. Dies wäre jedoch sehr unübersichtlich.

ENUM

Eine weitaus elegantere Lösung bietet der Aufzählungstyp. In der Datei \$OPERATE.SRC wurde dazu ein Aufzählungsdattentyp mit dem Namen MODE\_OP generiert:

```
ENUM MODE_OP T1, T2, AUT, EX, INVALID
```

Der Befehl zur Vereinbarung von Aufzählungstypen lautet also ENUM. Variablen des Aufzählungstyps MODE\_OP können nur die Werte T1, T2, AUT, EX oder INVALID annehmen. Die Variablenvereinbarung erfolgt wieder mit dem Schlüsselwort DECL:

```
DECL MODE_OP $MODE_OP
```

Die Aufzählungsvariable \$MODE\_OP können sie nun durch normale Zuweisung mit einem der vier Werte des Datentyps MODE\_OP belegen. Zur Unterscheidung von einfachen Konstanten

# – Zeichen      wird den selbstdefinierten Aufzählungskonstanten bei Initialisierungen oder Abfragen ein “#”-Zeichen vorangestellt, z.B.:

```
$MODE_OP = #T1
```

Mit `ENUM` können Sie sich nun beliebig viele selbstdefinierte Aufzählungsdatentypen erzeugen.



#### NOTIZEN:

### 3.3 Datenmanipulation

Zur Manipulation der verschiedenen Datenobjekte gibt es eine Fülle von Operatoren und Funktionen, mit deren Hilfe Formeln aufgebaut werden können. Die Mächtigkeit einer Roboterprogrammiersprache hängt gleichermaßen von den zugelassenen Datenobjekten und deren Manipulationsmöglichkeiten ab.

#### 3.3.1 Operatoren

Operand

Unter Operatoren sind die üblichen mathematischen Operatoren zu verstehen im Gegensatz zu Funktionen wie beispielsweise  $\text{SIN}(30)$ , welche den Sinus des Winkels  $30^\circ$  liefert. In der Operation  $5+7$  bezeichnet man demnach 5 und 7 als Operanden und + als Operator.

Bei jeder Operation prüft der Compiler die Zulässigkeit der Operanden. So ist beispielsweise  $7 - 3$  als Subtraktion zweier Integerzahlen eine zulässige Operation,  $5 + "A"$  als Addition eines Integerwertes zu einem Zeichen eine ungültige Operation.

Bei manchen Operationen, wie  $5 + 7.1$ , also der Addition von Integer- mit Realwerten wird eine Typanpassung vorgenommen und der Integerwert 5 in den Realwert 5.0 umgewandelt. Auf diese Problematik wird bei der Besprechung der einzelnen Operatoren noch näher eingegangen.

##### 3.3.1.1 Arithmetische Operatoren

Arithmetische Operatoren betreffen die Datentypen INTEGER und REAL. Alle 4 Grundrechenarten sind in KRL zulässig (s. Tab. 3).

Operator	Beschreibung
+	Addition oder positives Vorzeichen
-	Subtraktion oder negatives Vorzeichen
*	Multiplikation
/	Division

**Tab. 3** Arithmetische Operatoren

Das Ergebnis einer arithmetischen Operation ist nur dann INT, wenn beide Operanden vom Typ INTEGER sind. Ist das Ergebnis einer Integerdivision nicht ganzzahlig, so wird die Nachkommastelle abgeschnitten. Wenn mindestens einer der beiden Operanden REAL ist, dann ist auch das Ergebnis vom Typ REAL (s. Tab. 4).

Operanden	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

**Tab. 4** Ergebnis einer arithmetischen Operation



Zur Verdeutlichung dient folgendes Programmbeispiel:



```

DEF ARITH( )

;----- Deklarationsteil -----
INT A,B,C
REAL K,L,M

;----- Initialisierung -----
;vor der initialisierung sind alle variablen ungueltig!
A = 2           ;A=2
B = 9.8         ;B=10
C = 7/4         ;C=1
K = 3.5         ;K=3.5
L = 0.1 E01     ;L=1.0
M = 3           ;M=3.0

;----- Hauptteil -----
A = A * C       ;A=2
B = B - 'HB'    ;B=-1
C = C + K       ;C=5
K = K * 10      ;K=35.0
L = 10 / 4      ;L=2.0
L = 10 / 4.0    ;L=2.5
L = 10 / 4.     ;L=2.5
L = 10./ 4      ;L=2.5
C = 10./ 4.     ;C=3
M = (10/3) * M  ;M=9.0

END

```

### 3.3.1.2 Geometrischer Operator

Der geometrische Operator wird in KRL durch einen Doppelpunkt ":" symbolisiert. Er führt zwischen den Datentypen **FRAME** und **POS** eine Frameverknüpfung durch.

Die Verknüpfung zweier Frames ist die übliche Transformation von Koordinatensystemen. Daher wirkt sich die Verknüpfung einer **FRAME**- mit einer **POS**-Struktur nur auf das Frame innerhalb der **POS**-Struktur aus. Die Komponenten **S** und **T** bleiben von der Transformation unberührt und müssen daher auch nicht mit einem Wert besetzt sein. Die Werte **X**, **Y**, **Z**, **A**, **B** und **C** müssen jedoch sowohl bei **POS**-Operanden als auch bei **FRAME**-Operanden immer mit einem Wert besetzt sein.

Frame-  
verknüpfung

Eine Frameverknüpfung wird von links nach rechts ausgewertet. Das Ergebnis hat immer den Datentyp des am weitesten rechts stehenden Operanden (s. Tab. 5).

linker Operand (Bezugs-KS)	Operator	rechter Operand (Ziel-KS)	Ergebnis
POS	:	POS	<b>POS</b>
POS	:	FRAME	<b>FRAME</b>
FRAME	:	POS	<b>POS</b>
FRAME	:	FRAME	<b>FRAME</b>

**Tab. 5** Datentyp-Kombinationen beim geometrischen Operator



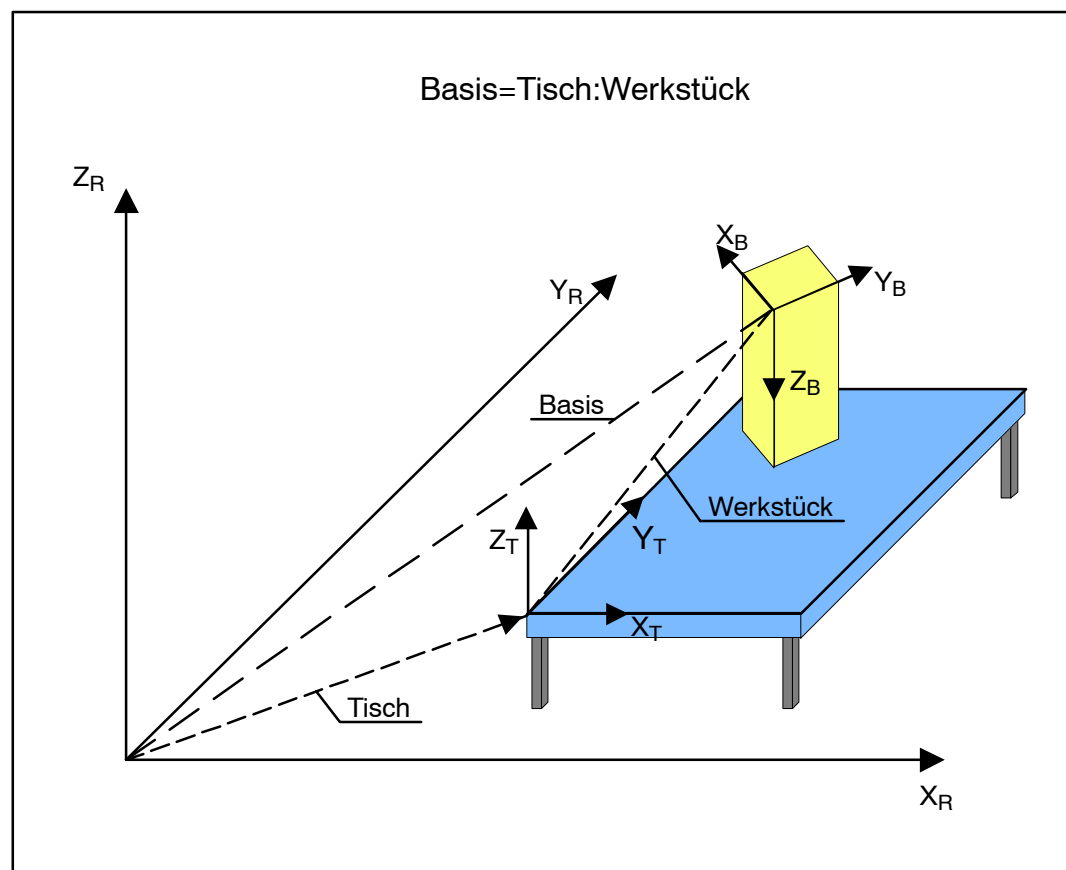
Wenn der linke Operand den Datentyp POS hat, dann findet eine Typanpassung statt. Die durch die POS-Struktur angegebene Position wird in ein Frame umgewandelt. Das heißt, das System ermittelt das Werkzeug-Frame zu dieser Position.

Die Wirkungsweise des geometrischen Operators sei an einem einfachen Beispiel erläutert (s. Abb. 5):

In einem Raum steht ein Tisch. Das RAUM-Koordinatensystem sei als festes Koordinatensystem in der linken vorderen Ecke des Raumes definiert.

Der Tisch steht parallel zu den Wänden des Raumes. Die linke vordere Ecke des Tisches liegt genau 600 mm von der vorderen Wand und 450 mm von der linken Wand des Raumes entfernt. Der Tisch ist 800 mm hoch.

Auf dem Tisch steht ein quaderförmiges Werkstück. Das WERKSTUECK-Koordinatensystem legen Sie wie in Abb. 5 gezeigt in eine Ecke des Werkstücks. Um das Teil später zweckmäßig handhaben zu können, zeigt die Z-Achse des WERKSTUECK-Koordinatensystems nach unten. Das Werkstück ist bezüglich der Z-Achse des TISCH-Koordinatensystems um 40° gedreht. Die Position des WERKSTUECK-Koordinatensystems bezogen auf das TISCH-Koordinatensystem ist  $X=80$  mm,  $Y=110$  mm und  $Z=55$  mm.



**Abb. 5** Wirkungsweise des geometrischen Operators

Die Aufgabenstellung ist nun, das WERKSTUECK-Koordinatensystem bezüglich des RAUM-Koordinatensystems zu beschreiben. Dazu vereinbaren Sie zunächst folgende Framevariablen:

FRAME TISCH, WERKSTUECK, BASIS

Das RAUM-Koordinatensystem sei bereits systemspezifisch festgelegt. Die Koordinatensysteme TISCH und WERKSTUECK werden nun entsprechend den Randbedingungen initialisiert:

TISCH = {X 450,Y 600,Z 800,A 0,B 0,C 0}

WERKSTUECK = {X 80,Y 110,Z 55,A -40,B 180,C 0}

Das WERKSTUECK-Koordinatensystem bezüglich des RAUM-Koordinatensystems ergibt sich nun mit Hilfe des geometrischen Operators zu

$\text{BASIS} = \text{TISCH:WERKSTUECK}$

In unserem Fall ist BASIS nun folgendermaßen besetzt:

$\text{BASIS} = \{X\ 530, Y\ 710, Z\ 855, A\ 140, B\ 0, C\ -180\}$

Eine weitere Möglichkeit wäre:

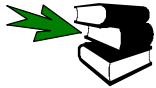
$\text{BASIS} = \{X\ 530, Y\ 710, Z\ 855, A\ -40, B\ 180, C\ 0\}$



Nur in diesem speziellen Fall ergeben sich die Komponenten von BASIS als Addition der Komponenten von TISCH und WERKSTUECK. Dies liegt daran, daß das TISCH-Koordinatensystem nicht bezüglich des RAUM-Koordinatensystems verdreht ist.

Im allgemeinen ist jedoch eine einfache Addition der Komponenten nicht möglich!

Eine Frameverknüpfung ist auch nicht kommutativ, das heißt, durch Vertauschen von Bezugsframe und Zielframe wird sich normalerweise auch das Ergebnis ändern!



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Verwendung verschiedener Koordinatensysteme]**.

Zur Anwendung des geometrischen Operators ein weiteres Beispiel: Verschiedene Koordinatensysteme und Verknüpfungen von Koordinatensystemen werden darin angefahren. Zur Verdeutlichung von Orientierungsänderungen verfährt die Werkzeugspitze in jedem Koordinatensystem zunächst ein Stück in X-Richtung, dann ein Stück in Y-Richtung und schließlich ein Stück in Z-Richtung.



```

DEF  GEOM_OP ( );

----- Deklarationsteil -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME           ;Variable HOME vom Typ AXIS
DECL FRAME MYBASE[2]     ;Feld 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 30,A6 0}; Basiskoordinaten-
system setzen
$BASE={X 1000,Y 0,Z 1000,A 0,B 0,C 0}
REF_POS_X={X 100,Y 0,Z 0,A 0,B 0,C 0}      ;Referenzpos.
REF_POS_Y={X 100,Y 100,Z 0,A 0,B 0,C 0}
REF_POS_Z={X 100,Y 100,Z 100,A 0,B 0,C 0}; eigene Koordinaten-
systeme definieren

MYBASE[1]={X 200,Y 100,Z 0,A 0,B 0,C 180}
MYBASE[2]={X 0,Y 200,Z 250,A 0,B 90,C 0};

----- Hauptteil -----
PTP  HOME ; SAK-Fahrt; Bewegung bezueglich des $BASE-Koordina-
tensystems
PTP  $Base           ;Ursprung des $BASE-KS direkt anfahren
WAIT SEC 2           ;2 Sekunden warten
PTP  REF_POS_X       ;100mm in x-Richtung fahren
PTP  REF_POS_Y       ;100mm in y-Richtung fahren
PTP  REF_POS_Z       ;100mm in z-Richtung fahren; Bewegung bezueg-
lich des um MYBASE[1] verschobenen $BASE-KS
PTP  MYBASE[1]
WAIT SEC 2
PTP  MYBASE[1]:REF_POS_X
PTP  MYBASE[1]:REF_POS_Y
PTP  MYBASE[1]:REF_POS_Z; Bewegung bezueglich des um MYBASE[2]
verschobenen $BASE-KS
PTP  MYBASE[2]
WAIT SEC 2
PTP  MYBASE[2]:REF_POS_X
PTP  MYBASE[2]:REF_POS_Y
PTP  MYBASE[2]:REF_POS_Z; Bewegung bez. des um MYBASE[1]:MY-
BASE[2] versch. $BASE-KS
PTP  MYBASE[1]:MYBASE[2]
WAIT SEC 2
PTP  MYBASE[1]:MYBASE[2]:REF_POS_X
PTP  MYBASE[1]:MYBASE[2]:REF_POS_Y
PTP  MYBASE[1]:MYBASE[2]:REF_POS_Z; Bewegung bez. des um MY-
BASE[2]:MYBASE[1] versch. $BASE-KS
PTP  MYBASE[2]:MYBASE[1]
WAIT SEC 2
PTP  MYBASE[2]:MYBASE[1]:REF_POS_X
PTP  MYBASE[2]:MYBASE[1]:REF_POS_Y
PTP  MYBASE[2]:MYBASE[1]:REF_POS_Z
PTP  HOME
END

```

## 3.3.1.3 Vergleichsoperatoren

Mit den in Tab. 6 aufgeführten Vergleichsoperatoren können logische Ausdrücke gebildet werden. Das Ergebnis eines Vergleichs ist daher immer vom Datentyp `BOOL`, da ein Vergleich immer nur wahr (`TRUE`) oder falsch (`FALSE`) sein kann.

Operator	Beschreibung	zulässige Datentypen
<code>==</code>	gleich	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code> , <code>BOOL</code>
<code>&lt;&gt;</code>	ungleich	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code> , <code>BOOL</code>
<code>&gt;</code>	größer	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>
<code>&lt;</code>	kleiner	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>
<code>&gt;=</code>	größer gleich	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>
<code>&lt;=</code>	kleiner gleich	<code>INT</code> , <code>REAL</code> , <code>CHAR</code> , <code>ENUM</code>

Tab. 6 Vergleichsoperatoren

Vergleiche können in Programmablaufanweisungen (s. Abschnitt 6) verwendet werden, das Ergebnis eines Vergleichs kann einer boolschen Variablen zugewiesen werden.

Der Test auf Gleichheit oder Ungleichheit ist bei Realwerten nur bedingt sinnvoll, da durch Rundungsfehler bei Berechnung der zu vergleichenden Werte algebraisch identische Formeln ungleiche Werte liefern können (s. 3.2.2).



- Kombinationen von Operanden aus `INT`, `REAL`, und `CHAR` sind möglich.
- Ein `ENUM`-Typ darf nur mit dem selben `ENUM`-Typ verglichen werden.
- Ein `BOOL`-Typ darf nur mit einem `BOOL`-Typ verglichen werden.

Der Vergleich von Zahlenwerten (`INT`, `REAL`) und Zeichenwerten (`CHAR`) ist deshalb möglich, weil jedem ASCII-Zeichen ein ASCII-Code zugeordnet ist. Dieser Code ist eine Zahl, die die Reihenfolge der Zeichen im Zeichensatz bestimmt.

Die einzelnen Konstanten eines Aufzählungstyps werden bei ihrer Deklaration in der Reihenfolge ihres Auftretens durchnummeriert. Die Vergleichsoperatoren beziehen sich auf diese Nummern.

Neben einfachen sind auch mehrfache Vergleiche zulässig. Dazu einige Beispiele:

```

...
BOOL A,B
...
B = 10 < 3                                ;B=FALSE
A = 10/3 == 3                             ;A=TRUE
B = ((B == A) <> (10.00001 >= 10)) == TRUE ;B=TRUE
A = "F" < "Z"                             ;A=TRUE
...

```

## 3.3.1.4 Logische Operatoren

Diese dienen zur logischen Verknüpfung von booleschen Variablen, Konstanten und einfachen logischen Ausdrücken, wie sie mit Hilfe der Vergleichsoperatoren gebildet werden. So hat z.B. der Ausdruck

$(A > 5) \text{ AND } (A < 12)$

nur dann den Wert TRUE, wenn A im Bereich zwischen 5 und 12 liegt. Solche Ausdrücke werden häufig in Anweisungen zur Ablaufkontrolle (s. Abschnitt 6) verwendet. Die logischen Operatoren sind in Tab. 7 aufgelistet.

Operator	Operandenzahl	Beschreibung
<b>NOT</b>	1	Invertierung
<b>AND</b>	2	logisches UND
<b>OR</b>	2	logisches ODER
<b>EXOR</b>	2	exklusives ODER

**Tab. 7** Logische Operatoren

Die Operanden einer logischen Verknüpfung müssen vom Typ `BOOL` sein, das Ergebnis ist ebenfalls immer vom Typ `BOOL`. In Tab. 8 sind die möglichen Ergebnisse der jeweiligen Verknüpfungen in Abhängigkeit vom Wert der Operanden dargestellt.

Operation		NOT A	A AND B	A OR B	A EXOR B
<b>A = TRUE</b>	<b>B = TRUE</b>	FALSE	TRUE	TRUE	FALSE
<b>A = TRUE</b>	<b>B = FALSE</b>	FALSE	FALSE	TRUE	TRUE
<b>A = FALSE</b>	<b>B = TRUE</b>	TRUE	FALSE	TRUE	TRUE
<b>A = FALSE</b>	<b>B = FALSE</b>	TRUE	FALSE	FALSE	FALSE

**Tab. 8** Wahrheitstabelle für logische Verknüpfungen

Einige Beispiele zu logischen Verknüpfungen:

```

...
BOOL A,B,C
...
A = TRUE                                ;A=TRUE
B = NOT A                               ;B=FALSE
C = (A AND B) OR NOT (B EXOR NOT A)    ;C=TRUE
A = NOT NOT C                           ;A=TRUE
...
```

## 3.3.1.5 Bit-Operatoren

Mit Hilfe der Bit-Operatoren (s. Tab. 9) lassen sich ganze Zahlen miteinander verknüpfen, indem man die einzelnen Bits der Zahlen logisch miteinander verknüpft. Die Bit-Operatoren verknüpfen einzelne Bits genauso wie die logischen Operatoren zwei boolsche Werte, wenn man den Bit-Wert 1 als `TRUE` und den Wert 0 als `FALSE` ansieht.

Eine bitweise `UND`-Verknüpfung der Zahlen 5 und 12 ergibt somit z.B. die Zahl 4, eine bitweise `ODER`-Verknüpfung die Zahl 13 und eine bitweise exklusiv `ODER`-Verknüpfung die Zahl 9:

	0	1	0	1	= 5
	1	1	0	0	= 12
<b>AND</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	= 4
<b>OR</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	= 13
<b>EXOR</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	= 9

Bei der bitweisen Invertierung werden nicht alle Bits einfach umgedreht. Stattdessen wird bei Verwendung von `B_NOT` zum Operanden 1 dazu addiert und das Vorzeichen gekippt, z.B:

```
B_NOT 10 = -11
B_NOT -10 = 9
```

Bit-Operatoren werden beispielsweise eingesetzt, um digitale Ein-/Ausgangssignale miteinander zu verknüpfen (s. 7.3).

Operator	Operandenzahl	Beschreibung
<b>B_NOT</b>	1	bitweise Invertierung
<b>B_AND</b>	2	bitweise <code>UND</code> -Verknüpfung
<b>B_OR</b>	2	bitweise <code>ODER</code> -Verknüpfung
<b>B_EXOR</b>	2	bitweise exklusive <code>ODER</code> -Verknüpfung

Tab. 9 Logische Bit-Operatoren



Da ASCII-Zeichen auch über den ganzzahligen ASCII-CODE ansprechbar sind, kann der Datentyp der Operanden neben `INT` auch `CHAR` sein. Das Ergebnis ist immer vom Typ `INT`.

Beispiele zur Verwendung von Bit-Operatoren:

```
...
INT A
...
A = 10 B_AND 9           ;A=8
A = 10 B_OR 9            ;A=11
A = 10 B_EXOR 9          ;A=3
A = B_NOT 197            ;A=-198
A = B_NOT 'HC5'          ;A=-198
A = B_NOT 'B11000101'    ;A=-198
A = B_NOT "E"            ;A=-70
...
```

Angenommen, Sie haben einen 8-Bit breiten digitalen Ausgang definiert. Sie können den Ausgang über die INTEGER-Variable DIG ansprechen. Zum Setzen der Bits 0, 2, 3 und 7 können Sie nun einfach

Einblenden  
von Bits

```
DIG = 'B10001101' B_OR DIG
```

programmieren. Alle anderen Bits bleiben unbeeinflusst, egal welchen Wert sie haben.

Wollen Sie z.B. die Bits 1, 2 und 6 ausblenden, so programmieren Sie

Ausblenden  
von Bits

```
DIG = 'B10111001' B_AND DIG
```

Alle anderen Bits werden dadurch nicht verändert.

Genauso leicht können Sie mit den Bit-Operatoren überprüfen, ob einzelne Bits des Ausgangs gesetzt sind. Der Ausdruck

Herausfiltern  
von Bits

```
('B10000001' B_AND DIG) == 'B10000001'
```

wird TRUE, wenn die Bits 0 und 7 gesetzt sind, ansonsten wird er FALSE.

Wollen Sie nur Testen, ob wenigstens eines der beiden Bits 0 oder 7 gesetzt ist, so muß die bitweise UND-Verknüpfung lediglich größer Null sein:

```
('B10000001' B_AND DIG) > 0
```



## 3.3.1.6 Prioritäten von Operatoren

Priorität Verwenden Sie komplexere Ausdrücke mit mehreren Operatoren, so müssen Sie die unterschiedlichen Prioritäten der einzelnen Operatoren beachten (s. Tab. 10), da die einzelnen Ausdrücke in der Reihenfolge ihrer Prioritäten ausgeführt werden.

Priorität	Operator
1	NOT B_NOT
2	* /
3	+ -
4	AND B_AND
5	EXOR B_EXOR
6	OR B_OR
7	== <> < > >= <=

Tab. 10 Prioritäten von Operatoren

Grundsätzlich gilt:

- Geklammerte Ausdrücke werden zuerst bearbeitet.
- Bei ungeklammerten Ausdrücken wird in der Reihenfolge der Priorität ausgewertet.
- Verknüpfungen mit Operatoren gleicher Priorität werden von links nach rechts ausgeführt.

Beispiele:

```

...
INT A,B
BOOL E,F
...
A = 4
B = 7
E = TRUE
F = FALSE
...
E = NOT E OR F AND NOT (-3 + A * 2 > B) ;E=FALSE
A = 4 + 5 * 3 - B_NOT B / 2 ;A=23
B = 7 B_EXOR 3 B_OR 4 B_EXOR 3 B_AND 5 ;B=5
F = TRUE == (5 >= B) AND NOT F ;F=TRUE
...

```

## 3.3.2 Standardfunktionen

Zur Berechnung gewisser mathematischer Probleme sind in KRL einige Standardfunktionen vordefiniert (s. Tab. 11). Sie können ohne weitere Vereinbarung direkt benutzt werden.

Beschreibung	Funktion	Datentyp Argument	Wertebereich Argument	Datentyp Funktion	Wertebereich Ergebnis
Betrag	<b>ABS (X)</b>	REAL	$-\infty \dots +\infty$	REAL	$0 \dots +\infty$
Wurzel	<b>SQRT (X)</b>	REAL	$0 \dots +\infty$	REAL	$0 \dots +\infty$
Sinus	<b>SIN (X)</b>	REAL	$-\infty \dots +\infty$	REAL	$-1 \dots +1$
Cosinus	<b>COS (X)</b>	REAL	$-\infty \dots +\infty$	REAL	$-1 \dots +1$
Tangens	<b>TAN (X)</b>	REAL	$-\infty \dots +\infty^*$	REAL	$-\infty \dots +\infty$
Arcuscos.	<b>ACOS (X)</b>	REAL	$-1 \dots +1$	REAL	$0^\circ \dots 180^\circ$
Arcustang.	<b>ATAN2 (Y, X)</b>	REAL	$-\infty \dots +\infty$	REAL	$-90^\circ \dots +90^\circ$
* keine ungeradzahligen Vielfache von $90^\circ$ , d.h. $X \neq (2k-1) * 90^\circ$ , $k \in \mathbb{N}$					

Tab. 11 Mathematische Standardfunktionen

**Betrag** Die Funktion **ABS (X)** berechnet den Betrag des Wertes x, z.B.:

```
B = -3.4
A = 5 * ABS(B) ;A=17.0
```

**Wurzel** **SQRT (X)** errechnet die Quadratwurzel aus der Zahl x, z.B.:

```
A = SQRT(16.0801) ;A=4.01
```

**Sinus Cosinus Tangens** Die trigonometrischen Funktionen **SIN (X)**, **COS (X)** und **TAN (X)** berechnen den Sinus, Cosinus oder Tangens des Winkels x, z.B.:

```
A = SIN(30) ;A=0.5
B = 2 * COS(45) ;B=1.41421356
C = TAN(45) ;C=1.0
```

Der Tangens von  $\pm 90^\circ$  und ungeradzahligen Vielfachen von  $\pm 90^\circ$  ( $\pm 270^\circ$ ,  $\pm 450^\circ$ ,  $\pm 630^\circ \dots$ ), ist unendlich. Deshalb führt der Versuch der Berechnung eines dieser Werte zu einer Fehlermeldung.

**Arcuskosinus** **ACOS (X)** ist die Umkehrfunktion zu COS(X):

```
A = COS(60) ;A=0.5
B = ACOS(A) ;B=60
```

**Arcussinus** Für den Arcussinus, die Umkehrfunktion zu SIN (X), ist keine Standardfunktion vordefiniert. Aufgrund der Beziehung  $\text{SIN}(X) = \text{COS}(90^\circ - X)$  können sie aber auch diesen sehr leicht berechnen:

```
A = SIN(60) ;A=0.8660254
B = 90 - ACOS(A) ;B=60
```

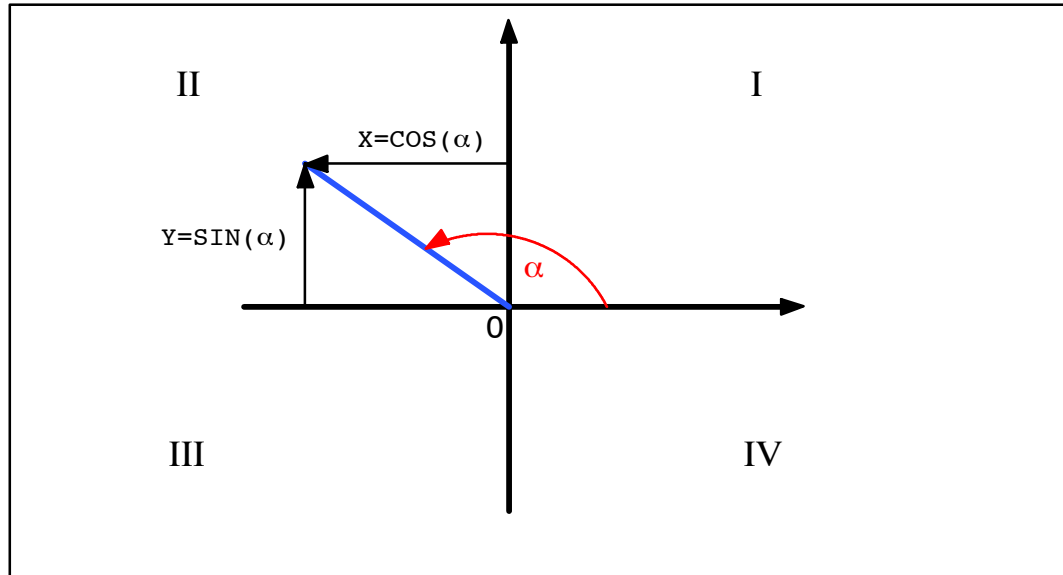
**Arcustangens** Der Tangens eines Winkels ist definiert als Gegenkathete (y) dividiert durch Ankathete (x) im rechtwinkligen Dreieck. Hat man die Länge der beiden Katheten, kann man also den Winkel zwischen Ankathete und Hypotenuse mit dem Arcustangens berechnen.

Betrachtet man jetzt einen Vollkreis, so ist es entscheidend, welches Vorzeichen die Komponenten x und y haben. Würde man nur den Quotienten berücksichtigen, so könnten mit dem Arcustangens nur Winkel zwischen  $0^\circ$  und  $180^\circ$  berechnet werden. Dies ist auch bei allen üblichen Taschenrechnern der Fall: Der Arcustangens von positiven Werten ergibt einen Winkel zwischen  $0^\circ$  und  $90^\circ$ , der Arcustangens von negativen Werten einen Winkel zwischen  $90^\circ$  und  $180^\circ$ .

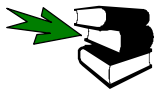
Durch die explizite Angabe von x und y ist durch deren Vorzeichen eindeutig der Quadrant festgelegt, in dem der Winkel liegt (s. Abb. 6). Sie können daher auch Winkel in den Qua-

dranten III und IV berechnen. Deshalb sind zur Berechnung des Arcustangens in der Funktion **ATAN2 (Y, X)** auch diese beiden Angaben notwendig, z.B.:

A = ATAN2 (0.5, 0.5)	;A=45
B = ATAN2 (0.5, -0.5)	;B=135
C = ATAN2 (-0.5, -0.5)	;C=225
D = ATAN2 (-0.5, 0.5)	;D=315



**Abb. 6** Verwendung von x und y in der Funktion ATAN2 (Y, X)



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



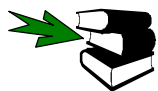
**NOTIZEN:**

### 3.4 Systemvariablen und Systemdateien

Eine wichtige Voraussetzung für die Bearbeitung von komplexen Anwendungen in der Robotertechnik ist eine frei und komfortabel programmierbare Steuerung.

Dazu muß in einfacher Weise die Funktionalität der Robotersteuerung in der Robotersprache programmierbar sein. Erst wenn die Integration der Steuerungsparameter in ein Roboterprogramm vollständig und doch einfach möglich ist, können Sie die volle Funktionalität einer Robotersteuerung ausnützen. Bei der KR C1 ist dies durch das Konzept der vordefinierten Systemvariablen und –dateien hervorragend gelöst.

Beispiele für vordefinierte Variablen sind \$POS\_ACT (aktuelle Roboterposition), \$BASE (Basiskoordinatensystem) oder \$VEL.CP (Bahngeschwindigkeit).



Eine Auflistung aller vordefinierter Variablen finden Sie in der Dokumentation **Anhang, Kapitel [Systemvariablen]**.

Systemvariablen sind vollständig in das Variablenkonzept von KRL integriert. Sie besitzen einen entsprechenden Datentyp und können von Ihnen wie jede andere Variable im Programm gelesen und geschrieben werden, wenn es nicht Einschränkungen wegen der Art der Daten gibt. Die aktuelle Roboterposition kann z.B. nur gelesen, nicht geschrieben werden. Solche Einschränkungen werden von der Steuerung geprüft.

Soweit es sicherheitstechnisch möglich ist, verfügen Sie sogar über den schreibenden Zugriff auf Systemdaten. Damit eröffnen sich viele Möglichkeiten für die Diagnose, da vom KCP und vom Programmiersystem eine Vielzahl von Systemdaten geladen oder beeinflußt werden können.

Nützliche Systemvariablen mit Schreibzugriff sind zum Beispiel \$TIMER[ ] und \$FLAG[ ].

Timer

Die 16 Timervariablen \$TIMER[ 1 ]...\$TIMER[ 16 ] dienen zum Messen von zeitlichen Abläufen und können somit als "Stoppuhr" eingesetzt werden. Das Starten und Stoppen des Meßvorganges erfolgt mit den Systemvariablen

\$TIMER\_STOP[ 1 ]...\$TIMER\_STOP[ 16 ]:

\$TIMER\_STOP[ 4 ] = FALSE

startet beispielsweise den Timer 4,

\$TIMER\_STOP[ 4 ] = TRUE

stoppt den Timer 4 wieder. Über eine normale Wertzuweisung kann die betreffende Timervariable jederzeit zurückgesetzt werden, z.B.:

\$TIMER[ 4 ] = 0

Wechselt der Wert einer Timervariablen von Minus nach Plus, so wird ein zugehöriges Flag auf TRUE gesetzt (Timer-Out-Bedingung), z.B.:

\$TIMER\_FLAG[ 4 ] = TRUE

Beim Steuerungshochlauf werden alle Timervariablen mit 0, die Flags \$TIMER\_FLAG[ 1 ]...\$TIMER\_FLAG[ 16 ] mit FALSE und die Variablen \$TIMER\_STOP[ 1 ]...\$TIMER\_STOP[ 16 ] mit TRUE vorbesetzt.

Die Einheit der Timervariablen ist Millisekunden (ms). Die Aktualisierung von \$TIMER[ 1 ]...\$TIMER[ 16 ] sowie \$TIMER\_FLAG[ 1 ]...\$TIMER\_FLAG[ 16 ] erfolgt im 12ms-Takt.

Flags	Die 1024 Flags <code>\$FLAG[ 1 ]...\$FLAG[ 1024 ]</code> werden als globale Merker eingesetzt. Diese boolschen Variablen werden mit <code>FALSE</code> vorbesetzt. Sie können sich den aktuellen Wert der Flags jederzeit auf der Bedienoberfläche unter dem Menüpunkt "Anzeige" ansehen.
Zyklische Flags	<p>Des weiteren stehen in der KR C1 32 zyklische Flags <code>\$CYCFLAG[ 1 ]...\$CYCFLAG[ 32 ]</code> zur Verfügung. Nach dem Steuerungshochlauf sind sie alle mit <code>FALSE</code> vorbelegt.</p> <p>Die Flags sind nur auf der Roboterebene zyklisch aktiv. In einer Submit-Datei sind die zyklischen Flags zwar zulässig, es erfolgt aber keine zyklische Auswertung.</p> <p>Zyklische Flags können auch in Unterprogrammen, Funktionen und Interrupt- Unterprogrammen definiert und aktiviert werden.</p> <p><code>\$CYCFLAG[ 1 ]...\$CYCFLAG[ 32 ]</code> haben den Datentyp <code>BOOL</code>. Bei einer Zuweisung an ein zyklisches Flag kann ein beliebiger boolscher Ausdruck verwendet werden.</p> <p>Zulässig sind</p> <ul style="list-style-type: none"> <li>▪ boolsche Systemvariablen und</li> <li>▪ boolsche Variablen, welche in einer Datenliste deklariert und initialisiert wurden.</li> </ul> <p>Nicht zulässig sind</p> <ul style="list-style-type: none"> <li>▪ Funktionen, die einen boolschen Wert zurückliefern.</li> </ul> <p>Die Anweisung</p> <pre>\$CYCFLAG[ 10 ] = \$IN[ 2 ] AND \$IN[ 13 ]</pre> <p>bewirkt beispielsweise, daß der boolsche Ausdruck "<code>\$IN[ 2 ] AND \$IN[ 13 ]</code>" zyklisch ausgewertet wird. Das heißt, sobald sich Eingang 2 oder Eingang 13 ändert, ändert sich auch <code>\$CYCFLAG[ 10 ]</code>, egal an welcher Stelle sich der Programmlaufzeiger nach Abarbeitung des obigen Ausdrucks befindet.</p> <p>Alle definierten zyklischen Flags bleiben solange gültig, bis ein Modul abgewählt oder mit Reset eine Satzanwahl durchgeführt wird. Wird das Programmende erreicht, bleiben alle zyklischen Flags weiterhin aktiv.</p>



Weitere Informationen finden Sie im Kapitel **[Interruptbehandlung]**, Abschnitt **[Verwendung zyklischer Flags]**.

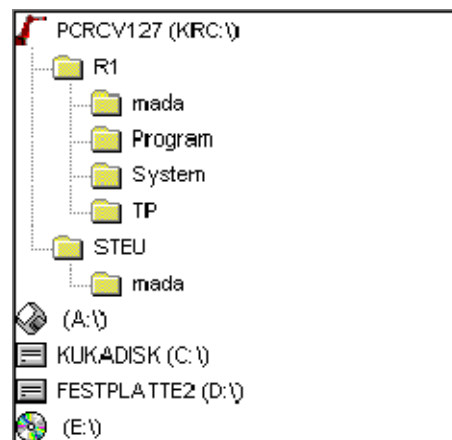
\$ – Zeichen

Allgemein sind die Namen der vordefinierten Variablen so gewählt, daß man sie sich leicht einprägen kann. Sie beginnen alle mit einem \$-Zeichen und bestehen dann aus einer sinnvollen englischen Abkürzung. Da sie wie gewöhnliche Variablen behandelt werden, müssen Sie sich keine außergewöhnlichen Befehle oder ausgefallene Optionen merken.

**Um Verwechslungen zu vermeiden, sollten Sie selbst keine Variablen vereinbaren, die mit einem \$-Zeichen beginnen.**

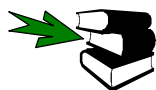
Ein Teil der vordefinierten Variablen bezieht sich auf die gesamte Steuerung KR C1 (z.B. `$ALARM_STOP` für die Definition des Ausgangs für das Not-Aus-Signal zur SPS). Andere sind dagegen nur für den Roboter von Bedeutung (z.B. `$BASE` für das Basiskoordinatensystem).

Auf der KUKA-Bof erscheint das Roboterlaufwerk auf dem die steuerungsrelevanten Daten im Verzeichnis "Steu" und die roboterrelevanten Daten im Verzeichnis "R1" abgelegt sind.



**Abb. 7** Verschiedene Ebenen auf der KUKA-Bedienoberfläche

Bei der Programmierung der KR C1 können Sie Programmdateien und Datenlisten erstellen. In den Programmdateien stehen Datendefinitionen und ausführbare Anweisungen, die Datenlisten enthalten nur Datendefinitionen und eventuell Initialisierungen.



Weitere Informationen finden Sie im Kapitel **[Datenlisten]**.

Neben den Datenlisten, die Sie bei der Programmierung erstellen, gibt es auf der KR C1 noch Datenlisten, die von KUKA definiert sind und die mit der Steuerungssoftware ausgeliefert werden. Diese Datenlisten heißen vordefinierte Datenlisten und beinhalten hauptsächlich die vordefinierten Variablen.

Die vordefinierten Datenlisten können Sie weder löschen noch selbst erzeugen. Sie werden bei der Installation der Software erzeugt und sind dann immer vorhanden. Auch die Namen der vordefinierten Datenlisten beginnen wie die Namen der vordefinierte Daten mit einem \$-Zeichen.

Auf der KR C1 gibt es folgende vordefinierte Datenlisten:

- **\$MACHINE.DAT**  
ist eine vordefinierte Datenliste mit ausschließlich vordefinierten Systemvariablen. Mit den Maschinendaten wird die Steuerung an den angeschlossenen Roboter angepaßt (Kinematik, Regelparameter, etc.). Es gibt sowohl eine \$MACHINE.DAT auf dem Steuerungssystem als auch im Robotersystem. Sie können keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Beispiele:

\$ALARM_STOP	Signal für Not-Aus (steuerungsspezifisch)
\$NUM_AXT	Anzahl der Roboterachsen (roboterspezifisch)

- **\$CUSTOM.DAT**  
ist eine Datenliste, die es nur auf dem Steuerungssystem gibt. Sie beinhaltet Daten, mit denen Sie bestimmte Steuerungsfunktionen projektieren oder parametrieren können. Für den Programmierer besteht nur die Möglichkeit, die Werte der vordefinierten Variablen zu ändern. Sie können keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Beispiele:

\$PSER_1	Protokollparameter der seriellen Schnittstelle 1
\$IBUS_ON	Einschalten alternativer Interbusgruppen

#### ▪ **\$CONFIG.DAT**

ist eine von KUKA vordefinierte Datenliste, die aber keine vordefinierten Systemvariablen enthält. Dabei existiert eine \$CONFIG.DAT auf Steuerungsebene und auf Roboterebene. Es können darin Variablen, Strukturen, Kanäle und Signale definiert werden, die längere Zeit gültig sind und für viele Programme von übergeordneter Bedeutung sind.

Die Datenliste auf Roboterebene ist in folgende Blöcke untergliedert:

- BAS
- AUTOEXT
- GRIPPER
- PERCEPT
- SPOT
- A10
- A50
- A20
- TOUCHSENSE
- USER

Globale Deklarationen des Anwenders sollten unbedingt in den USER-Block eingetragen werden. Denn nur hier werden die Vereinbarungen bei einem späteren Software-Update übernommen.

#### ▪ **\$ROBCOR.DAT**

Die Datei \$ROBCOR.DAT enthält roboterspezifische Daten für das Dynamikmodell des Roboters. Diese Daten werden für die Bahnplanung benötigt. Auch hier können Sie keine neuen Variablen erstellen oder vorhandene Variablen löschen.

Tab. 12 gibt eine Zusammenstellung der vordefinierten Datenlisten.

Datenliste	System		Wertzuweisung	
Datenliste	Steuerung	Roboter	bei	durch
<b>\$MACHINE.DAT</b>	✓	✓	Inbetriebnahme	KUKA/Anwender
<b>\$CUSTOM.DAT</b>	✓		Inbetriebnahme	Anwender/KUKA
<b>\$CONFIG.DAT</b>	✓	✓	Zellenauf- oder -umbau	Anwender/KUKA
<b>\$ROBCOR.DAT</b>		✓	Lieferung	KUKA

**Tab. 12** Vordefinierte Datenlisten auf der KR C1



#### NOTIZEN:





## 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.

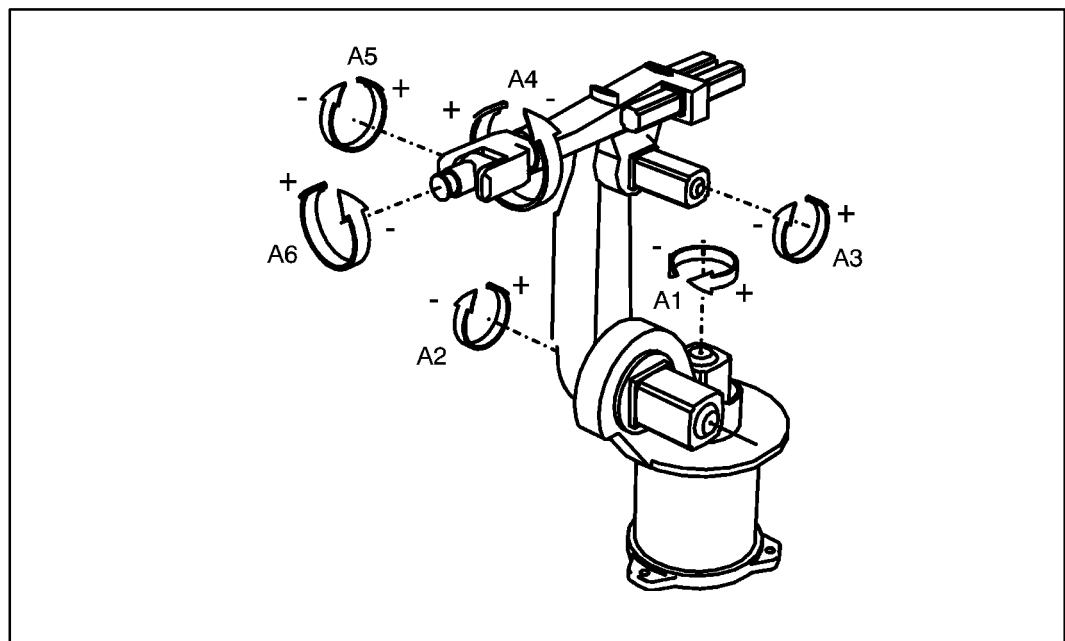
### 4.1 Verwendung verschiedener Koordinatensysteme

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:

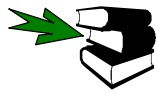
achsspezi-  
fisch

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.

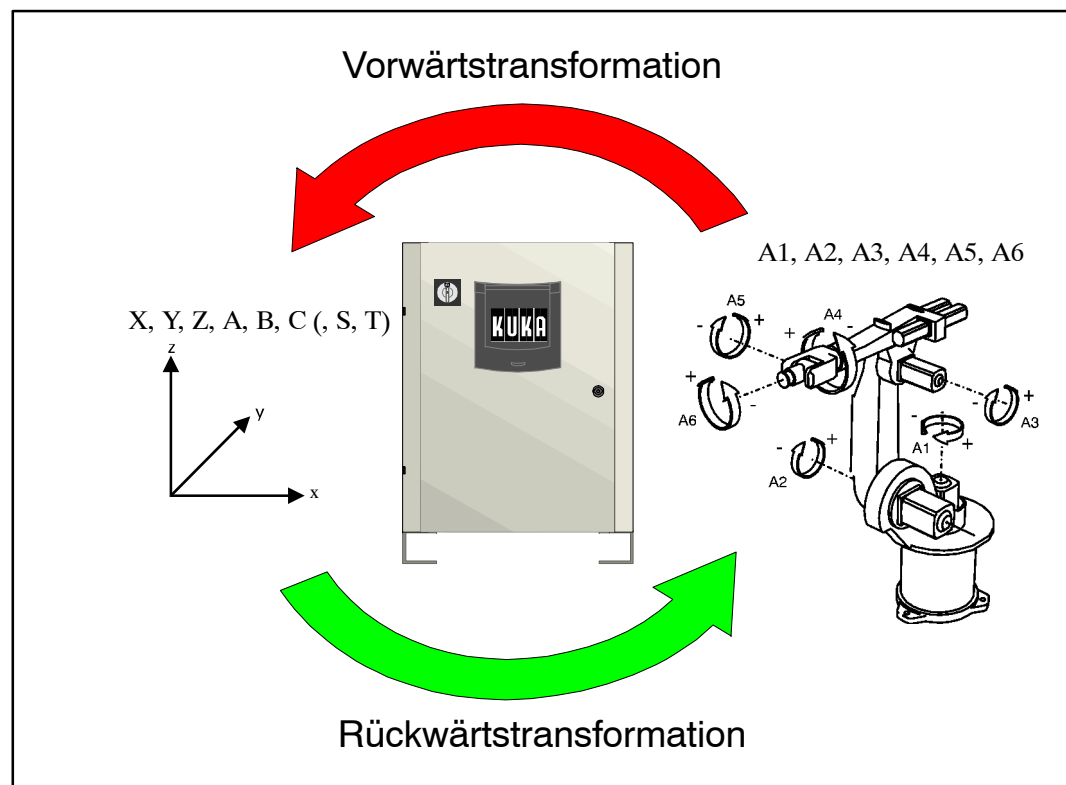


Weitere Informationen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Strukturen]**.

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).

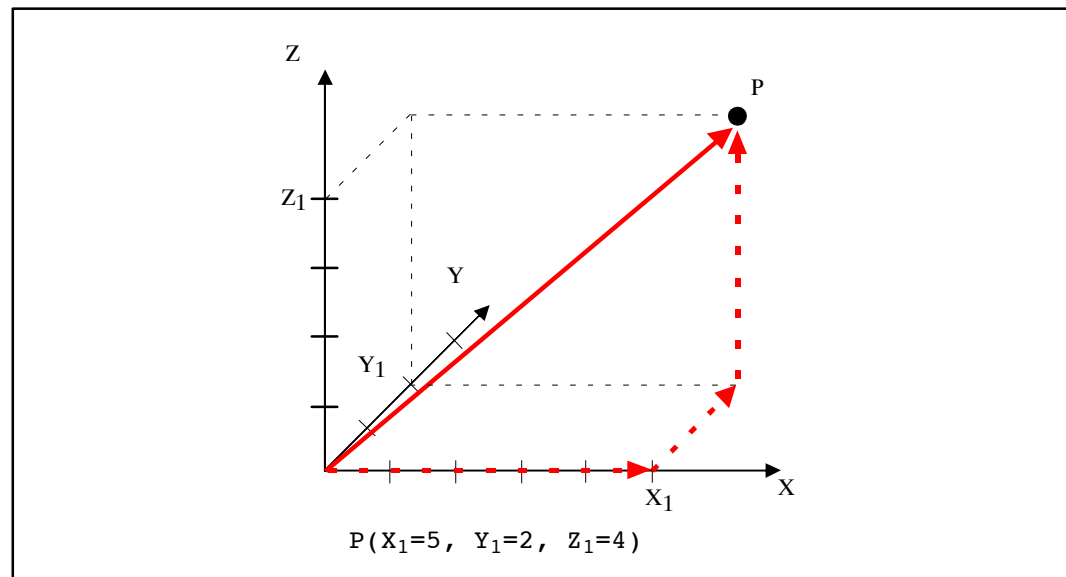


**Abb. 9** Koordinatentransformation

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).



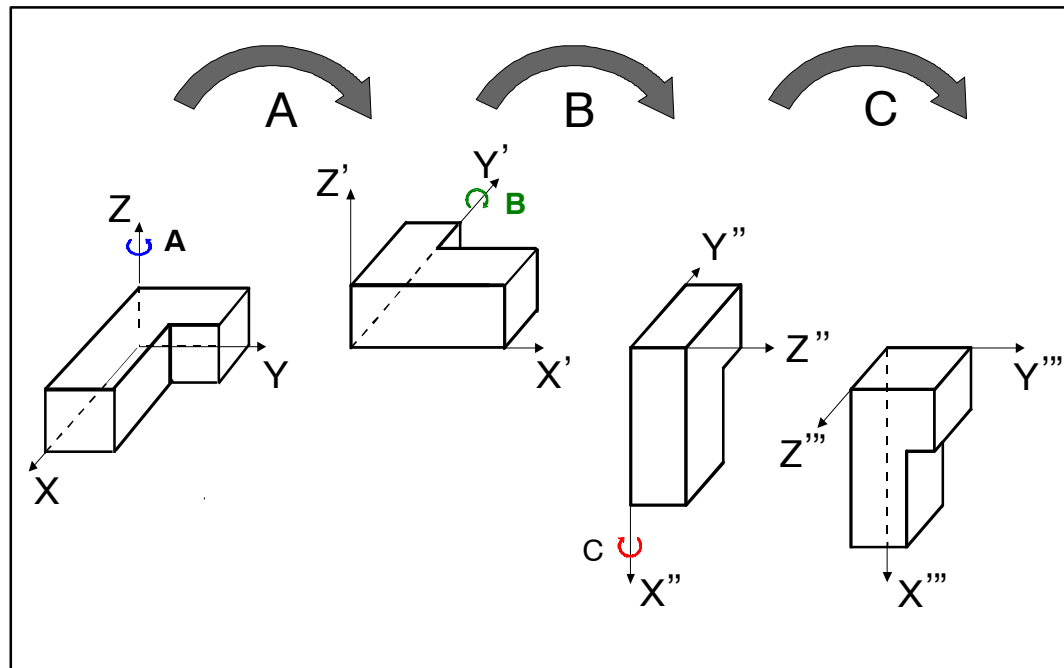
**Abb. 10** Translatorische Beschreibung der Lage eines Punktes

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:

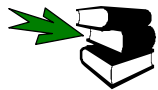
1. Drehung um die  $Z$ -Achse um den Winkel A
2. Drehung um die neue  $Y$ -Achse um den Winkel B
3. Drehung um die neue  $X$ -Achse um den Winkel C

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).



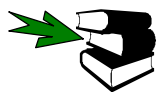
**Abb. 11** Rotatorische Beschreibung der Orientierung eines Punktes

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

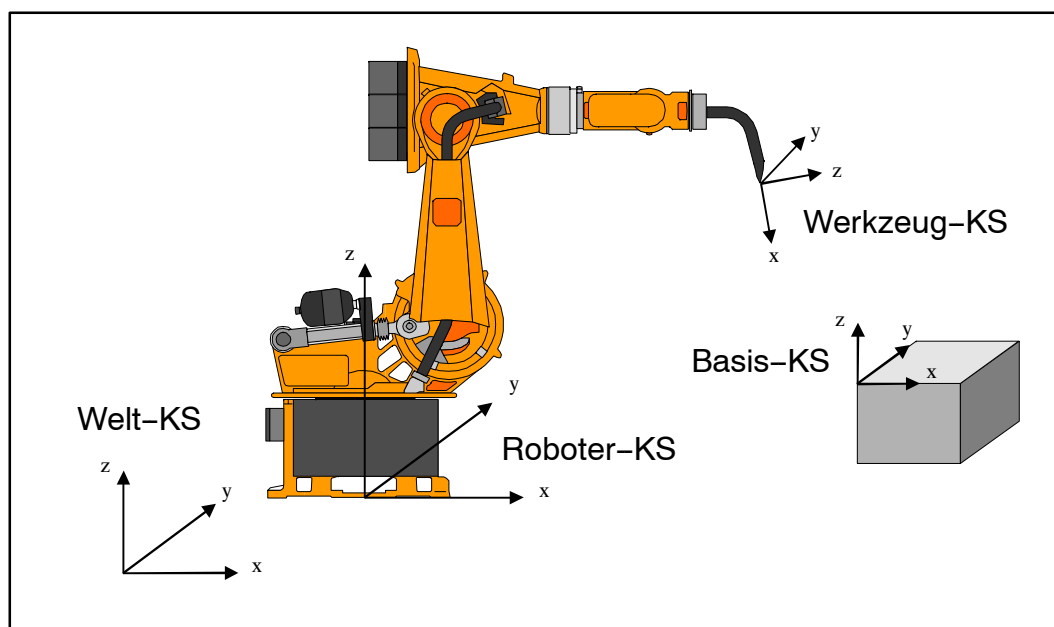


Abb. 12 Kartesische Koordinatensysteme bei Robotern

### 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.

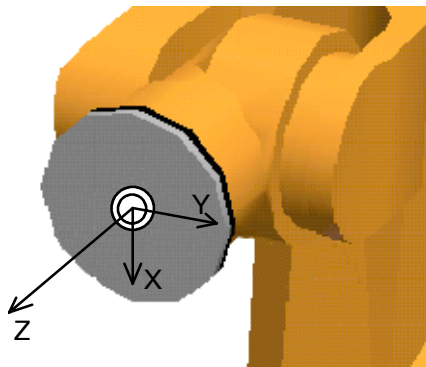
### Roboterkoordinatensystem

Dieses Koordinatensystem befindet sich im Roboterfuß und dient als Bezugskordinatensystem 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.

### 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).

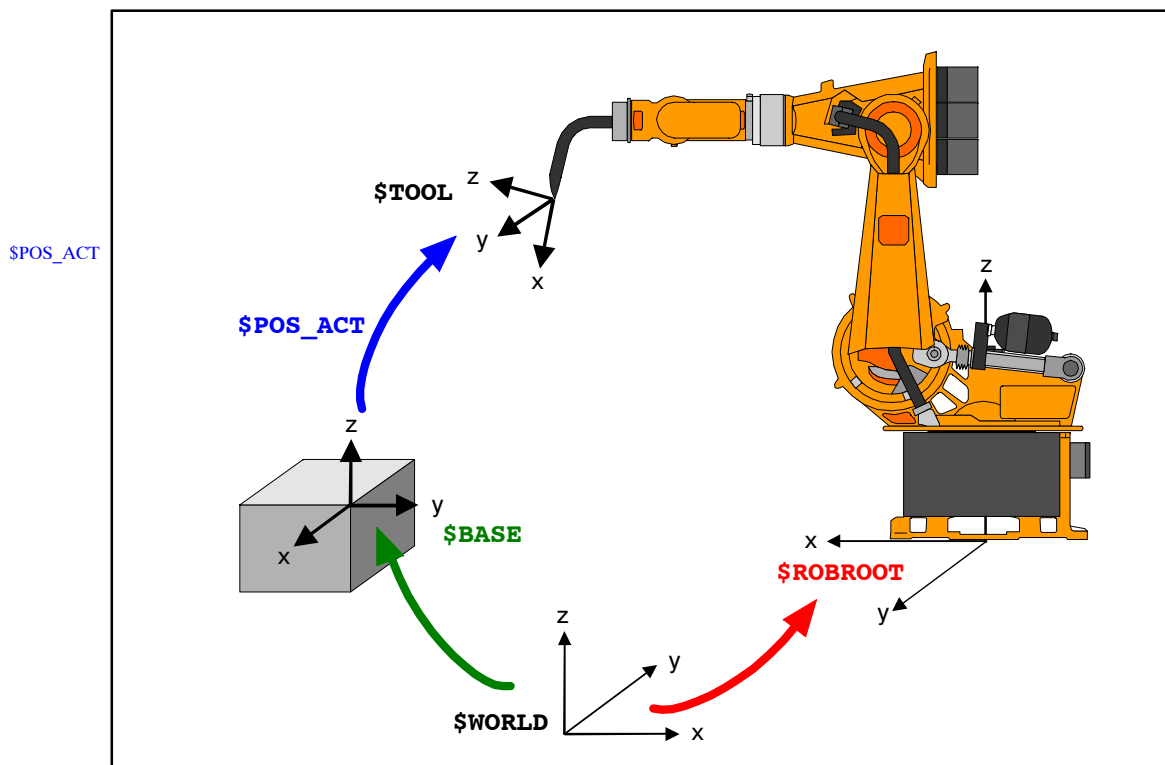


Abb. 13 Kinematische Kette bei basisbezogener Interpolation

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

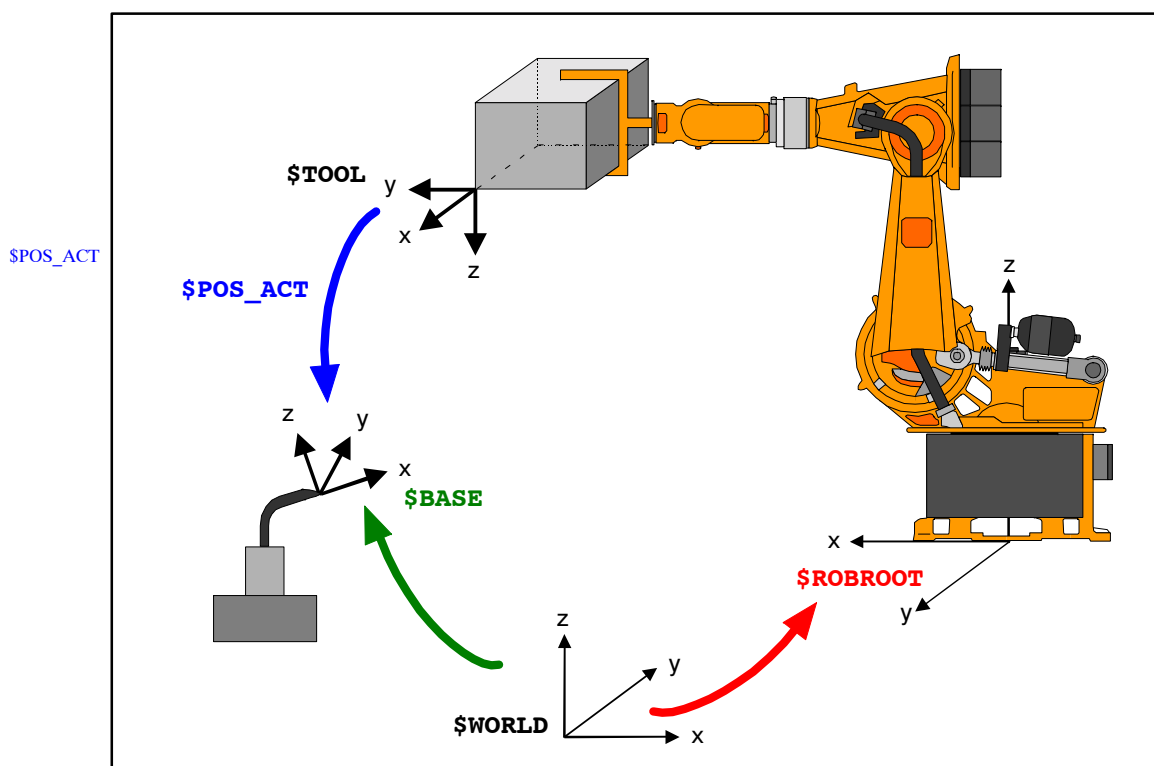
Da nun Werkstück und Werkzeug ihre Position getauscht haben, die Bewegung aber weiter bezüglich des Werkstücks erfolgen soll, muß die Interpolation der Bewegungsbahn jetzt über das **\$TOOL**-Koordinatensystem erfolgen. Diese Zuordnung der Interpolationsart erfolgt implizit beim Verwenden eines normalen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

```
$IPO_MODE = #TCP
```

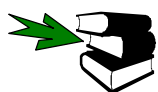
ermöglicht eine greiferbezogene Interpolation im **\$TOOL**-Koordinatensystem. Die aktuelle Position **\$POS\_ACT** wird nun also bezüglich **\$TOOL** berechnet (s. Abb. 14). Mit

```
$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

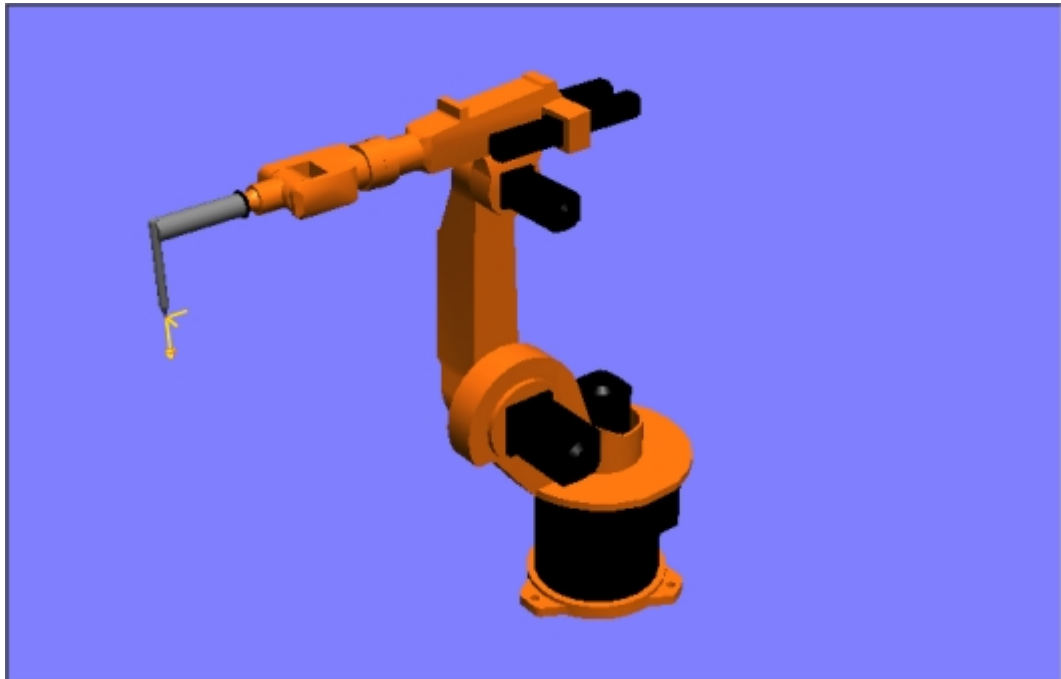
PTP {AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

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

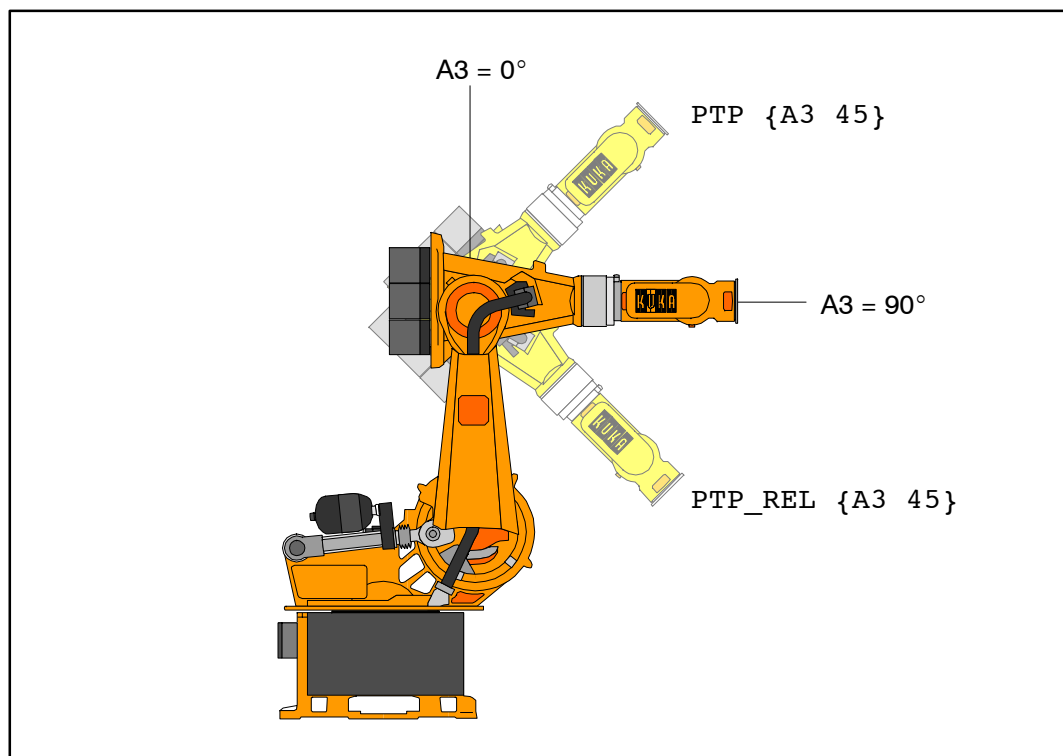
Werden bei der Angabe der Achskoordinaten einzelne Komponenten weggelassen, so verfährt der Roboter nur die angegebenen Achsen, die anderen verändern ihre Lage nicht. Mit

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.

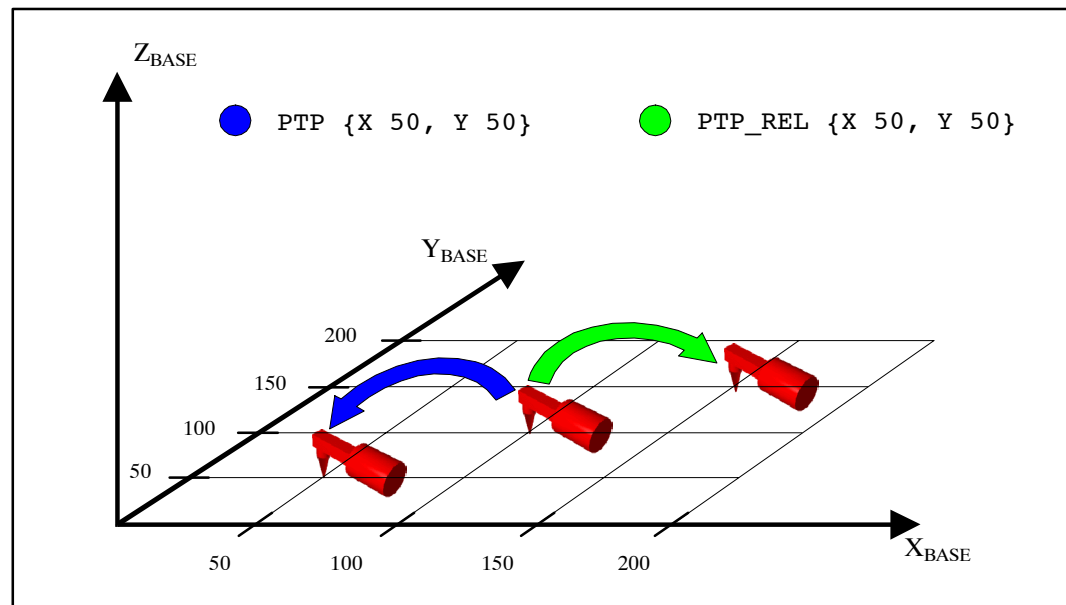
Auch bei Zielpunktangabe in kartesischen Koordinaten können wieder einzelne Komponenten der Geometrieangabe weggelassen werden. Die Anweisung

```
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.



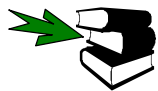
Die Basisverschiebung mit Hilfe des Doppelpunkt-Operators hat außerdem einen entscheidenden Vorteil gegenüber einer Neubesetzung von `$BASE`:

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 Neuanswahl 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
```

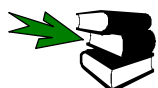


In diesem Beispiel werden außerdem die notwendigen Belegungen der Geschwindigkeiten, Beschleunigungen sowie \$BASE- und \$TOOL-Koordinatensystemen nicht mehr "von Hand" vorgenommen. Stattdessen wird das standardmäßig vorhandene "BAS.SRC" hierzu verwendet. Dazu muß es zunächst mit der EXT-Anweisung dem 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  $< 0^\circ$

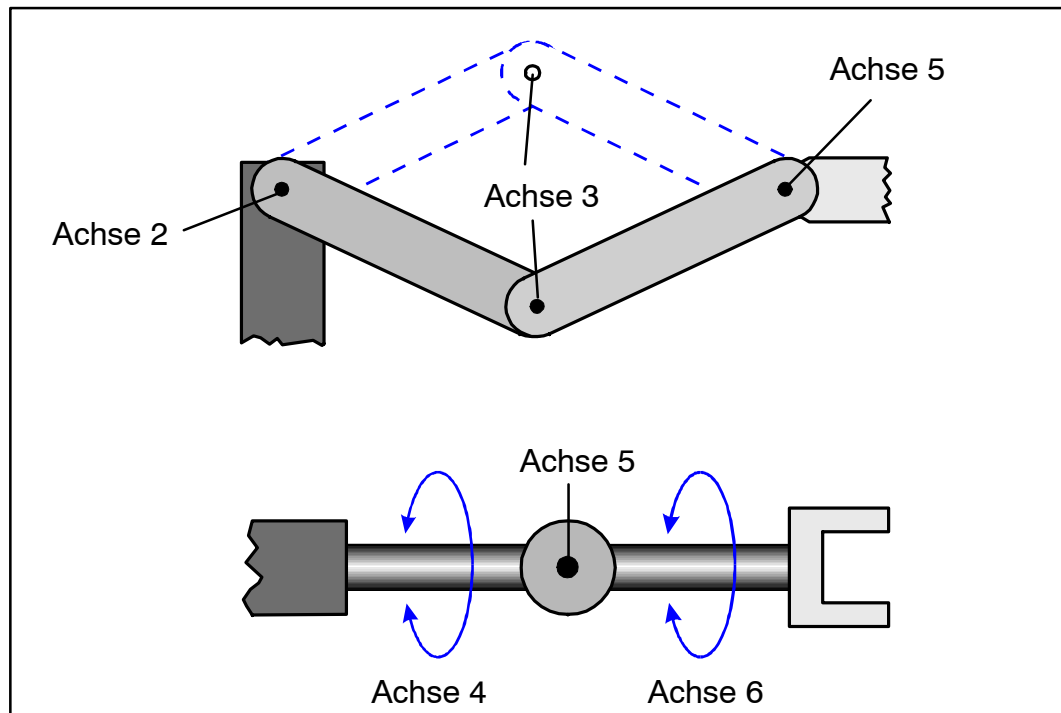
Wert	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	$A6 \geq 0^\circ$	$A5 \geq 0^\circ$	$A4 \geq 0^\circ$	$A3 \geq 0^\circ$	$A2 \geq 0^\circ$	$A1 \geq 0^\circ$
1	$A6 < 0^\circ$	$A5 < 0^\circ$	$A4 < 0^\circ$	$A3 < 0^\circ$	$A2 < 0^\circ$	$A1 < 0^\circ$

**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

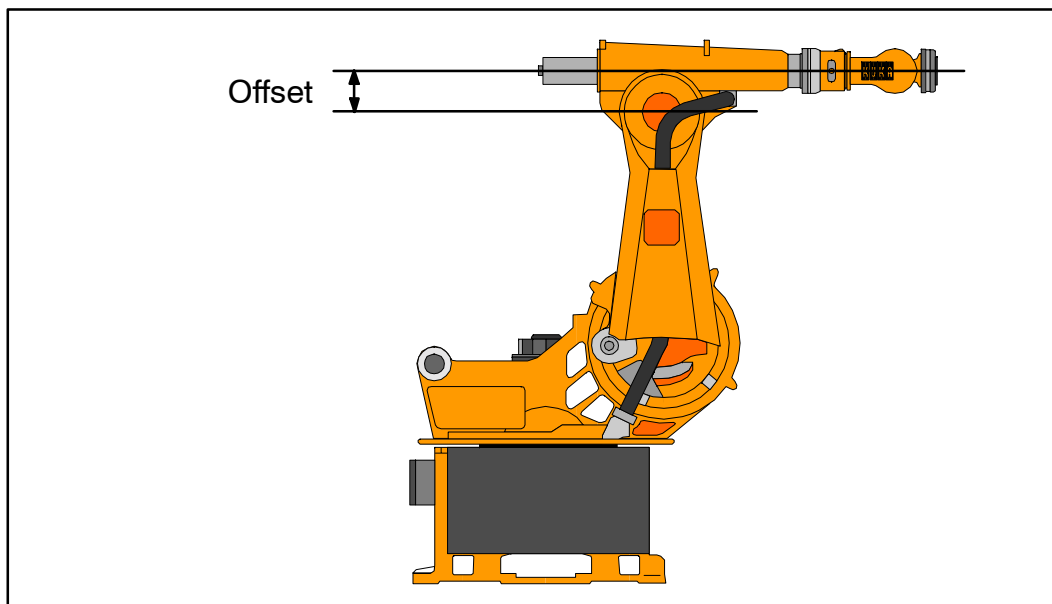
A1-Koordinatensystem: Steht die Achse 1 auf  $0^\circ$ , ist es mit dem \$ROBROOT-Koordinatensystem identisch. Bei Werten ungleich  $0^\circ$  wird es mit der Achse 1 mitbewegt.

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.



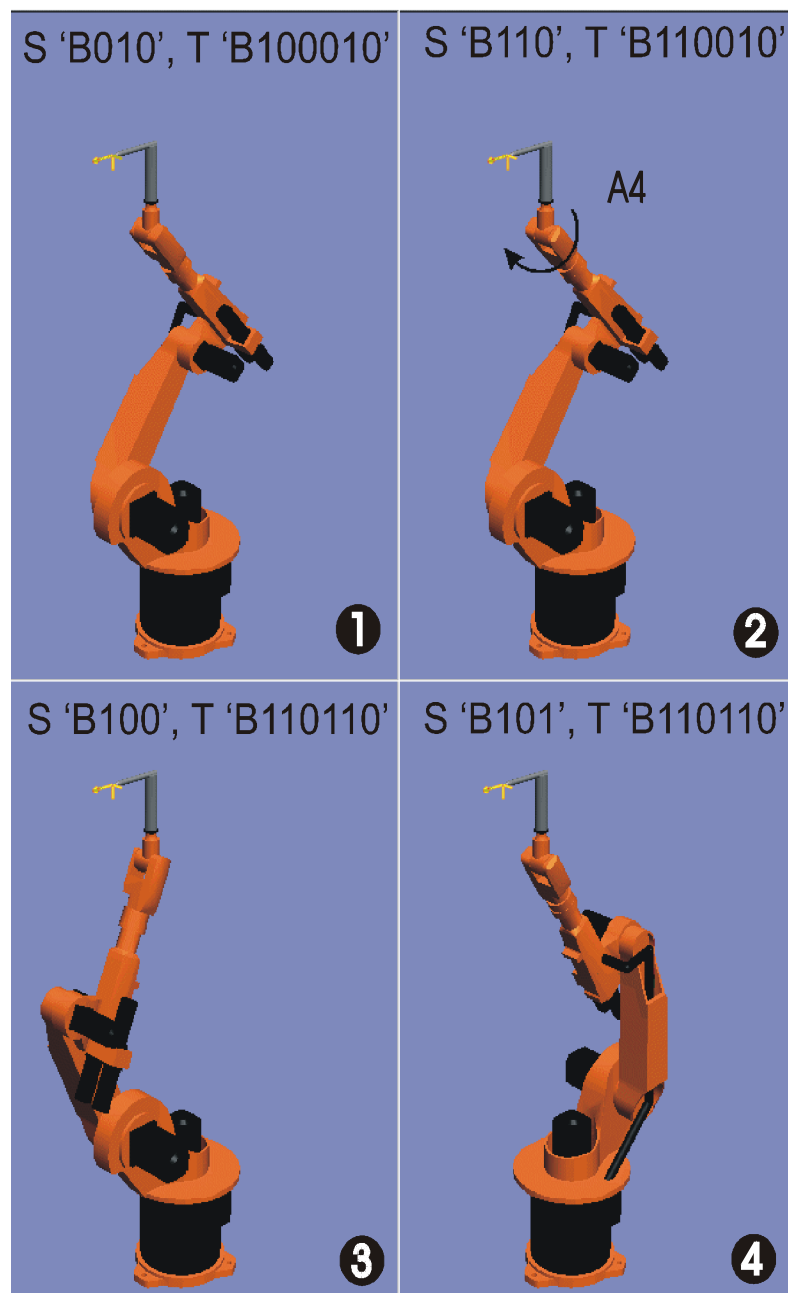
**Abb. 19** Offset zwischen Achse 3 und 4 bei einem KR 30

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.

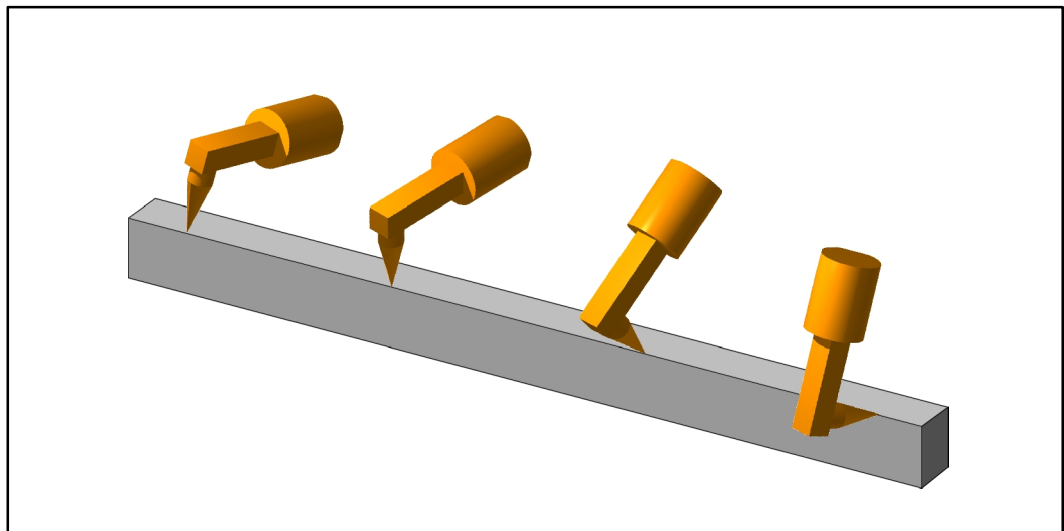
Es besteht die Möglichkeit über die Systemvariable \$CP\_VEL\_TYPE die Achsvorschübe und somit die Beschleunigung und die Geschwindigkeit zu reduzieren, um ein Ansprechen der Überwachungsgrenzen (Stoppreaktion) zu vermeiden. Als Defaulteinstellung ist die Variable mit #CONSTANT belegt, d.h. die Reduzierung 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.



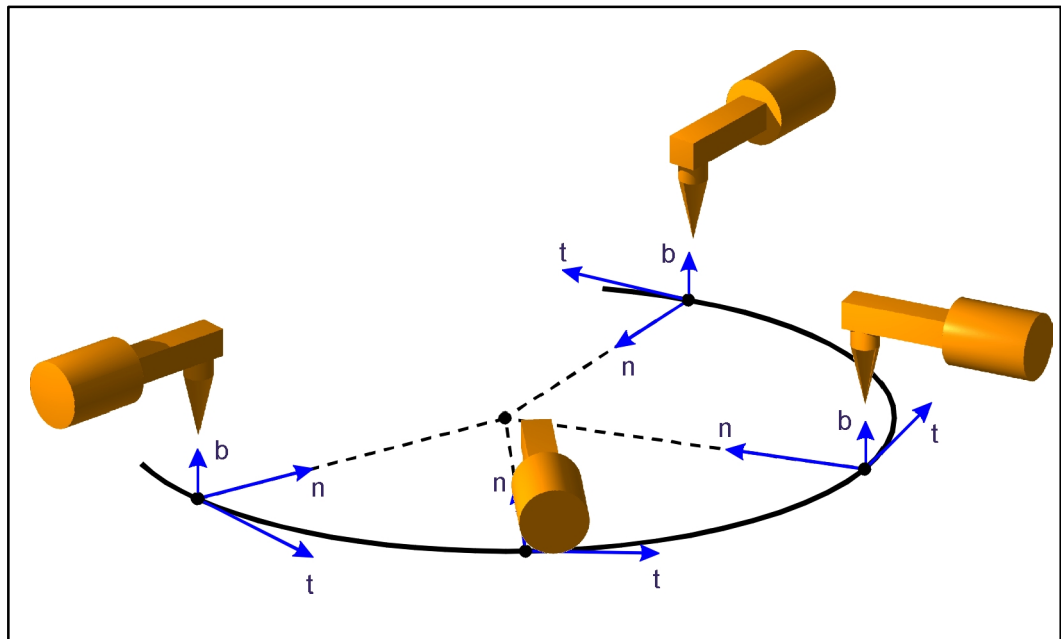
**Abb. 21** Orientierungsänderung bei einer Linearbewegung

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).



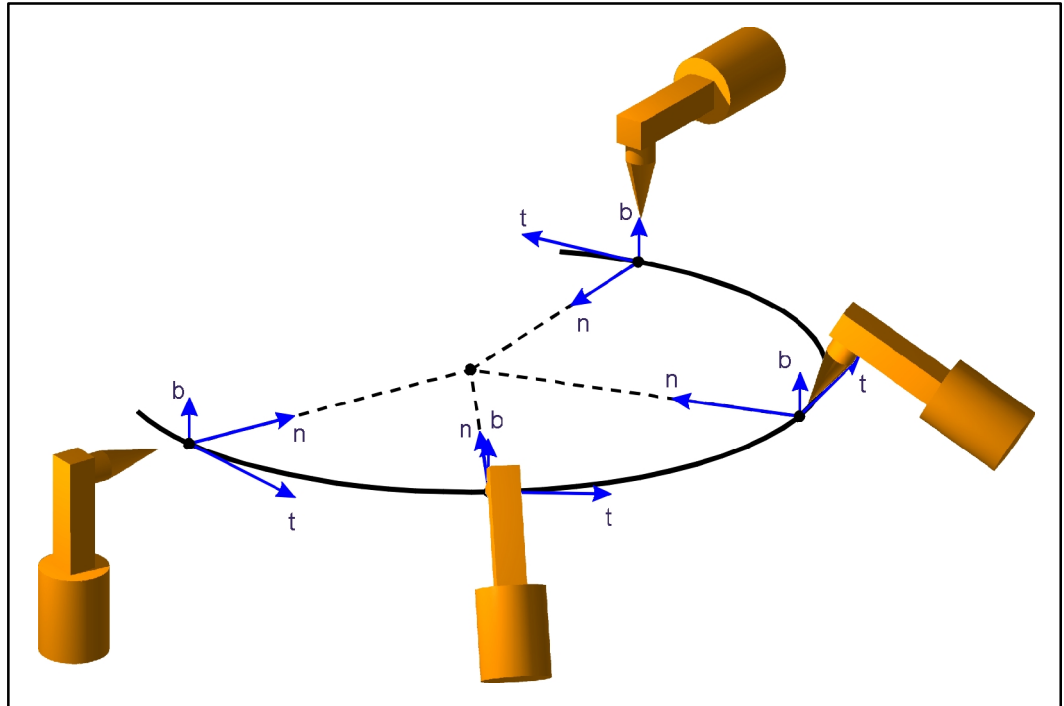
**Abb. 22** Konstante bahnbezogene Orientierungsführung bei Kreisbewegungen

Das bahnbegleitende Dreibein setzt sich aus dem Kreistangentenvektor  $\mathbf{t}$ , dem Normalenvektor  $\mathbf{n}$  und dem Binormalenvektor  $\mathbf{b}$  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.

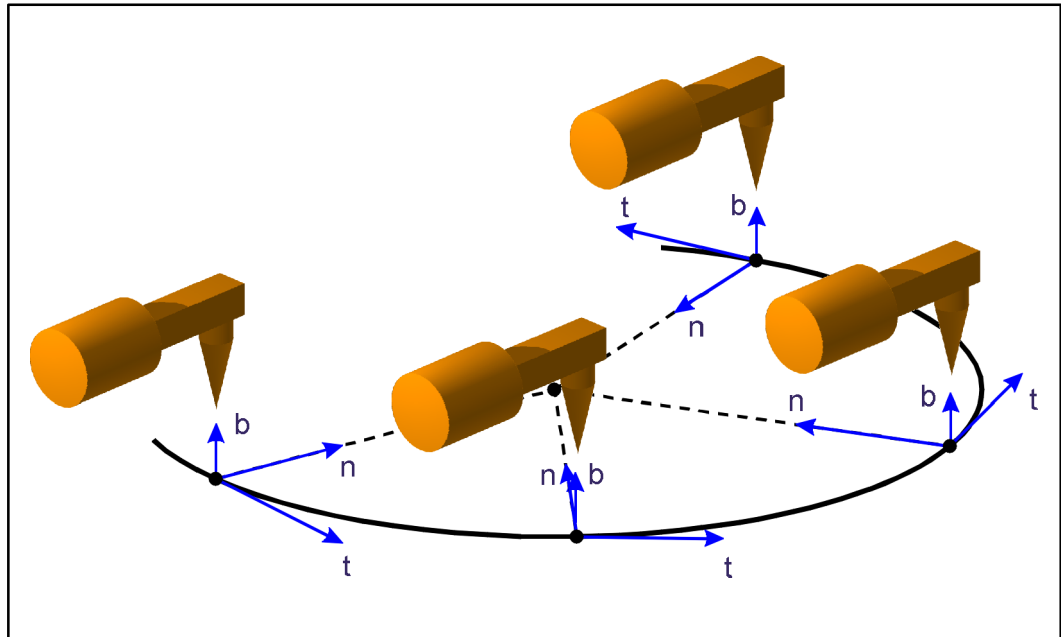


**Abb. 23** Variable bahnbezogene Orientierungsführung bei Kreisbewegungen

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).

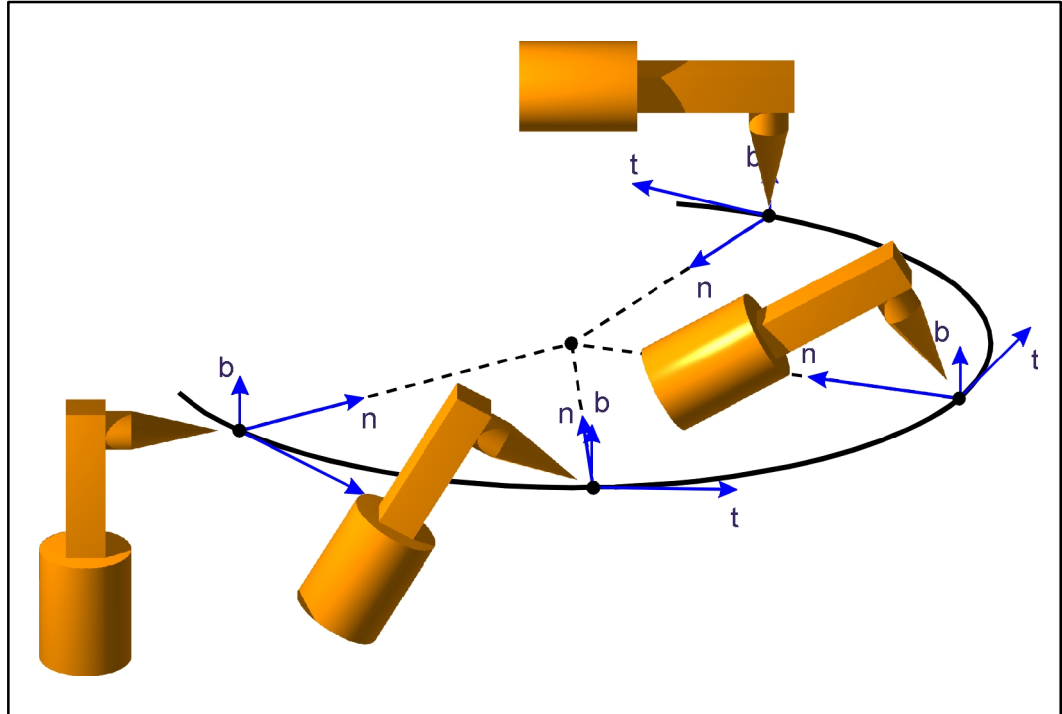


**Abb. 24** Konstante raumbezogene Orientierungsführung bei Kreisbewegungen



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 $BAS(\#INITMOV, 0)$
$\$ORI\_TYPE$	#VAR	
$\$CIRC\_TYPE$	#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 Geraden 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.

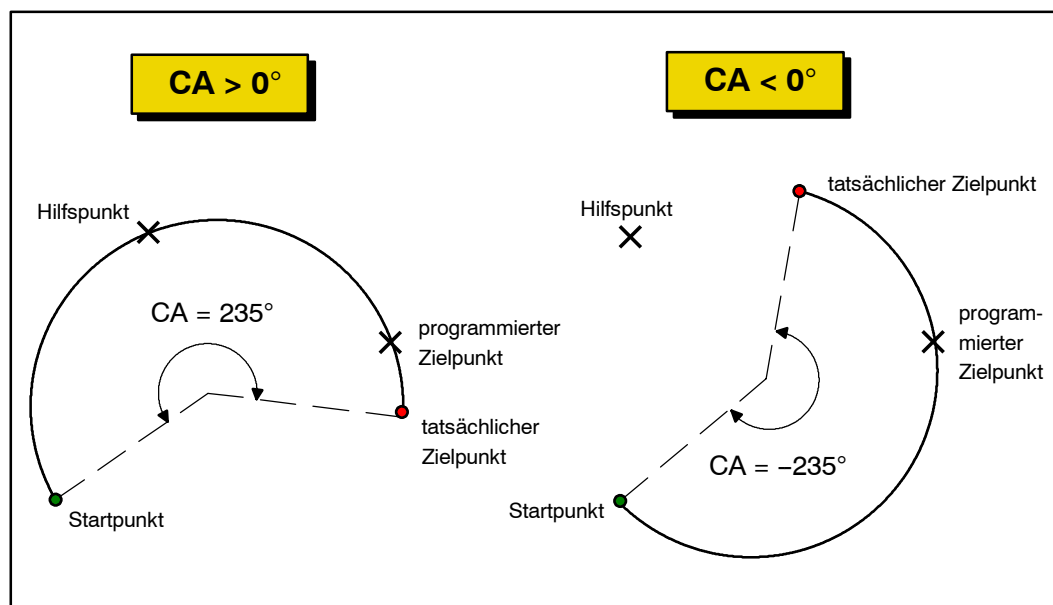
Der abzufahrende Kreisbogen kann entsprechend dem Kreiswinkel verlängert oder verkürzt werden. Die programmierte Zielorientierung wird dann im tatsächlichen Zielpunkt erreicht. Über das Vorzeichen des Kreiswinkels kann der Drehsinn, d.h. die Richtung in der der Kreisbogen abgefahren werden soll, festgelegt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

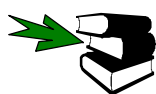
## 4.4 Rechnervorlauf

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)
<b>\$ADVANCE</b>	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.

<b>Anweisungen</b>	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			
<b>Anweisungen in Verbindung mit einem Interrupt</b>	END (falls im Modul ein nicht globaler Interrupt definiert wurde)			
	INTERRUPT DECL (falls der Interrupt bereits deklariert wurde)			
	RESUME ohne BRAKE			
<b>gebräuchliche Systemvariablen</b>	\$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

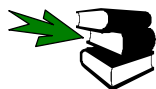
CONTINUE

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

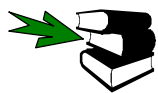
Überschleif-  
kontur

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.

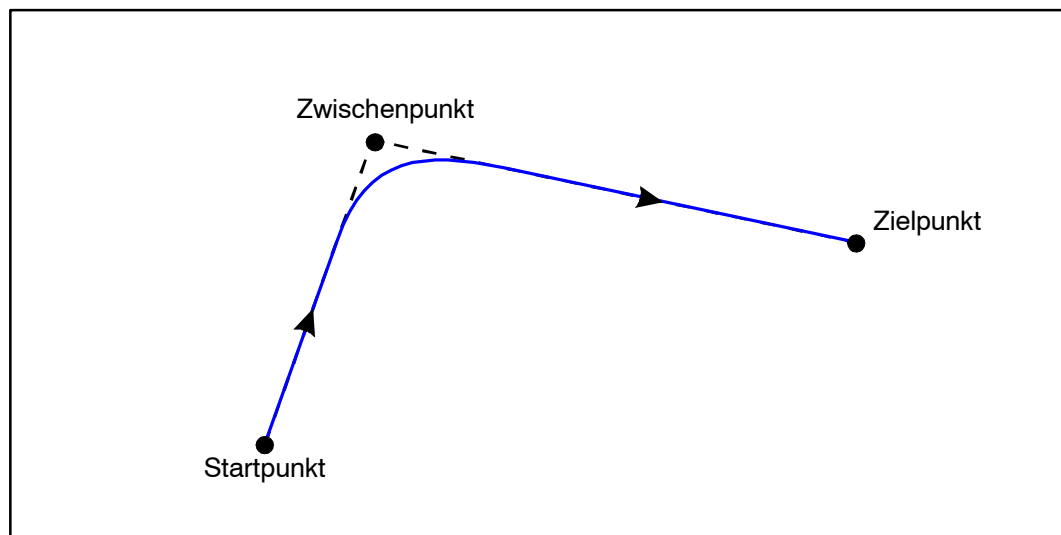
Um ein Überschleifen zu ermöglichen, muß daher der Rechnervorlauf (\$ADVANCE) mindestens auf 1 gesetzt sein. Falls der Rechnervorlauf zu klein ist, erscheint die Meldung "Überschleifen nicht möglich" und die Punkte werden genau angefahren.

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

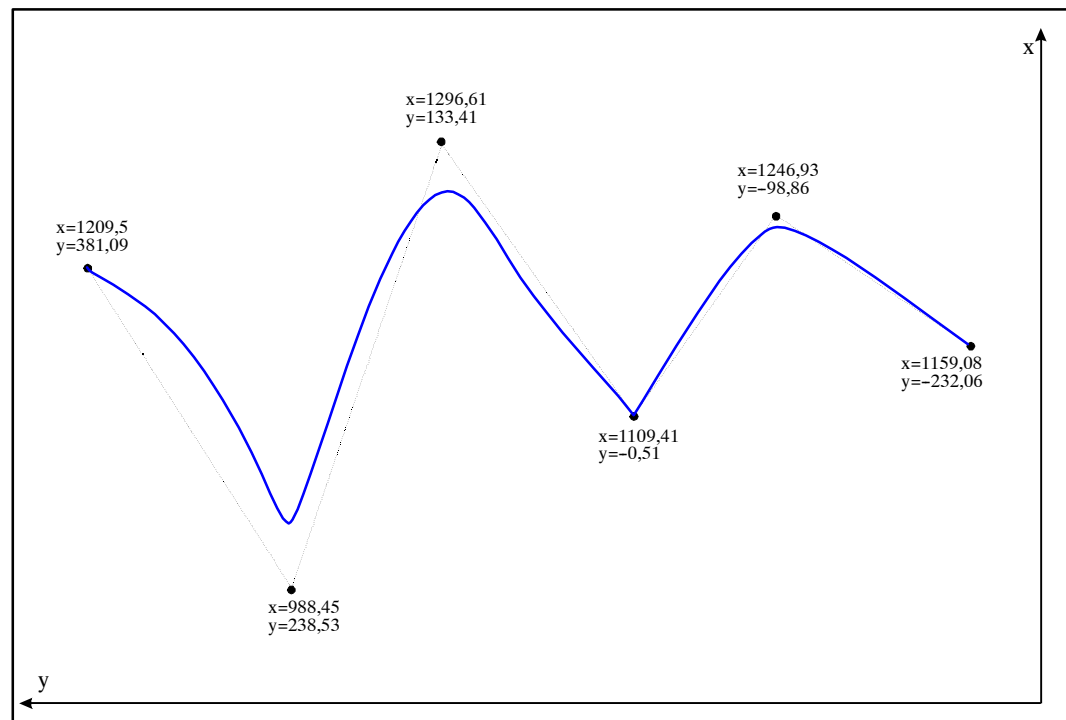
PTP {POS:X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C 70.95,S 2,T
10}

;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

```

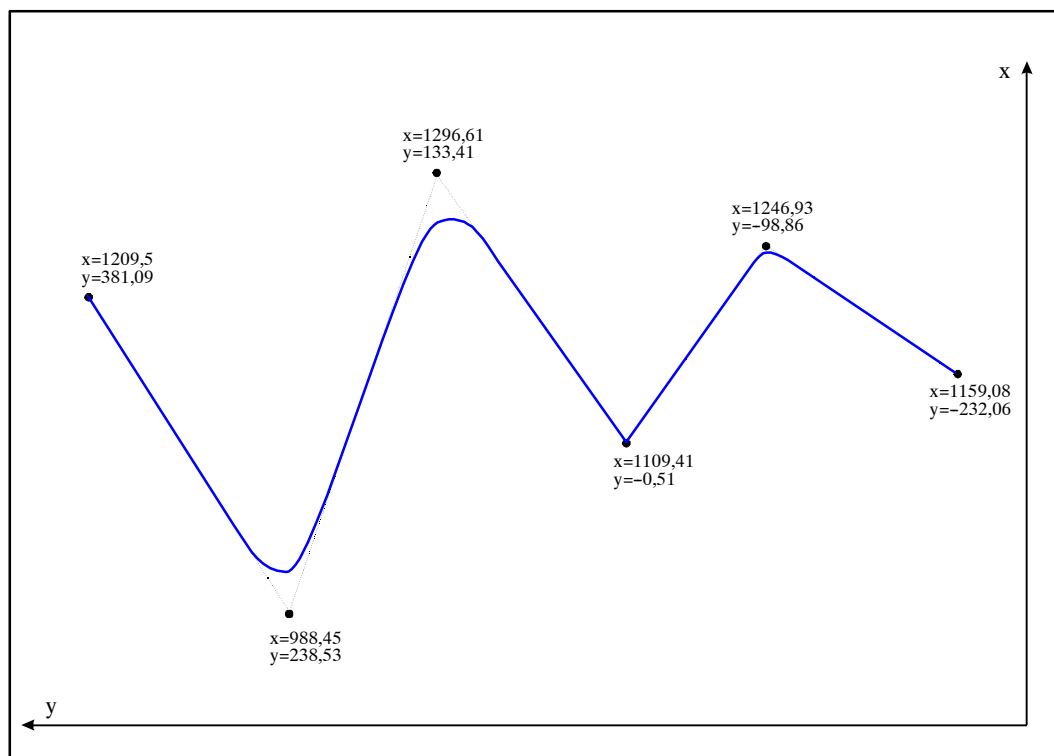


Abb. 30 Beispiel für LIN-LIN-Überschleifen



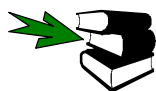
Die Position, bei der die Überschleifkontur in den nachfolgenden LIN-Satz mündet, wird von der Steuerung automatisch berechnet. Falls \$ACC und \$VEL für beide Einzelsätze identisch sind und die Satzlängen ausreichen, ist die Überschleifparabel symmetrisch zur Winkelhalbierenden zwischen den beiden Einzelsätzen.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

```

```

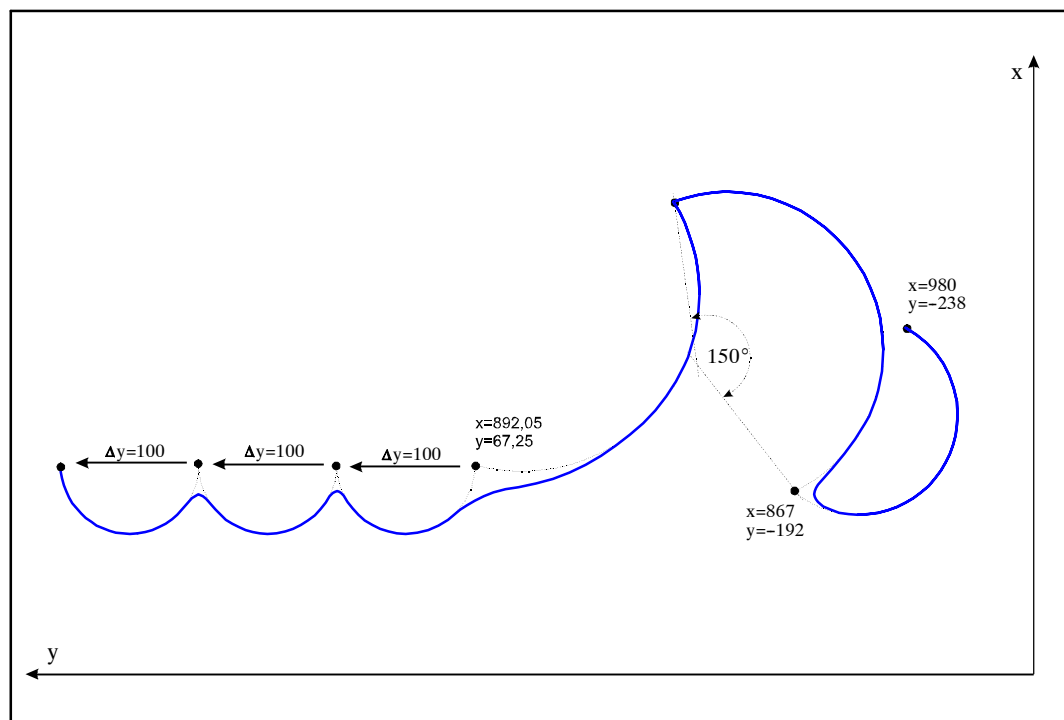
; relative Kreisbewegungen

; 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
    
```



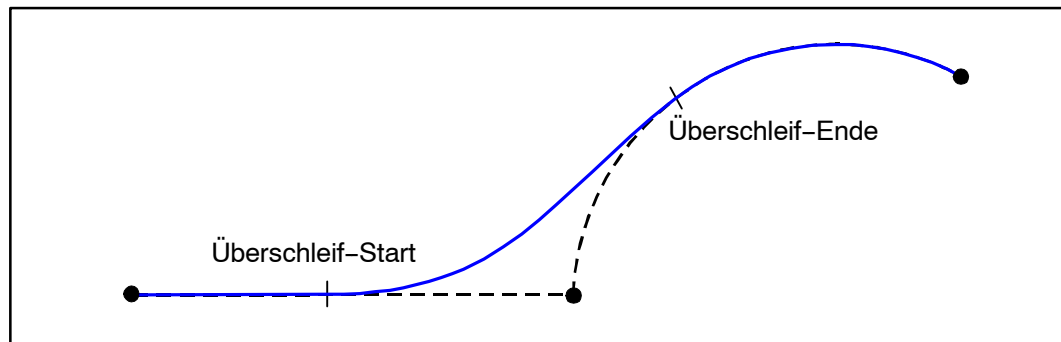
**Abb. 31** Beispiel für CIRC-CIRC-Überschleifen



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.

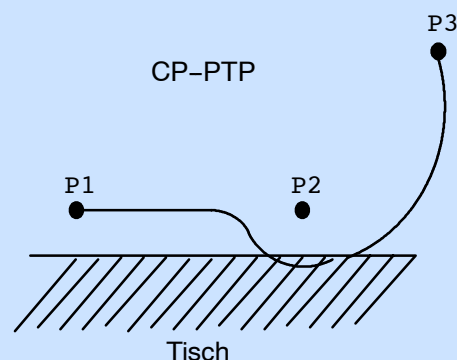
Bsp.:

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
...
```

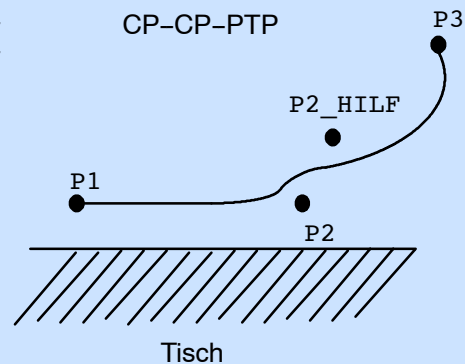
Hierbei entsteht das Problem, daß die Überschleifbewegung nicht voraussehbar ist. Eine Kollision mit dem Tisch wäre denkbar.



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}

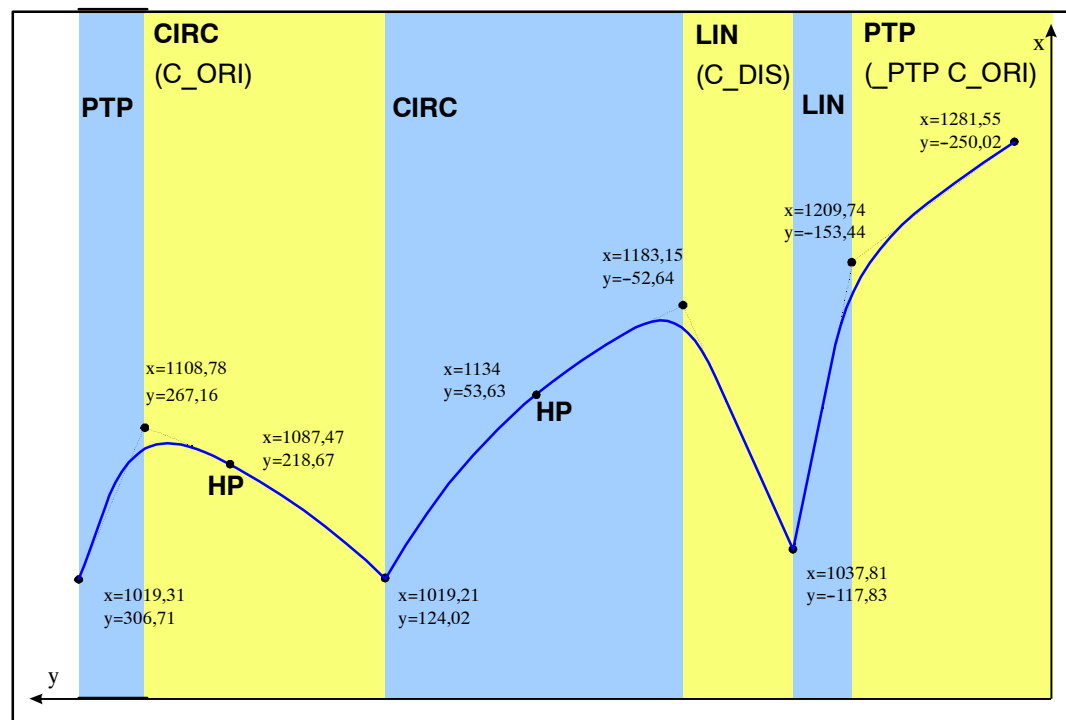
;----- Hauptteil -----
PTP  HOME ; SAK-Fahrt
PTP  {POS: X 1281.55,Y -250.02,Z 716,A 79.11,B 68.13,C 79.73,S
6,T 50}

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.

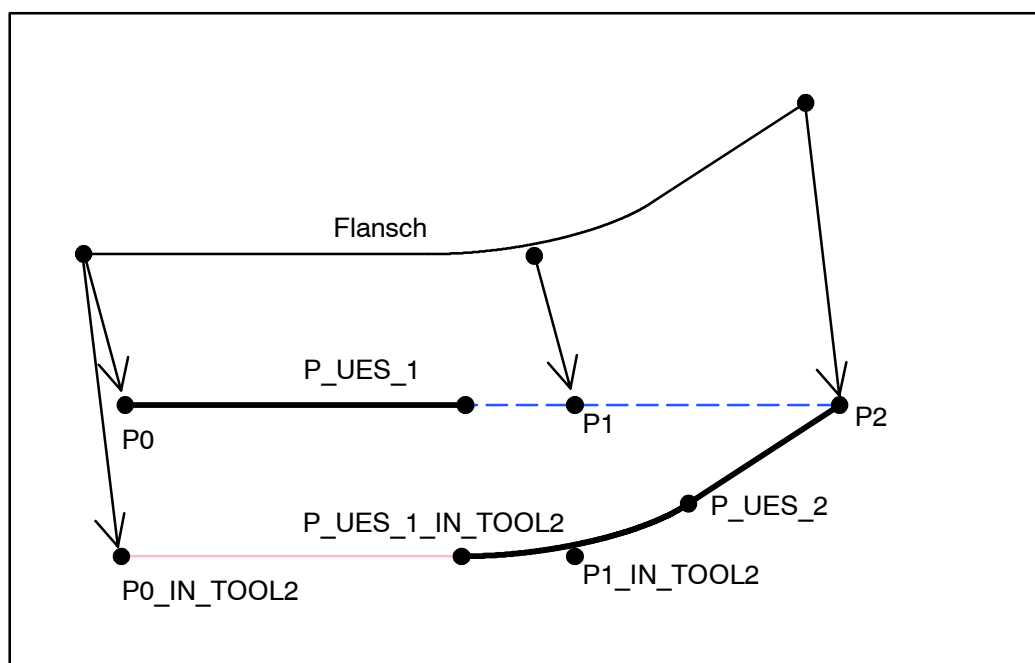
Es ist möglich ein Werkzeug auch während des Überschleifens virtuell zu wechseln. D.h. ein Werkzeug wird zweimal unterschiedlich gelernt. Z.B. ist bei einer Spritzpistole der Abstand zum Werkstück bei "TOOL1" 5cm und bei "TOOL2" 10cm.



#### 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.



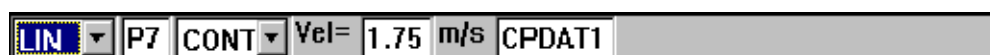
Weitere Informationen zum Thema “Datenlisten” finden Sie im Kapitel **[Datenlisten]**.



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!!

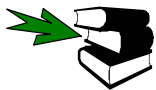
## 5 KRL-Assistent

Die KUKA-Technologiepakete enthalten die wichtigsten Funktionen für übliche Roboteranwendungen. Spezielle Funktionen, die nicht in den KUKA-Technologiepaketen enthalten sind, können vom Anwender über die direkte Programmierung des Robotersystems in "KRL", der "KUKA Robot Language" realisiert werden.

Um auch Anwendern, die diese Programmiersprache nicht oft benutzen, die effektive Programmierung spezieller Funktionen zu ermöglichen, wurde der "KRL-Assistent" integriert.

Der "KRL-Assistent" bietet seinem Benutzer syntaxunterstützte Programmierung. Nach der Auswahl des gewünschten KRL-Befehls werden befehlsbegleitende Angaben in Masken vorgegeben. Die Inhalte dieser Masken werden übernommen oder geändert. Alle Inhalte können auch zu einem späteren Zeitpunkt jederzeit geändert werden.

### Bedienung



Zum Programmieren eines Bewegungsbefehls müssen Sie ein Programm auswählen oder es in den Editor laden. Einzelheiten über die Erstellung und Änderung von Programmen entnehmen Sie bitte dem Kapitel **[Allgemeines zu KRL-Programmen]**, Abschnitt **[Programme erstellen und editieren]**.



```

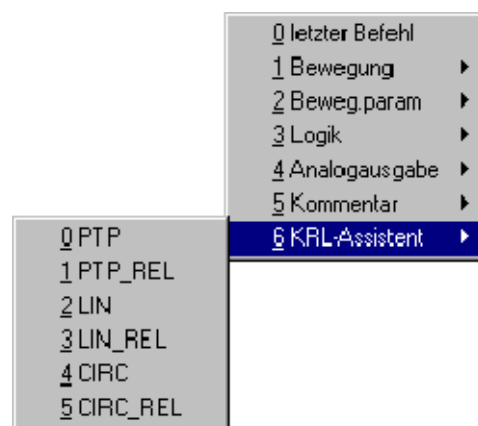
1  → INI
2  PTP HOME  Vel= 100 % DEFAULT
3  |
4  PTP HOME  Vel= 100 % DEFAULT
5  END

```

Achten Sie bitte auf die Position des Editier-Cursors. Die von Ihnen erstellte Programmzeile wird als neue Zeile unterhalb des Cursors eingefügt.

### Befehle

Öffnen Sie über die Menütaste "Befehle" das Menü. Wählen Sie "KRL-Assistent" aus. Es zeigt sich folgendes Bild:



Hier können Sie nun aus den angebotenen Bewegungsbefehlen auswählen.

## 5.1 Positionsangaben

### Der Platzhalter “!”



Das Zeichen “!” ist ein Platzhalter. Mit seiner Hilfe können Sie ein Bewegungsprogramm erstellen, ohne die genaue Lage der Punkte zu kennen, die später die Bahn des Roboters bestimmen.

Beim späteren Ablauf wird das Programm an dieser Stelle angehalten und Sie können den Punkt wie nachfolgend beschrieben aufnehmen:

Quitt

Wenn während des späteren Ablaufs des Programmes im Meldungsfenster die Mitteilung “Anweisung unzulässig” erscheint, löschen Sie diese Meldung mit dem Softkey “Quitt”.

Zeit	Nr.	Abs.	Meldung
15:42	1425		Anweisung unzulässig

Verfahren Sie das Robotersystem an die gewünschte Position.

Touch Up

Drücken Sie dort dann den Softkey “Touch Up”. Beachten Sie die Mitteilung im Meldungsfenster.

Die Übernahme der Position wird Ihnen im Meldungsfenster mitgeteilt.

Zeit	Nr.	Abs.	Meldung
15:44	0		TPEXPERT aktuelle Position “(POS: X1620, Y 0, Z 1910, A 0, B 90, C 0, S 2, T 2)” wurde übernommen

Quitt

Sie können diese Meldung durch Betätigen des Softkeys “Quitt” wieder löschen.

### Positionsangabe durch Variablen



Anstelle des Platzhalters können Sie auch einen gültigen Variablennamen einsetzen. Eine Liste der für KRL reservierten Schlüsselwörter, die Sie dabei nicht verwenden können, finden Sie im [KRL Reference Guide].



Sie sollten das Robotersystem bereits vor der Programmierung dieser Funktion in die gewünschte Position verfahren.



Informationen zur manuellen Bewegung des Roboters finden Sie im **Bedienhandbuch** in der Dokumentation **Bedienung**, Kapitel **[Handverfahren des Roboters]**.

VAR

Sollte der Name, den Sie angegeben haben, im System noch nicht bekannt sein, erscheint in der Softkey-Leiste der Softkey “VAR”. Mit ihm werden Sie aufgefordert, dem Namen ein Datenformat zuzuordnen.

Nach seiner Betätigung ändert sich die Softkey-Leiste:

<<	E6POS	POS	FRAME	E6AXIS	AXIS	!
----	-------	-----	-------	--------	------	---





Die Softkeyleiste steht nur zur Verfügung, wenn neben der ".SRC"-Datei auch eine Datenliste ".DAT" existiert.

Nach Betätigung einer der Softkeys "E6POS", "POS", "FRAME", "E6AXIS" oder "AXIS" wird die aktuelle Position des Robotersystems im ausgewählten Datenformat übernommen. Dies wird Ihnen durch eine Mitteilung im Meldungsfenster bestätigt.

Zeit	Nr.	Abs.	Meldung
16:15 0	TPBASIS		Die aktuellen Koordinaten wurde in Punkt KUKA01 übernommen.

Quitt

Sie können diese Meldung dann durch Betätigen des Softkeys "Quitt" wieder löschen.

### Positionsübernahme durch "Touch Up"



Bevor Sie in einem Programm durch "Touch Up" Positionen übernehmen können, müssen der Steuerung Daten über die Lage des gültigen Roboterkoordinatensystems, sowie gültige Werkzeug/Werkstückdaten mitgeteilt werden.

Dazu muß die INI-Sequenz am Programmkopf durchlaufen werden.



Sie sollten das Robotersystem bereits vor der Programmierung der Funktion in die gewünschte Position verfahren.

Touch Up

Drücken Sie den Softkey "Touch Up", so wird die aktuelle Position des Robotersystems übernommen.

PTP[{POS: X 1620, Y 0, Z 1910, A 0, B 90, C 0, S 2, T 2}]

Die Übernahme der Position wird durch eine Mitteilung im Meldungsfenster bestätigt.

Zeit	Nr.	Abs.	Meldung
00:01 0	TPEXPERT		aktuelle Position "{POS: X 1620, Y 0, Z 1910, A 0, B 90, C 0, S 2, T 2}" wurde übernommen.

Quitt

Sie können diese Meldung durch Betätigen des Softkeys "Quitt" wieder löschen.

### Manuelle Positionsangabe

Zusätzlich zur Möglichkeit bereits angefahrte Positionen zu übernehmen, können Sie Raumpunkte auch manuell angeben.

{?}

Drücken Sie dazu nach dem Erscheinen des Inline-Formulars den Softkey "{ ? }". Die Belegung der Softkey-Leiste ändert sich:

<<	E6POS	POS	FRAME	E6AXIS	AXIS	!
----	-------	-----	-------	--------	------	---

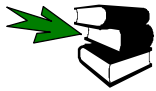
Nach Auswahl eines Datenformats durch Betätigen des entsprechenden Softkeys wird die aktuelle Position in das Inline-Formular übernommen.

```
PTP[{A1 0, A2 -90, A3 90, A4 0, A5 0, A6 0}]
```

Mit Hilfe der Editierfunktionen können Sie diese Positionsangaben nun Ihren Wünschen entsprechend abändern.

### Der geometrische Operator “:”

Mit dem geometrischen Operator “:” werden Positionsangaben der Typen POS und FRAME verknüpft. Dies ist immer dann erforderlich, wenn z.B. der Ursprung eines Koordinatensystems um einen Korrekturwert verschoben werden muß.



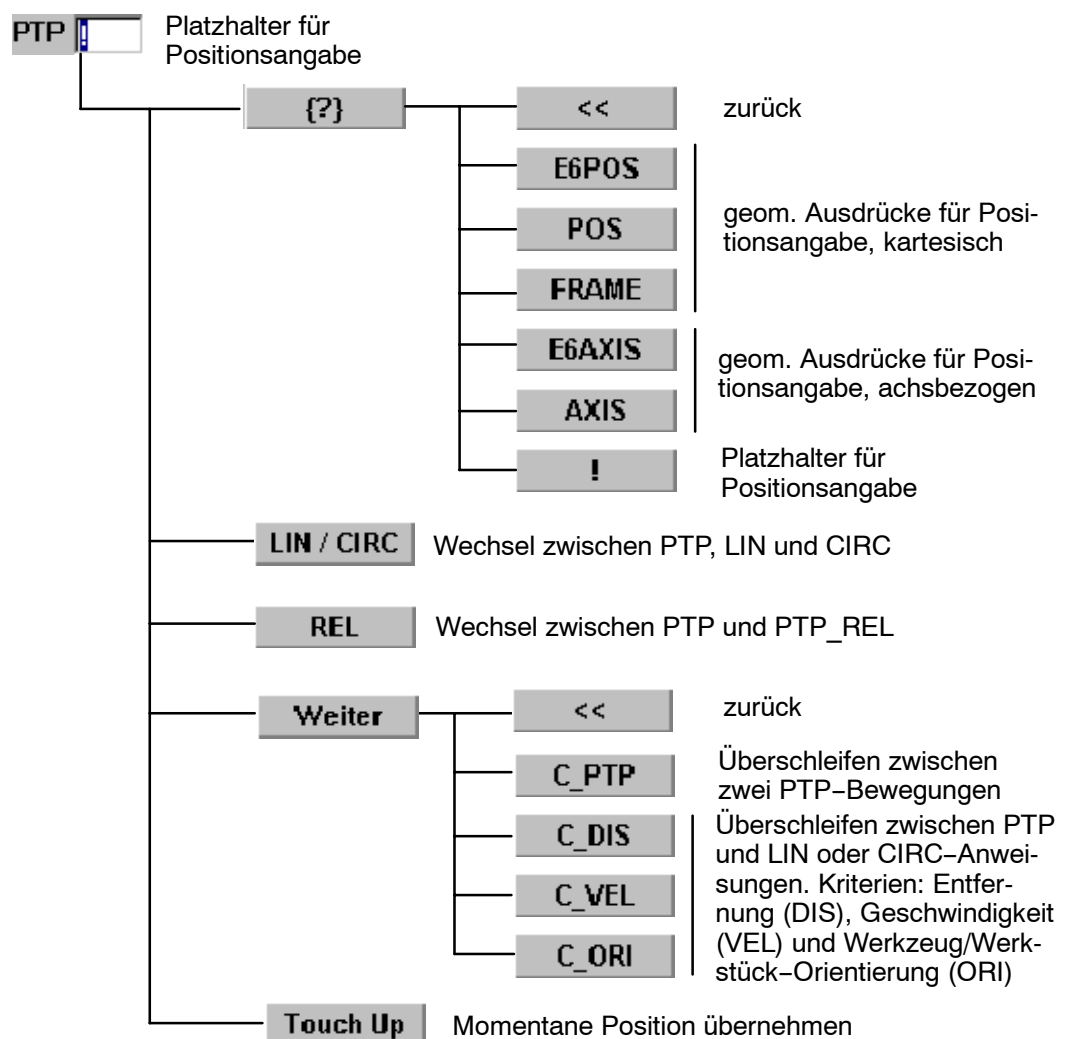
Weitere Informationen hierzu finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Datenmanipulation]** unter **“Geometrischer Operator”**.

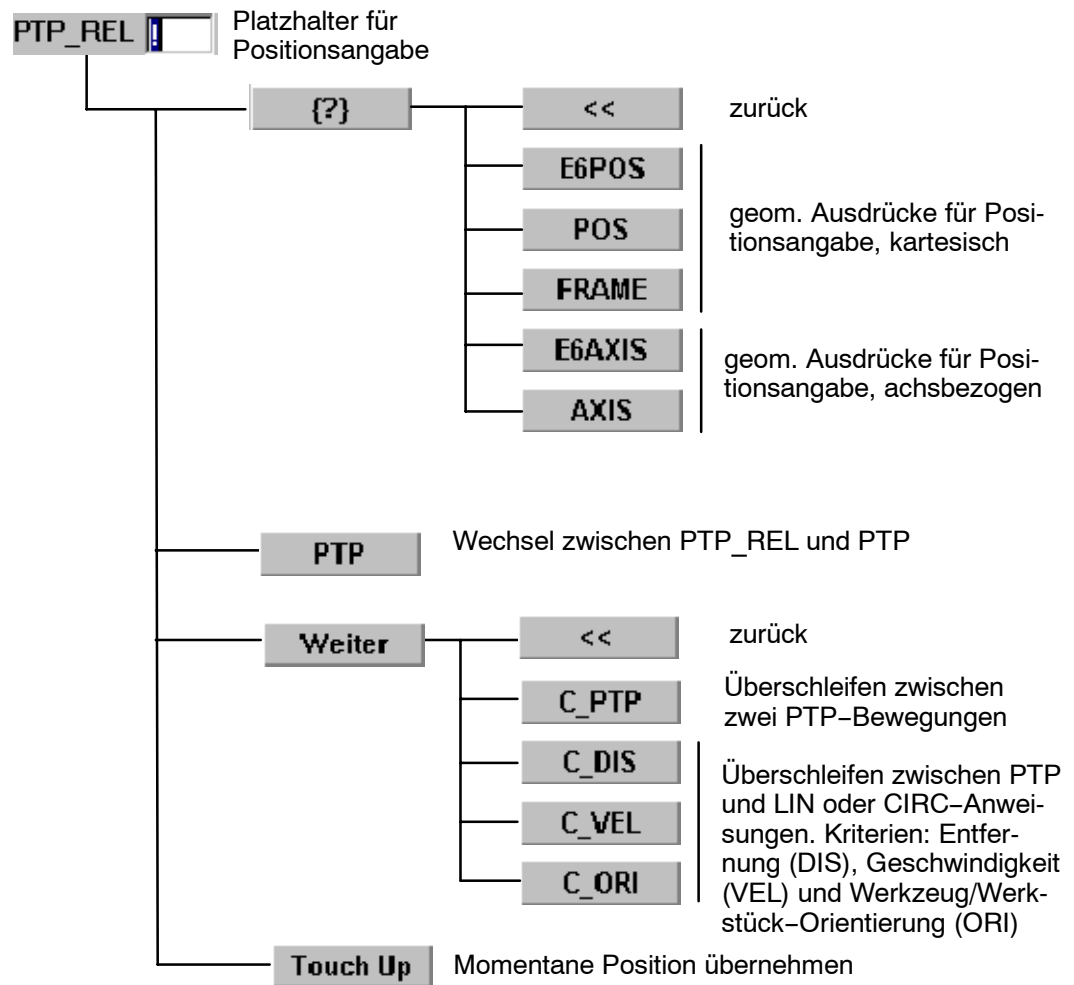
## 5.2 [PTP] Positionierung

Die Positionierung des Robotersystems erfolgt hier auf dem schnellsten Weg zwischen zwei Punkten im Arbeitsraum. Da die Bewegung in allen Achsen gleichzeitig beginnt und endet, ist eine Synchronisation der Achsen notwendig. Die Bahn des Roboters ist dabei nicht exakt vorherzusehen.



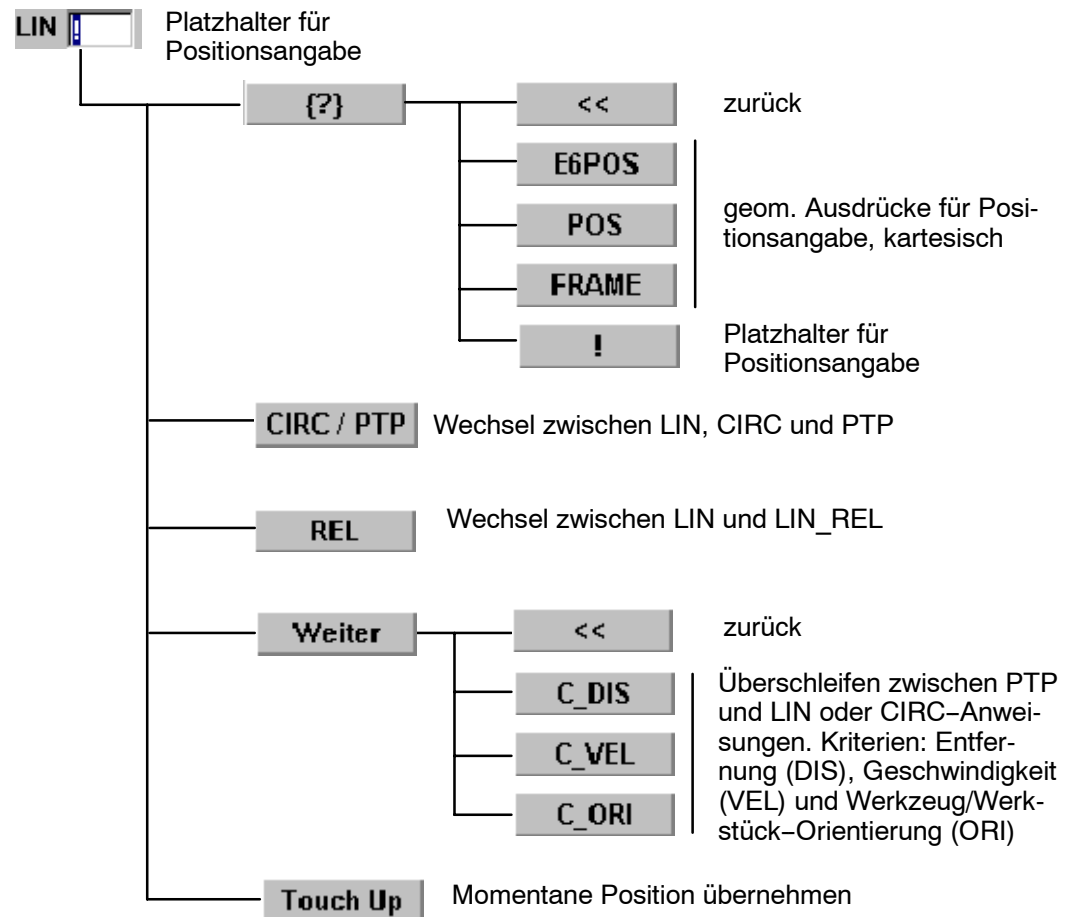
**Bei der Verwendung dieses Befehls läßt sich die genaue Bahn des Roboters nicht exakt vorhersehen. In der Nähe von Hindernissen innerhalb des Arbeitsraumes besteht deshalb Kollisionsgefahr. Das Fahrverhalten des Roboters muß in der Nähe der Hindernisse mit verminderter Geschwindigkeit getestet werden !**

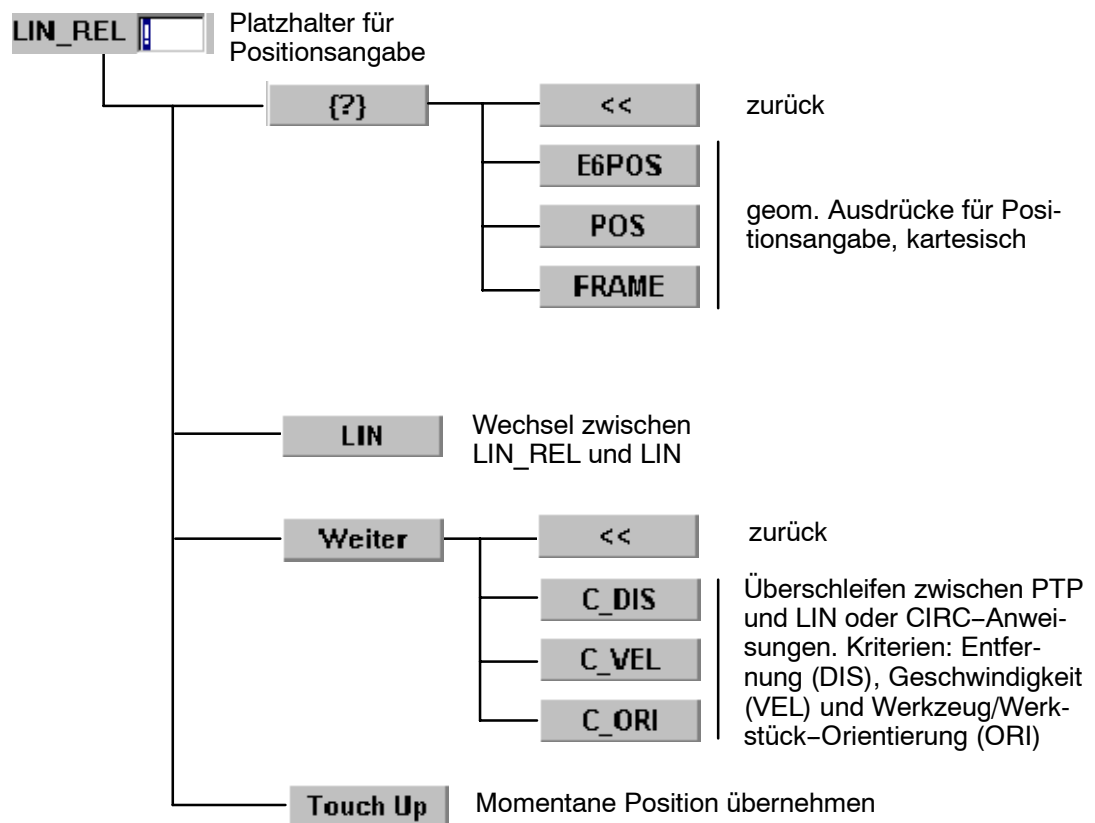




### 5.3 [LIN] Geradlinige Bewegung

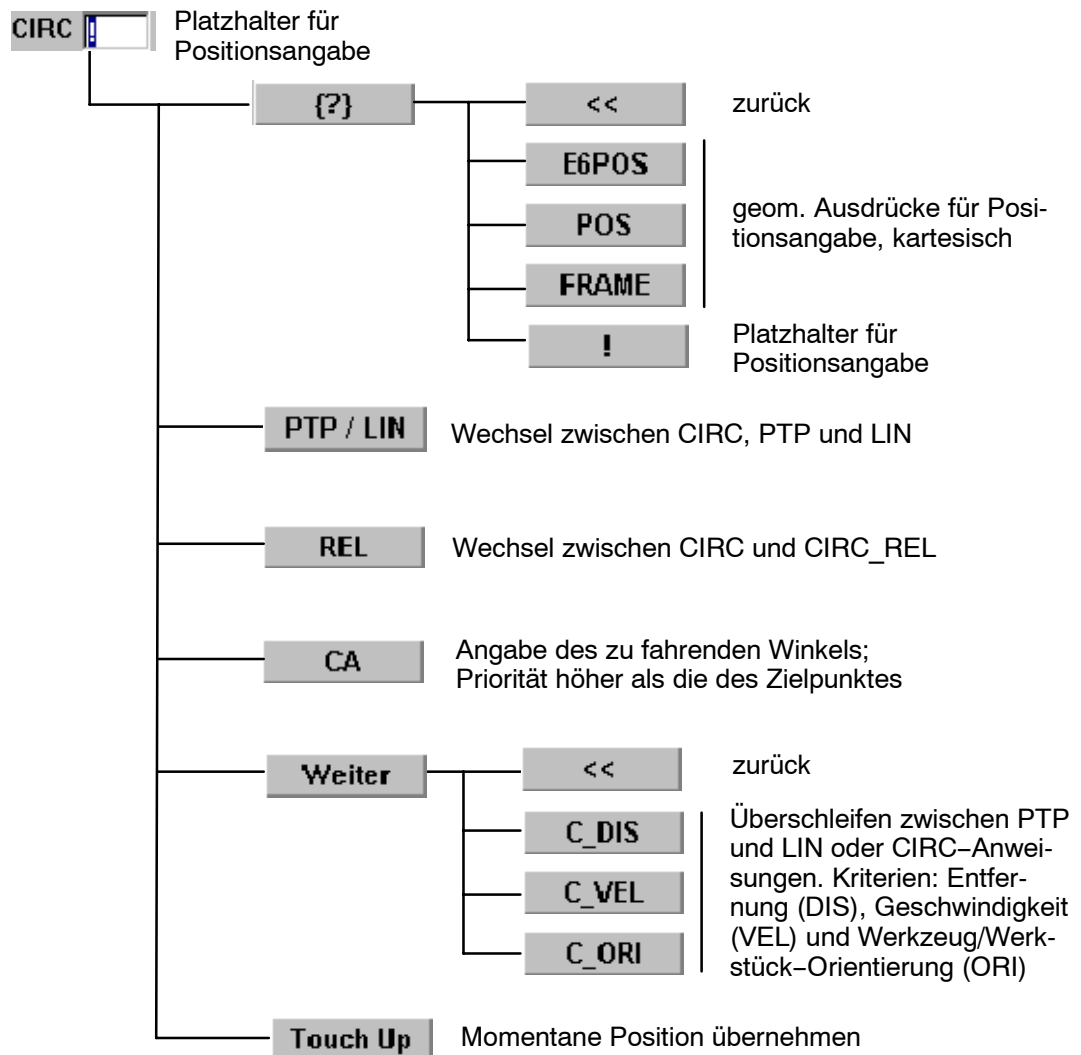
Die Positionierung des Robotersystems erfolgt hier auf dem kürzesten Weg zwischen zwei Punkten im Arbeitsraum, einer Geraden. Die Achsen des Robotersystems werden dabei so synchronisiert, daß die Bahngeschwindigkeit auf dieser Geraden immer gleich bleibt.

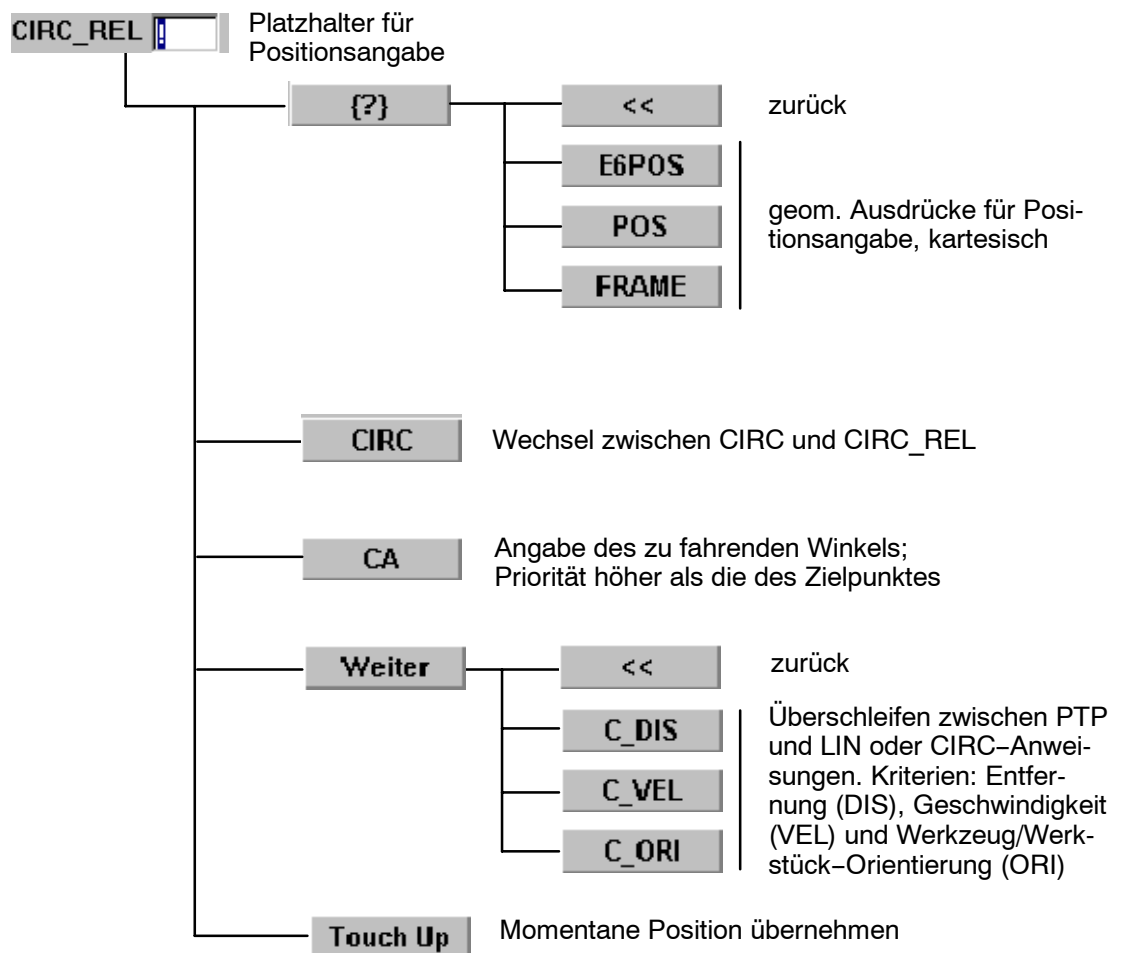




## 5.4 [CIRC] Kreisbahnbewegung

Die Positionierung des Robotersystems erfolgt hier entlang einer Kreisbahn im Arbeitsraum, die durch Startpunkt, Hilfspunkt und Zielpunkt definiert wird. Die Achsen des Robotersystems werden dabei so synchronisiert, daß die Bahngeschwindigkeit auf dieser Kreisbahn immer gleich bleibt.







## 6 Programmablaufkontrolle

### 6.1 Programmverzweigungen

#### 6.1.1 Sprunganweisung

Die einfachste Form der Programmverzweigung ist die der unbedingten Sprunganweisung. Sie wird ohne eine bestimmte Bedingung zu reflektieren auf jeden Fall ausgeführt. Durch die Anweisung

GOTO

**GOTO MERKER**

springt der Programmzeiger an die Position **MERKER**. Die Position muß allerdings mit **MERKER**:

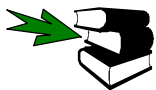
auch irgendwo im Programm definiert sein. Die Sprunganweisung selber läßt keine Rückschlüsse auf die damit erzeugte Programmstruktur zu. Deshalb sollte der Name der Sprungmarke möglichst so gewählt werden, daß die damit hervorgerufene Sprungaktion etwas verständlicher wird. So ist es z.B. ein Unterschied, ob man schreibt

GOTO MARKE\_1

oder

GOTO KLEBERSTOP

Da die **GOTO**-Anweisung sehr schnell zu unstrukturierten und unübersichtlichen Programmen führt, und ferner jede **GOTO**-Anweisung durch eine andere Schleifenanweisung ersetzt werden kann, sollte möglichst wenig mit **GOTO** gearbeitet werden.



Ein Beispiel zu "**GOTO**" finden Sie in diesem Kapitel im Abschnitt **[Schleifen]** unter "**Nicht abweisende Schleifen**"



NOTIZEN:

## 6.1.2 Bedingte Verzweigung

**IF** Die strukturierte **IF**-Anweisung gestattet die Formulierung bedingter Anweisungen und die Auswahl aus zwei Alternativen. In der allgemeinen Form lautet die Anweisung

```
IF Ausführbedingung THEN
    Anweisungen
ELSE
    Anweisungen
ENDIF
```

Die Ausführbedingung ist ein boolescher Ausdruck. Ist die Ausführbedingung erfüllt, wird der **THEN**-Block ausgeführt. Im anderen Fall kann der **ELSE**-Block ausgeführt oder auf den **ELSE**-Block verzichtet werden. Ein Verzicht bedeutet das sofortige Verlassen der Verzweigung.

Es sind beliebig viele Anweisungen zulässig. Die Anweisungen können insbesondere auch weitere **IF**-Anweisungen sein. Eine Schachtelung von **IF**-Blöcken ist also möglich. Jede **IF**-Anweisung muß jedoch mit einem eigenen **ENDIF** abgeschlossen werden.

In der folgenden Programmsequenz wird, sofern Eingang 10 auf **FALSE** ist, die **HOME**-Position angefahren. Ist Eingang 10 gesetzt, dann wird, wenn der Wert von Variable A größer als der von B ist, zunächst Ausgang 1 gesetzt und Punkt 1 angefahren. Unabhängig von A und B wird auf jeden Fall bei gesetztem Eingang 10 die Variable A um 1 erhöht und dann die **HOME**-Position angefahren:

```
...
INT A,B
...
IF $IN[10]==FALSE THEN
    PTP HOME
ELSE
    IF A>B THEN
        $OUT[1]=TRUE
        LIN PUNKT1
    ENDIF
    A=A+1
    PTP HOME
ENDIF
...
```



### NOTIZEN:

### 6.1.3 Verteiler

SWITCH  
Blockkennung

Liegen mehr als 2 Alternativen vor, kann dies entweder mit einer geschachtelten IF-Konstruktion oder – wesentlich komfortabler – mit dem Verteiler SWITCH programmiert werden.

Die SWITCH-Anweisung ist eine Auswahlanweisung für verschiedene Programmzweige. Ein Auswahlkriterium wird vor der SWITCH-Anweisung mit einem bestimmten Wert belegt. Stimmt dieser Wert mit einer Blockkennung überein, so wird der entsprechende Programmzweig abgearbeitet und das Programm springt ohne Berücksichtigung der folgenden Blockkennungen zur ENDSWITCH-Anweisung vor. Stimmt keine Blockkennung mit dem Auswahlkriterium überein, so wird, falls vorhanden, ein DEFAULT-Block abgearbeitet. Anderenfalls wird mit der Anweisung nach der ENDSWITCH-Anweisung fortgefahren.

Es ist zulässig, einem Programmzweig mehrere Blockkennungen zuzuordnen. Umgekehrt ist es nicht sinnvoll eine Blockkennung mehrmals zu verwenden, da immer nur der erste Programmzweig mit der entsprechenden Kennung berücksichtigt wird.

Zulässige Datentypen des Auswahlkriteriums sind INT, CHAR und ENUM. Der Datentyp von Auswahlkriterium und Blockkennung muß übereinstimmen.

Die DEFAULT-Anweisung kann fehlen, darf aber innerhalb einer SWITCH-Anweisung nur einmal vorkommen.

Mit der SWITCH-Anweisung können Sie somit zum Beispiel in Abhängigkeit von einer Programmnummer verschiedene Unterprogramme aufrufen. Die Programmnummer könnte beispielsweise von der SPS an die digitalen Eingänge der KR C1 angelegt werden (s. Abschnitt 7.3 zur SIGNAL-Anweisung). Dadurch steht sie als Auswahlkriterium in Form eines Integer-Wertes zur Verfügung.



```

DEF MAIN()
...
SIGNAL PROG_NR $IN[1] TO $IN[4]
                ;In die INT-Variable PROG_NR wird von der SPS
                ;jetzt die gewünschte Programmnummer abgelegt
...
SWITCH PROG_NR
  CASE 1                ;wenn PROG_NR=1
    TEIL_1()
  CASE 2                ;wenn PROG_NR=2
    TEIL_2()
    TEIL_2A()
  CASE 3,4,5            ;wenn PROG_NR=3, 4 oder 5
    $OUT[3]=TRUE
    TEIL_345()
  DEFAULT              ;wenn PROG_NR<>1,2,3,4,5
    ERROR_UP()
ENDSWITCH
...
END

```



In ähnlicher Weise ist das standardmäßig auf der Steuerung vorhandene CELL-Programm (CELL.SRC) aufgebaut.

## 6.2 Schleifen

Die nächste Grundstruktur zur Programmablaufkontrolle sind die Schleifen, die bis zum Eintritt einer bestimmten Bedingung die wiederholte Abarbeitung einer oder mehrerer Anweisungen beinhalten. Schleifen werden nach der Form der Bedingung und der Stelle der Abfrage auf Fortsetzung unterschieden.

Ein Sprung von außen in einen Schleifenkörper ist nicht erlaubt und wird von der Steuerung abgelehnt (Fehlermeldung).

### 6.2.1 Zählschleife

Zählschleifen werden so lange ausgeführt, bis eine Zählvariable entweder durch Hoch- oder Runterzählen einen bestimmten Endwert über- oder unterschreitet. In KRL steht dafür die **FOR**-Anweisung zur Verfügung. Mit

**FOR**

```
FOR Zähler = Start TO Ende STEP Schrittweite
    Anweisungen
ENDFOR
```

läßt sich somit sehr übersichtlich eine bestimmte Anzahl von Durchläufen programmieren.

Als **Start**-Wert und **Ende**-Wert des Zählers geben Sie jeweils einen Ausdruck vom Typ Integer an. Die Ausdrücke werden einmal vor Beginn der Schleife ausgewertet. Die **INT**-Variable **Zähler** (muß vorher deklariert sein) wird mit dem Startwert vorbesetzt und nach jedem Schleifendurchlauf um die Schrittweite erhöht oder vermindert.

Die **Schrittweite** darf keine Variable sein und darf nicht Null sein. Wenn keine Schrittweite angegeben ist, hat sie den Standardwert 1. Auch negative Werte sind für die Schrittweite zugelassen.

Zu jeder **FOR**-Anweisung muß es eine **ENDFOR**-Anweisung geben. Das Programm wird nach Beendigung des letzten Schleifendurchlaufs mit der ersten Anweisung hinter **ENDFOR** fortgesetzt.

Den Wert des Zählers können Sie sowohl innerhalb als auch außerhalb der Schleife benutzen. Innerhalb der Schleifen dient er zum Beispiel als aktueller Index für die Bearbeitung von Feldern. Nach dem Verlassen der Schleife hat der Zähler den zuletzt angenommenen Wert (also **Ende+Schrittweite**).

In folgendem Beispiel werden zunächst die Achsgeschwindigkeiten **\$VEL\_AXIS[1]...\$VEL\_AXIS[6]** auf 100% gesetzt. Danach werden die Komponenten eines 2-dimensionalen Feldes mit den berechneten Werten initialisiert. Das Ergebnis ist in Tab. 8 dargestellt.



```

DEF FOR_PROG()
...
INT I,J
INT FELD[10,6]
...
FOR I=1 TO 6
    $VEL_AXIS[I] = 100    ;alle Achsgeschwindigkeiten auf 100%
ENDFOR
...
FOR I=1 TO 9 STEP 2
    FOR J=6 TO 1 STEP -1
        FELD[I,J] = I*2 + J*J
        FELD[I+1,J] = I*2 + I*J
    ENDFOR
ENDFOR
;I hat jetzt den Wert 11, J den Wert 0
...
END

```

Index		I =									
		1	2	3	4	5	6	7	8	9	10
J =	6	38	8	42	24	46	40	50	56	54	72
	5	27	7	31	21	35	35	39	49	43	63
	4	18	6	22	18	26	30	30	42	34	54
	3	11	5	15	15	19	25	23	35	27	45
	2	6	4	10	12	14	20	18	28	22	36
	1	3	3	7	9	11	15	15	21	19	27

Tab. 8 Ergebnis der Berechnung aus Beispiel 5.2



## NOTIZEN:

## 6.2.2 Abweisende Schleife

**WHILE** Die WHILE-Schleife fragt zu Beginn der Wiederholung nach einer Ausführbedingung. Sie ist eine abweisende Schleife, weil sie kein einziges Mal durchlaufen wird, wenn die Ausführbedingung von Anfang an schon nicht erfüllt ist. Die Syntax der WHILE-Schleife lautet:

```
WHILE Ausfuehrbedingung
    Anweisungen
ENDWHILE
```

Die Ausfuehrbedingung ist ein logischer Ausdruck, der eine boolsche Variable, ein boolscher Funktionsaufruf oder eine Verknüpfung mit einem boolschen Ergebnis sein kann.

Der Anweisungsblock wird ausgeführt, wenn die logische Bedingung den Wert TRUE hat, d.h. die Ausführbedingung erfüllt ist. Wenn die logische Bedingung den Wert FALSE hat, wird das Programm mit der nächsten Anweisung hinter ENDWHILE fortgesetzt. Jede WHILE-Anweisung muß deshalb durch eine ENDWHILE-Anweisung beendet werden.

Die Verwendung von WHILE wird aus Beispiel 5.3 ersichtlich.



```
DEF WHILE_PR( )
...
INT X,W
...
WHILE $IN[4] == TRUE ;Durchlauf solange Eingang 4 gesetzt ist
    PTP PALETTE
    $OUT[2] = TRUE
    PTP POS_2
    $OUT[2] = FALSE
    PTP HOME
ENDWHILE
...
X = 1
W = 1
WHILE W < 5; ;Durchlauf solange W kleiner 5 ist
    X = X * W
    W = W + 1
ENDWHILE
;W ist jetzt 5
;X ist jetzt 1•2•3•4 = 24
...
W = 100
WHILE W < 100 ;Durchlauf solange W kleiner 100 ist
    $OUT[15] = TRUE
    W = W + 1
ENDWHILE
... ;Schleife wird nie durchlaufen, W bleibt 100
END
```



### NOTIZEN:

### 6.2.3 Nicht abweisende Schleife

**REPEAT** Das Gegenstück zur **WHILE**-Schleife ist die **REPEAT**-Schleife. Bei **REPEAT** wird erst am Ende der Schleife nach einer Abbruchbedingung gefragt. Deshalb werden **REPEAT**-Schleifen auf jeden Fall einmal durchlaufen, auch wenn die Abbruchbedingung schon vor Schleifenanfang erfüllt ist.

**REPEAT**  
 Anweisungen  
**UNTIL** Abbruchbedingung

Die Abbruchbedingung ist analog zur Ausführbedingung der **WHILE**-Schleife ein logischer Ausdruck, der eine boolsche Variable, ein boolscher Funktionsaufruf oder eine Verknüpfung mit einem boolschen Ergebnis sein kann:



```

DEF REPEAT_P()
...
INT W
...
REPEAT
  PTP PALETTE
  $OUT[2]=TRUE
  PTP POS_2
  $OUT[2]=FALSE
  PTP HOME
UNTIL $IN[4] == TRUE    ;Durchlauf bis Eingang 4 gesetzt wird
...
X = 1
W = 1
REPEAT
  X = X * W
  W = W + 1
UNTIL W == 4           ;Durchlauf bis W gleich 4 wird
                        ;W ist jetzt 4
                        ;X ist jetzt 1•2•3•4 = 24
...
W = 100
REPEAT
  $OUT[15] = TRUE
  W = W + 1
UNTIL W > 100          ;Durchlauf bis W groesser 100 wird
                        ;mindestens 1 Schleifendurchlauf, d.h.
                        ;W ist jetzt 101, Ausgang 15 ist gesetzt
...
END

```

Mit WHILE und REPEAT haben Sie nun ein sehr mächtiges Werkzeug zur strukturierten Programmierung an der Hand, mit dem Sie die meisten GOTO-Anweisungen ersetzen können. Die Anweisungsfolge

```
...
X = 0
G = 0
MERKER:
X = X + G
G = G + 1
    IF G > 100 THEN
        GOTO FERTIG
    ENDIF
GOTO MERKER:
FERTIG:
...
```

läßt sich beispielsweise mit REPEAT sehr viel eleganter verwirklichen:

```
...
X = 0
G = 0
REPEAT
    X = X + G
    G = G + 1
UNTIL G > 100
...
```



### NOTIZEN:

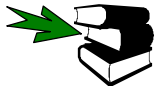


### 6.2.4 Endlosschleife

**LOOP** Mit der **LOOP**-Anweisung lassen sich Endlosschleifen programmieren:

```
LOOP
    Anweisungen
ENDLOOP
```

Die wiederholte Ausführung des Anweisungsblocks läßt sich nur mittels der **EXIT**-Anweisung beenden.



Weitere Informationen zur **Exit-Anweisung** finden Sie im nächsten Abschnitt.

### 6.2.5 Vorzeitige Beendigung von Schleifendurchläufen

**EXIT** Mit der **EXIT**-Anweisung können Sie jede Schleife vorzeitig beenden. Durch Aufruf von **EXIT** innerhalb des Anweisungsblocks der Schleife werden die Schleifendurchläufe sofort beendet und das Programm wird hinter der Schleifenendanweisung fortgesetzt.

Durch die Wahl geschickter Abbruch- oder Ausführbedingungen ist die **EXIT**-Anweisung in **REPEAT**- oder **WHILE**-Schleifen meist nicht notwendig. Für die Endlosschleife stellt **EXIT** allerdings die einzige Möglichkeit dar, die Schleifendurchläufe zu beenden. Dazu folgendes Beispiel:



```
DEF EXIT_PRO()
PTP HOME
LOOP                                ;Start der Endlosschleife
    PTP POS_1
    LIN POS_2
    IF $IN[1] == TRUE THEN
        EXIT                        ;Abbruch, wenn Eingang 1 gesetzt
    ENDIF
    CIRC HELP_1,POS_3
    PTP POS_4
ENDLOOP                            ;Ende der Endlosschleife
PTP HOME
END
```

#### NOTIZEN:



## 6.3 Warteanweisungen

**WAIT** Mit der **WAIT**-Anweisung können Sie das Anhalten des Programms bis zum Eintritt einer bestimmten Situation bewirken. Man unterscheidet zwischen dem Warten auf das Eintreten eines bestimmten Ereignisses und dem Einlegen von Wartezeiten.

### 6.3.1 Warten auf ein Ereignis

Mit der Anweisung

**WAIT FOR** Bedingung

können Sie den Programmlauf bis zum Eintreten des mit **Bedingung** spezifizierten Ereignisses anhalten:

- Wenn der logische Ausdruck **Bedingung** beim **WAIT**-Aufruf bereits **TRUE** ist, wird der Programmablauf nicht angehalten (es wird jedoch trotzdem ein Vorlaufstop ausgelöst).
- Ist **Bedingung** **FALSE**, so wird der Programmlauf angehalten, bis der Ausdruck den Wert **TRUE** annimmt.

Beispiele:

```
WAIT FOR $IN[14]           ;wartet bis Eingang 14 TRUE ist
WAIT FOR BIT_1 == FALSE    ;wartet bis die Var. BIT_1 = FALSE ist
```

Der Compiler erkennt nicht, wenn der Ausdruck durch eine fehlerhafte Formulierung nie den Wert **TRUE** annehmen kann. In diesem Fall wird der Programmablauf endlos angehalten, weil der Interpreter auf eine unerfüllbare Bedingung wartet.

### 6.3.2 Wartezeiten

Die **WAIT SEC**-Anweisung dient zum Programmieren von Wartezeiten in Sekunden:

**WAIT SEC** Zeit

**Zeit** ist ein arithmetischer **REAL**-Ausdruck, mit dem Sie die Anzahl der Sekunden angeben, für die der Programmlauf unterbrochen werden soll. Ist der Wert negativ, so wird nicht gewartet.

Beispiele:

```
WAIT SEC 17.542
WAIT SEC ZEIT*4+1
```



#### NOTIZEN:

## 6.4 Anhalten des Programms

**HALT** Wollen Sie den Programmablauf unterbrechen und die Bearbeitung anhalten, so programmieren Sie die Anweisung

**HALT**

Die zuletzt durchlaufene Bewegungsanweisung wird jedoch noch vollständig ausgeführt. Fortgesetzt wird der Programmlauf nur durch Drücken der Starttaste. Danach wird die nächste Anweisung nach **HALT** ausgeführt.



**Sonderfall:** In einer Interrupt-Routine wird der Programmlauf durch eine **HALT**-Anweisung erst nach vollständiger Abarbeitung des Vorlaufs angehalten (s. Abschnitt 9 Interrupt).

Ausnahme: Bei Programmierung einer **BRAKE** – Anweisung wird sofort gehalten.



NOTIZEN:

## 6.5 Quittieren von Meldungen

CONFIRM Mit der Anweisung

**CONFIRM V\_Nummer**

können Sie quittierbare Meldungen programmgesteuert quittieren. Nach erfolgreicher Quittierung ist die mit der Verwaltungsnummer V\_Nummer spezifizierte Meldung nicht mehr vorhanden.

Nach dem Aufheben eines Stop-Signals wird beispielsweise stets eine Quittungsmeldung ausgegeben. Vor dem Weiterarbeiten muß diese zunächst quittiert werden. Folgendes Unterprogramm erkennt und quittiert diese Meldung automatisch, sofern die richtige Betriebsart (kein Handbetrieb) gewählt ist, und der Stop-Zustand tatsächlich aufgehoben wurde (da ein Roboterprogramm nicht startbar ist, wenn eine Quittungsmeldung anliegt, muß das Unterprogramm in einem Submit-File laufen):



```
DEF AUTO_QUIT()
INT M
DECL STOPMESS MLD      ;vordefinierte Strukturtyp fuer Stopmel-
dungen
IF $STOPMESS AND $EXT THEN      ;Stopmeldung und Betriebsart
prüfen
    M=MBX_REC($STOPMB_ID,MLD)    ;aktuellen Zustand in MLD einle-
sen
    IF M==0 THEN                ;Ueberpruefen, ob Quittierung erfolgen
darf
        IF ((MLD.GRO==2) AND (MLD.STATE==1)) THEN
            CONFIRM MLD.CONFNO    ;Quittierung dieser Meldung
        ENDIF
    ENDIF
ENDIF
END
```



### NOTIZEN:

## 7 Ein-/Ausgabeeinheiten

### 7.1 Allgemeines

Die KR C1 kennt 1026 Eingänge und 1024 Ausgänge. Im KUKA-Standardsteuerschrank stehen dem Anwender am Stecker X11 (MFC-Baugruppe) folgende Eingänge und Ausgänge zur Verfügung:

Eingänge	1 ...16	
Ausgänge	1 ...16	(mit max. 100 mA belastbar; 100% Gleichzeitigkeit)
Ausgänge	17 ...20	(mit max. 2 A belastbar; 100% Gleichzeitigkeit)

Optional können weitere Ein-/Ausgänge z.B. über Feldbusse projektiert werden.

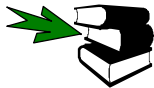
Eingänge können gelesen, Ausgänge gelesen und geschrieben werden. Sie werden über die Systemvariablen  $\$IN[Nr]$  bzw.  $\$OUT[Nr]$  angesprochen. Nicht benutzte Ausgänge können als Merker benutzt werden.

Die Ein-/Ausgänge der MFC-Baugruppe können in der Datei "IOSYS.INI" auf andere Bereiche umrangiert werden.



**Aus Sicherheitsgründen lösen alle Ein-/Ausgabeeinheiten und Zugriffe auf Systemvariablen zur Ein-/Ausgabe einen Vorlaufstop aus.**

Zugriffe auf Systemvariablen zur Ein-/Ausgabe lösen mit einer vorangestellten CONTINUE Anweisung keinen Vorlaufstop aus.



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



#### NOTIZEN:

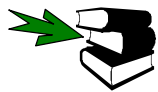
## 7.2 Binäre Ein-/Ausgänge

Werden Ein-/Ausgänge einzeln angesprochen, so spricht man von binären Ein-/Ausgängen. Binäre Ausgänge können nur 2 Zustände haben: Low oder High. Deshalb werden sie als Variablen vom Typ `BOOL` behandelt.

Ausgänge können demnach mit den Systemvariablen

`$OUT[Nr] = TRUE` gesetzt werden, und mit  
`$OUT[Nr] = FALSE` zurückgesetzt werden.

Der Zustand eines Eingangs `$IN[Nr]` kann in eine boolsche Variable eingelesen werden oder als boolscher Ausdruck in Programmablauf-, Interrupt- oder Triggeranweisungen verwendet werden.



Weitere Informationen finden Sie im Kapitel **[Programmablaufkontrolle]**, Kapitel **[Interruptbehandlung]**, Kapitel **[Bahnbezogene Schaltaktionen]**.

Die Anweisungsfolgen

```

BOOL SCHALTER
:
SCHALTER = $IN[6]
IF SCHALTER == FALSE THEN
:
ENDIF
    
```

und

```

IF $IN[6] == FALSE THEN
:
ENDIF
    
```

haben somit die gleiche Bedeutung.

**SIGNAL**

In der KR C1 ist es weiterhin möglich, einzelnen Ein- oder Ausgängen Namen zuzuweisen. Hierzu dient die Signalvereinbarung. Sie muß wie alle Vereinbarungen im Vereinbarungsteil des Programms stehen. Man kann also auch

```

SIGNAL SCHALTER $IN[6]
:
IF SCHALTER == FALSE THEN
:
ENDIF
    
```

programmieren. Die Variable Schalter wird intern wieder als `BOOL` deklariert.



Systemeingänge und -ausgänge können ebenfalls mit `$IN` und `$OUT` angesprochen werden. Systemausgänge sind allerdings schreibgeschützt. Eingang 1025 ist immer `TRUE`, Eingang 1026 ist immer `FALSE`. Diese Eingänge werden z.B. in den Maschinendaten als "Dummy"-Variablen benutzt. Eine mehrfache Nutzung ist zulässig.



Wie Ein-/Ausgänge eingesetzt werden, erläutert Beispiel 6.1:

```

DEF BINSIG ( )
;----- Deklarationsteil -----
EXT BAS (BAS_COMMAND:IN,REAL:IN )
DECL AXIS HOME
SIGNAL ABRUCH $IN[16]
SIGNAL LINKS $OUT[13]
SIGNAL MITTE $OUT[14]
SIGNAL RECHTS $OUT[15]
;----- 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
LINKS=FALSE
MITTE=TRUE ;in mittlerer Stellung
RECHTS=FALSE

WHILE ABRUCH==FALSE ;Abbruch, wenn Eingang 16 gesetzt
  IF $IN[1] AND NOT LINKS THEN ;Eingang 1 gesetzt
    PTP {A1 45}
    LINKS=TRUE ;in linker Stellung
    MITTE=FALSE
    RECHTS=FALSE
  ELSE
    IF $IN[2] AND NOT MITTE THEN ;Eingang 2 gesetzt
      PTP {A1 0}
      LINKS=FALSE
      MITTE=TRUE ;in mittlerer Stellung
      RECHTS=FALSE
    ELSE
      IF $IN[3] AND NOT RECHTS THEN ;Eingang 3 gesetzt
        PTP {A1 -45}
        LINKS=FALSE
        MITTE=FALSE
        RECHTS=TRUE ;in rechter Stellung
      ENDIF
    ENDIF
  ENDIF
ENDWHILE
PTP HOME
END

```

Durch das Setzen der Eingänge 1, 2 oder 3 kann der Roboter in drei verschiedene Stellungen verfahren werden. Ist der Roboter in der gewünschten Position angekommen, so wird dies durch Setzen der entsprechenden Ausgänge 13, 14 oder 15 angezeigt.

Da diese Ausgänge also immer die aktuelle Stellung des Roboters anzeigen, kann mit der Abfrage

```

IF $IN[1] AND NOT $OUT[13] THEN
:
ENDIF

```

auch verhindert werden, daß der Roboter bei jedem Durchlauf der While-Schleife erneut versucht, auf die bereits eingenommen Position zu verfahren. Der Roboter verfährt also nur, wenn ein Eingang gesetzt ist (d.h. Anweisung zum Verfahren in gewünschte Position) **und** der zugehörige Ausgang **nicht** gesetzt ist (d.h. Roboter ist noch nicht in dieser Position) (siehe Tab. 9).

Durch Setzen des Eingangs 16 wird die While-Schleife und somit das Programm beendet.

\$IN [Nr] AND NOT \$OUT[Nr]		\$OUT[Nr]	
		TRUE	FALSE
\$IN[Nr]	TRUE	FALSE	TRUE
	FALSE	FALSE	FALSE

**Tab. 9** Wahrheitstabelle für eine “AND NOT” – Verknüpfung



NOTIZEN:



### 7.3 Digitale Ein-/Ausgänge

Mit der Signalvereinbarung kann man nicht nur einzelne Ein-/Ausgänge mit Namen versehen, sondern auch mehrere binäre Ein- oder Ausgänge zu einem digitalen Signal zusammenfassen. Mit der Vereinbarung

```
SIGNAL AUS $OUT[10] TO $OUT[20]
```

können nun z.B. die Ausgänge 10 bis 20 über die intern als **Integer** deklarierte Variable `AUS` als 11-Bit-Wort angesprochen werden.

Den so deklarierten Digitalausgang kann man über jede mögliche Integer-Zuweisung an die Variable `AUS` beschreiben, z.B.:

```
AUS = 35  
AUS = 'B100011'  
AUS = 'H23'
```

- Ein-/Ausgänge müssen in der Signalvereinbarung lückenlos und in aufsteigender Reihenfolge angegeben werden.
- Es können höchstens 32 Ein- bzw. Ausgänge zu einem Digitalsignal zusammengefaßt werden.
- Ein Ausgang darf in mehreren Signalvereinbarungen vorkommen.

Wenn man die Ausgänge 13 bis 15 von Beispiel 6.1 damit zu einer Variablen `POSITION` zusammenfaßt, ergibt sich folgendes modifizierte Programm:



```

DEF  BINSIG_D ( )

;----- Deklarationsteil -----
EXT  BAS (BAS_COMMAND:IN,REAL:IN )
DECL AXIS HOME
SIGNAL ABBRUCH $IN[16]
SIGNAL POSITION $OUT[13] TO $OUT[15]

;----- 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

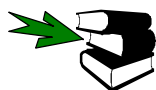
POSITION='B010'                ;in mittlerer Stellung

WHILE ABBRUCH==FALSE           ;Abbruch, wenn Eingang 16 gesetzt

    IF $IN[1] AND (POSITION<>'B001') THEN      ;Eingang 1 gesetzt
        PTP {A1 45}
        POSITION='B001'                        ;in linker Stellung
    ELSE
        IF $IN[2] AND (POSITION<>'B010') THEN  ;Eingang 2 gesetzt
            PTP {A1 0}
            POSITION='B010'                    ;in mittlerer Stellung
        ELSE
            IF $IN[3] AND (POSITION<>'B100') THEN;Eingang 3 gesetzt
                PTP {A1 -45}
                POSITION='B100'                ;in rechter Stellung
            ENDIF
        ENDIF
    ENDIF

ENDWHILE

PTP  HOME
END
    
```



Weitere Informationen finden Sie in diesem Kapitel Abschnitt **[Vordefinierte Digitaleingänge]**.



NOTIZEN:

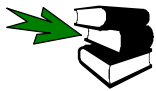
## 7.4 Impulsausgänge

**PULSE** Mit der **PULSE**-Anweisung können einzelne Ausgänge für eine bestimmte Dauer gesetzt oder rückgesetzt werden. Die Anweisung

```
PULSE( $OUT[ 4 ], TRUE, 0.7 )
```

setzt beispielsweise Ausgang 4 für eine Zeit von 0.7 Sekunden auf High-Pegel. Der Impuls kann dabei parallel zum Roboterprogramm ablaufen (der Interpreter wird dabei nicht angehalten).

Anstatt der direkten Angabe des Ausgangs mit `$OUT[Nr]` kann auch eine Signalvariable stehen.



Weitere Informationen finden Sie in diesem Kapitel, Abschnitt **[Binäre Ein-/Ausgänge]**.

Die realisierbaren Impulszeiten liegen zwischen 0.012 und  $2^{31}$  Sekunden. Das Raster beträgt 0.1 Sekunden. Die Steuerung rundet alle Werte auf ein Zehntel.



- Es dürfen maximal 16 Impulsausgänge gleichzeitig programmiert werden.
- Es können sowohl High-Impulse, als auch Low-Impulse programmiert werden.
- Bei "Programm RESET" oder "Programm abwählen" wird der Impuls abgebrochen.
- Ein anstehender Impuls kann durch Interrupts beeinflusst werden.
- Impulsausgänge können auch in der Steuerungsebene programmiert werden.
- Die **PULSE**-Anweisung löst einen Vorlaufstop aus. Nur in der **TRIGGER**-Anweisung wird sie bewegungsbegleitend ausgeführt.



**Ein Impuls wird NICHT abgebrochen bei**

- **NOT-AUS, Bedienstop oder Fehlerstop,**
- **Erreichen des Programmendes (END-Anweisung),**
- **Loslassen der Starttaste, wenn der Impuls vor der ersten Bewegungsanweisung programmiert wurde, und der Roboter SAK noch nicht erreicht hat.**

Im nächsten Programm finden Sie einige Beispiele zur Anwendung der PULSE-Anweisung:



```

DEF PULSIG ( )

;----- Deklarationsteil -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I
SIGNAL OTTO $OUT[13]

;----- 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}

FOR I=1 TO 16
$OUT[I]=FALSE      ;alle Ausgaenge auf LOW setzen
ENDFOR

;----- Hauptteil -----
PULSE ($OUT[1],TRUE,2.1) ;Impuls kommt direkt fuer 2.1s
PTP HOME                ;SAK-Fahrt

OTTO=TRUE                ;Ausgang 13 auf TRUE setzen
PTP {A3 45,A5 30}
PULSE (OTTO,FALSE,1.7) ;LOW-Impuls fuer 1.7s auf Ausgang 13
                        ;Impuls kommt erst nach Bewegung

WAIT SEC 2

FOR I=1 TO 4
PULSE ($OUT[I],TRUE,1) ;nacheinander die Ausgaenge 1-4
WAIT SEC 1              ;fuer 1s auf High
ENDFOR

;bahnbezogenes Aufbringen eines Impulses
TRIGGER WHEN DISTANCE=0 DELAY=50 DO PULSE ($OUT[8],TRUE,1.8)
LIN {X 1391,Y -319,Z 1138,A -33,B -28,C -157}

PTP HOME
CONTINUE                ;Vorlaufstop verhindern fuer Ausgang 15
PULSE ($OUT[15],TRUE,3) ;Impuls kommt direkt (im
Vorlauf)

                        ;auf Ausgang 16
PULSE ($OUT[16],TRUE,3) ;Impuls kommt erst nach HOME-Fahrt
END                    ;und steht noch nach END an
    
```

Beachten Sie in diesen Beispielen genau, ab wann die programmierten Impulse an den Ausgängen anliegen: Grundsätzlich führt die PULSE-Anweisung immer zu einem Stop des Rechnervorlaufs. Der Impuls liegt also erst nach Bewegungsbeendigung an.

Es gibt zwei Wege den Vorlaufstop zu verhindern:

- Programmierung einer CONTINUE-Anweisung direkt vor der PULSE-Anweisung
- Verwendung der PULSE-Anweisung in einer TRIGGER-Anweisung (bahnbezogene Schaltaktion)



Weitere Informationen finden Sie im Kapitel **[Bewegungsprogrammierung]** Abschnitt **[Rechnervorlauf]** (CONTINUE) und Kapitel **[Trigger – Bahnbezogene Schaltaktionen]** (TRIGGER).

## 7.5 Analoge Ein-/Ausgänge

Neben den binären Ein-/Ausgängen kennt die KR C1 auch analoge Ein-/Ausgänge. Über optionale Bussysteme stellt die KR C1 8 analoge Eingänge und 16 analoge Ausgänge zur Verfügung. Ausgänge können mit den Systemvariablen \$ANOUT[ 1 ] ... \$ANOUT[ 16 ] gelesen oder geschrieben werden, Eingänge können mit den Variablen \$ANIN[ 1 ] ... \$ANIN[ 8 ] nur gelesen werden.

ANIN  
ANOUT

Analoge Ein- und Ausgänge können sowohl statisch als auch dynamisch, d.h. durch ständige Abfrage im Interpolationstakt (z.Z 12 ms), angesprochen werden. Während das statische Lesen und Schreiben wie bei den binären Signalen durch einfache Wertzuweisungen erfolgt, benutzt man zum zyklischen Bearbeiten die speziellen Anweisungen ANIN und ANOUT.

### 7.5.1 Analoge Ausgänge

Die Ausgabewerte für die 16 analogen Ausgänge der KR C1 liegen zwischen  $-1.0 \dots +1.0$  und sind normiert auf die Ausgangsspannung  $\pm 10.0$  V. Falls der Ausgabewert die Grenzen  $\pm 1.0$  überschreitet wird der Wert abgeschnitten.

Zum Setzen eines Analogkanals weist man der entsprechenden \$ANOUT-Variable einfach einen Wert zu:

```
$ANOUT[ 2 ] = 0.5 ;Analogkanal 2 wird auf +5V gesetzt
```

oder

```
REAL V_KLEBER
```

```
:
```

```
V_KLEBER = -0.9
```

```
$ANOUT[ 15 ] = V_KLEBER ;Analogkanal 15 wird auf -9V gesetzt
```

Diese Zuweisungen sind statisch, da der Wert des beaufschlagten Kanals sich erst ändert, wenn der betreffenden Systemvariablen \$ANOUT[Nr] explizit ein neuer Wert zugewiesen wird.

Oft ist es jedoch wünschenswert, daß ein bestimmter analoger Ausgang während der Programmabarbeitung ständig in einer festgelegten Zykluszeit neu berechnet wird. Diese dynamische Analogausgabe erfolgt mit der ANOUT-Anweisung. Mit der Anweisung

```
ANOUT ON DRAHT = 0.8 * V_DRAHT
```

können Sie z.B. durch einfache Wertzuweisung an die Variable V\_DRAHT den mit der Signalvariablen DRAHT spezifizierten Analogausgang verändern. Die Spannung am entsprechenden Ausgang folgt somit der Variablen V\_DRAHT.

Die Variable DRAHT muß vorher natürlich mit der SIGNAL-Vereinbarung deklariert werden, z.B.:

```
SIGNAL DRAHT $ANOUT[ 2 ]
```

Mit

```
ANOUT OFF DRAHT
```

wird die zyklische Analogausgabe wieder beendet.

Der zyklisch aktualisierte Ausdruck, der für die Berechnung des Analog-Ausgabewertes angegeben werden muß, darf eine gewisse Komplexität nicht überschreiten. Die zulässige Syntax ist deshalb eingeschränkt und an der Technologie orientiert. Die vollständige Syntax lautet

ANOUT ON

```
ANOUT ON Signalname = Faktor * Regelglied ± Offset {DELAY=Zeit
```

für das Starten der zyklischen Analogausgabe, bzw.

ANOUT OFF

```
ANOUT OFF Signalname
```

für das Beenden. Die Bedeutung der einzelnen Argumente ist aus Tab. 10 ersichtlich.

Argument	Datentyp	Bedeutung
Signalname	REAL	Signalvariable, die den analogen Ausgang spezifiziert (muß mit SIGNAL deklariert sein). Nicht zulässig ist eine direkte Angabe von \$ANOUT[ Nr ] .
Faktor	REAL	Beliebiger Faktor, der eine Variable, ein Signalname oder eine Konstante sein kann.
Regelglied	REAL	Über das Regelglied wird der Analogausgang beeinflusst. Es kann eine Variable oder ein Signalname sein.
Offset	REAL	Optional kann ein Offset zum Regelglied programmiert werden. Der Offset muß eine Konstante sein.
Zeit	REAL	Mit dem Schlüsselwort DELAY und einer positiven oder negativen Zeitangabe in Sekunden kann optional das zyklisch berechnete Ausgangssignal verzögert (+) oder vorzeitig (-) ausgegeben werden.

**Tab. 10** Argumente in der ANOUT-Anweisung

**NOTIZEN:**

### 7.5.2 Analoge Eingänge

Die 8 analogen Eingänge der KR C1 können über die Variablen \$ANIN[ 1 ] bis \$ANIN[ 8 ] durch einfache Wertzuweisung an eine REAL-Variable gelesen werden:

```
REAL TEIL
:
TEIL = $ANIN[ 3 ]
```

oder

```
SIGNAL SENSOR3 $ANIN[ 3 ]
REAL TEIL
:
TEIL = SENSOR3
```

Die Werte in \$ANIN[ Nr ] bewegen sich zwischen +1.0 und –1.0 und repräsentieren eine Eingangsspannung von +10V bis –10V.

**ANIN** Zum zyklischen Lesen von Analogeingängen dient die ANIN-Anweisung. Mit ihr können bis zu 3 analoge Eingänge gleichzeitig eingelesen werden. Das Lesen erfolgt dabei im Interpolationstakt.

Mit der Anweisungsfolge

```
SIGNAL SENSOR3 $ANIN[ 3 ]
REAL TEIL
:
ANIN ON TEIL = 1 * SENSOR3
```

können Sie somit den analogen Eingang 3 zyklisch lesen, und mit der der Anweisung

```
ANIN OFF SENSOR3
```

das Lesen wieder beenden.

Zu beachten ist, daß zur gleichen Zeit höchstens 3 ANIN ON – Anweisungen aktiv sein dürfen. Es ist zulässig in beiden Anweisungen auf die selbe Analogschnittstelle zuzugreifen, oder die gleiche Variable zu beschreiben.

Die vollständige Syntax zum zyklische Lesen eines Analogeinganges lautet:

**ANIN ON**

```
ANIN ON Wert = Faktor * Signalname <± Offset
```

Das Beenden der zyklischen Überwachung wird mit

**ANIN OFF**

```
ANIN OFF Signalname
```

eingeleitet. Die Bedeutung der Argumente entnehmen Sie Tab. 11.

Argument	Datentyp	Bedeutung
Wert	REAL	Der Wert kann eine Variable oder ein (Ausgangs-) Signalname sein. In Wert wird das Ergebnis des zyklischen Lesens abgelegt.
Signalname	REAL	Signalvariable, die den analogen Eingang spezifiziert (muß mit SIGNAL deklariert sein). Nicht zulässig ist eine direkte Angabe von \$ANIN[ Nr ] .
Faktor	REAL	beliebiger Faktor, der eine Variable, ein Signalname oder eine Konstante sein kann.
Offset	REAL	Optional kann ein Offset programmiert werden. Der Offset kann eine Konstante, eine Variable oder ein Signalname sein.

**Tab. 11** Argumente in der ANIN-Anweisung

Im nachfolgenden Beispiel sind die Anweisungen zur Analogein- und ausgabe illustriert. Mit Hilfe der Systemvariablen \$TECHIN[1] und einem an einem Analogeingang angeschlossenen Bahnfolgesensor kann so z.B. eine Bahnkorrektur während der Bewegung vorgenommen werden. Die Variable \$VEL\_ACT, die stets die aktuelle Bahngeschwindigkeit enthält, kann mit entsprechenden Faktoren gewichtet zu einer geschwindigkeitsproportionalen Analogausgabe benutzt werden, also z.B. um die Klebermenge beim Kleberauftrag zu steuern.



```

DEF  ANSIG ( )

;----- Deklarationsteil -----
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I
SIGNAL KLEBER $ANOUT[1]           ;Duesenoeffnung fuer Kleber
SIGNAL KORREKTUR $ANIN[5]        ;Bahnfolgesensor

;----- 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}

FOR I=1 TO 16
$ANOUT[I]=0           ;alle Ausgaenge auf 0V setzen
ENDFOR

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

$ANOUT[3] = 0.7        ;Analogausgang 3 auf 7V

IF $ANIN[1] >= 0 THEN   ;Klebevorgang nur, wenn Analogeingang
1                        ;positive Spannung hat

    PTP POS1

    ;Bahnkorrektur entsprechend Sensorsignal mit Hilfe der
    ;Systemvariablen $TECHIN
    ANIN ON $TECHIN[1] = 1 * KORREKTUR + 0.1

    ;geschwindigkeitsproportionale Analogausgabe;Systemvariable
    $VEL_ACT enthält die aktuelle Bahngeschwindigkeit
    ANOUT ON KLEBER = 0.5 * $VEL_ACT + 0.2 DELAY = -0.12

    LIN POS2
    CIRC HILFSPOS,POS3
    ANOUT OFF KLEBER
    ANIN OFF KORREKTUR

    PTP POS4

ENDIF

PTP HOME
END
    
```



## 7.6 Vordefinierte Digitaleingänge

Die Steuerung stellt 6 digitale Eingänge zur Verfügung, welche über die Signalvariablen \$DIGIN1...\$DIGIN6 gelesen werden können. Die Eingänge sind Teil der normalen Anwendereingänge. Sie können eine Länge von 32 Bit und einen dazugehörigen Strobe-Ausgang haben.

Die Projektierung der digitalen Eingänge erfolgt in den Maschinendaten: "/mada/steu/\$maschine.dat". Mit einer Signalvereinbarung wird zunächst der Bereich und die Größe des Digitaleinganges festgelegt:

```
SIGNAL $DIGIN3 $IN[1000] TO $IN[1011]
```

Über die weiteren Systemvariablen \$DIGIN1CODE...\$DIGIN6CODE, \$STROBE1...\$STROBE6 und \$STROBE1LEV...\$STROBE6LEV werden die Vorzeicheninterpretationen, die zugehörigen Strobe-Ausgänge und die Art des Strobe-Signals festgelegt:

```
DECL DIGINCODE $DIGIN3CODE = #UNSIGNED ;ohne Vorzeichen
```

```
SIGNAL $STROBE3 $OUT[1000] ;Strobe-Ausgang festlegen
```

```
BOOL $STROBE3LEV = TRUE ;Strobe ist ein High-Impuls
```

Ein Strobe-Ausgang ist ein Ausgang der KR C1 mit einem bestimmten Impuls, der das Signal eines externen Gerätes (z.B. Drehgeber) zum Lesen einfriert.

Während verschiedene Digitaleingänge auf den selben Eingang zugreifen können, dürfen die Strobe-Signale NICHT den selben Ausgang beschreiben.

Der Wertebereich von \$DIGIN1...\$DIGIN6 hängt von der definierten Bit-Länge sowie von der Vorzeicheninterpretation (#SIGNED oder #UNSIGNED) ab:

12 Bit mit Vorzeichen (#SIGNED) Wertebereich: -2048...2047

12 Bit ohne Vorzeichen (#UNSIGNED) Wertebereich: 0...4095

Die Digitaleingänge können entweder statisch durch eine gewöhnliche Wertzuweisung gelesen werden:

```
INT ZAHL
:
ZAHL = $DIGIN2
```

DIGIN oder aber zyklisch mit einer DIGIN-Anweisung:

```
INT ZAHL
:
DIGIN ON ZAHL = FAKTOR * $DIGIN2 + OFFSET
:
DIGIN OFF $DIGIN2
```

Zur gleichen Zeit sind 6 DIGIN ON - Anweisungen zulässig. In der DIGIN ON - Anweisung kann auch auf analoge Eingangssignale zugegriffen werden (z.B. als FAKTOR). Die Syntax ist völlig analog zur ANIN ON-Anweisung:

DIGIN ON DIGIN ON Wert = Faktor \* Signalname (<± Offset

DIGIN OFF DIGIN OFF Signalname

Die Bedeutung der einzelnen Argumente ist in Tab. 12 beschrieben.

Argument	Datentyp	Bedeutung
Wert	REAL	Der Wert kann eine Variable oder ein (Ausgangs-)Signalname sein. In Wert wird das Ergebnis des zyklischen Lesens abgelegt.
Signalname	REAL	Signalvariable, die den digitalen Eingang spezifiziert. Zulässig sind nur \$DIGIN1...\$DIGIN6
Faktor	REAL	beliebiger Faktor, der eine Variable, ein Signalname oder eine Konstante sein kann.
Offset	REAL	Optional kann ein Offset programmiert werden. Der Offset kann eine Konstante, eine Variable oder ein Signalname sein.

**Tab. 12** Argumente in der DIGIN-Anweisung

**NOTIZEN:**

## 8 Unterprogramme und Funktionen

Damit für gleichartige, häufiger sich wiederholende Programmabschnitte die Schreibarbeit beim Programmieren sowie die Programmlänge reduziert werden, wurden Unterprogramme und Funktionen als Sprachkonstrukte eingeführt.

Ein bei größeren Programmen nicht zu unterschätzender Effekt von Unterprogrammen und Funktionen ist die Wiederverwendbarkeit einmal aufgeschriebener Algorithmen in anderen Programmen und besonders der Einsatz von Unterprogrammen zur Strukturierung des Programms. Diese Strukturierung kann zu einem hierarchischen Aufbau führen, so daß einzelne Unterprogramme, von einem übergeordneten Programm aufgerufen, Teilaufgaben vollständig bearbeiten und die Ergebnisse abliefern.

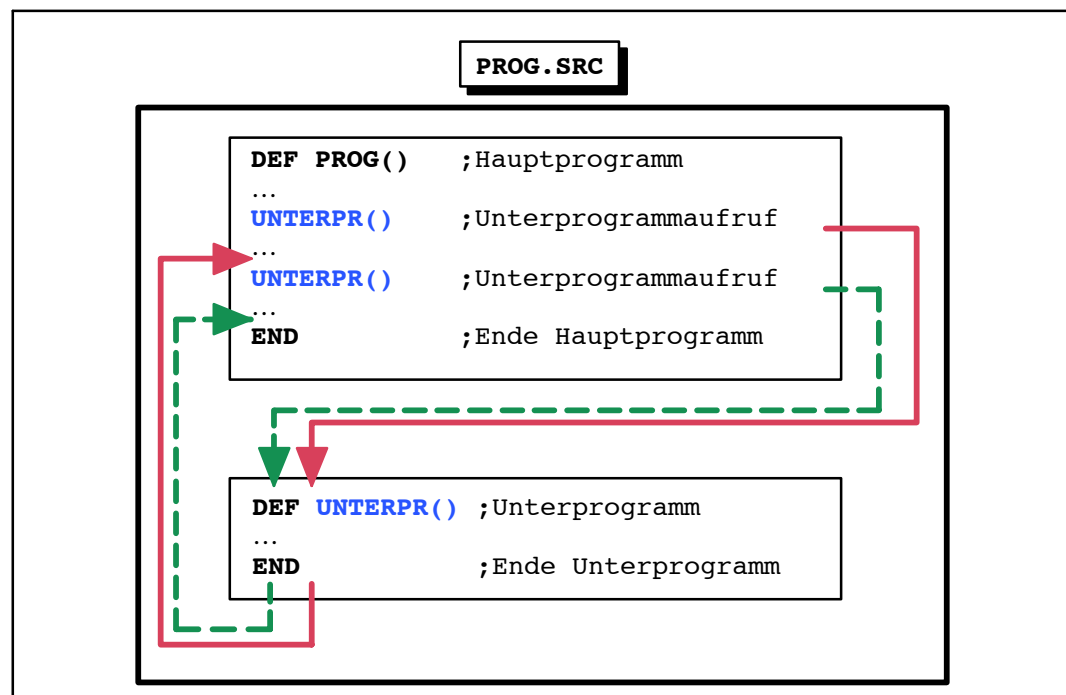
### 8.1 Vereinbarung

Ein Unterprogramm oder eine Funktion ist ein separater Programmteil mit Programmkopf, Vereinbarungsteil und Anweisungsteil, der von beliebigen Stellen im Hauptprogramm aufgerufen werden kann. Nach Abarbeitung des Unterprogramms oder der Funktion erfolgt ein Rücksprung an den nächsten Befehl nach dem Aufruf des Unterprogramms (s. Abb. 34).

Von einem Unterprogramm bzw. einer Funktion aus können weitere Unterprogramme und/oder Funktionen aufgerufen werden. Die hierbei zulässige Schachtelungstiefe ist 20. Darüber hinaus erfolgt die Fehlermeldung "ÜBERLAUF PROGRAMMSCHACHTELUNG". Der rekursive Aufruf von Unterprogrammen oder Funktionen ist erlaubt. Das heißt, ein Unterprogramm oder eine Funktion kann sich selbst wieder aufrufen.

**DEF** Alle Unterprogramme werden genau wie die Hauptprogramme mit der **DEF**-Vereinbarung plus Namen deklariert und mit **END** abgeschlossen, z.B.:

```
DEF UNTERPR ( )
...
END
```



**Abb. 34** Unterprogrammaufruf und Rücksprung

**DEFFCT** Eine Funktion ist eine Art Unterprogramm, jedoch ist der Programmname gleichzeitig eine Variable eines bestimmten Datentyps. Damit läßt sich das Ergebnis der Funktion durch einfache Wertzuweisung an eine Variable übergeben. Bei der Vereinbarung von Funktionen mit dem speziellen Schlüsselwort **DEFFCT** muß daher neben dem Namen der Funktion auch der Datentyp der Funktion angegeben werden. Abgeschlossen wird eine Funktion mit **ENDFCT**. Da eine Funktion einen Wert übergeben soll, muß dieser Wert vor der **ENDFCT**-Anweisung mit der **RETURN**-Anweisung spezifiziert werden. Beispiel:

```
DEFFCT INT FUNKTION( )
...
RETURN( X )
ENDFCT
```

**lokal** Grundsätzlich unterscheidet man zwischen lokalen und globalen Unterprogrammen bzw. Funktionen. Bei lokalen Unterprogrammen oder Funktionen befinden sich das Hauptprogramm und die Unterprogramme/Funktionen in der selben SRC-Datei. Die Datei trägt den Namen des Hauptprogrammes. Dabei steht das Hauptprogramm im Quelltext immer an erster Stelle, während die Unterprogramme und Funktionen in beliebiger Reihenfolge und Anzahl nach dem Hauptprogramm folgen.

**global** Lokale Unterprogramme/Funktionen können nur innerhalb des SRC-Files, indem sie programmiert wurden, aufgerufen werden. Sollen Unterprogramm-/Funktionsaufrufe auch von anderen Programmen möglich sein, so müssen sie global sein, also in einem eigenen SRC-File abgespeichert werden. Somit ist jedes Programm ein Unterprogramm, wenn es von einem anderen Programm (Hauptprogramm, Unterprogramm oder Funktion) aufgerufen wird.



- In lokalen Unterprogrammen bzw. Funktionen sind alle in der Datenliste des Hauptprogramms deklarierten Variablen bekannt. Variablen die im Hauptprogramm (SRC-File) deklariert wurden sind sogenannte "Runtime Variablen" und dürfen nur im Hauptprogramm verwendet werden. Ein Versuch diese Variablen im Unterprogramm zu verwenden führt zu einer entsprechenden Fehlermeldung. In globalen Unterprogrammen oder Funktionen sind die im Hauptprogramm deklarierten Variablen nicht bekannt.
- Im Hauptprogramm sind die in globalen Unterprogrammen oder Funktionen deklarierten Variablen nicht bekannt.
- Ein Hauptprogramm kann nicht auf lokale Unterprogramme oder Funktionen eines anderen Hauptprogramms zugreifen.
- Der Name von lokalen Unterprogrammen/Funktionen kann maximal 24 Zeichen lang sein. Bei globalen Unterprogrammen/Funktionen darf er höchstens 24 Zeichen lang sein (wegen der Dateierweiterungen).

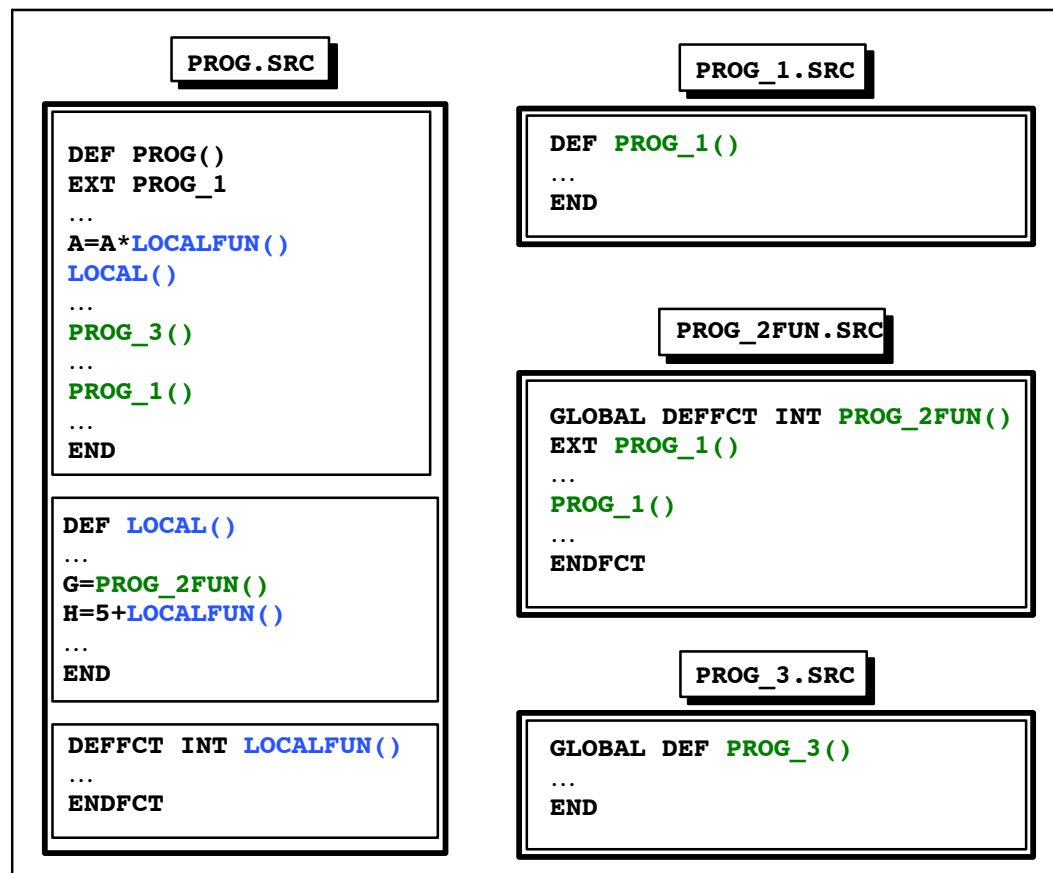
**GLOBAL** Damit das globale Unterprogramm dem Hauptprogramm bekannt ist, muß das Unterprogramm als **GLOBAL** definiert sein, z.B: `Global Def PROG_2( )`. Ist es nicht als **GLOBAL** definiert, müssen alle Namen und Pfade der aufzurufenden externen Unterprogramme und Funktionen sowie die verwendeten Parameter dem Compiler mit der **EXT**- bzw. **EXTFCT**-Vereinbarung kenntlich gemacht werden. Mit der Angabe der Parameterliste (s. 8.2) ist auch der benötigte Speicherplatz eindeutig festgelegt. Beispiele:

```
EXT PROG_3( )
EXTFCT FUNCTION(REAL:IN)
```



Möchte man ein Programm neu erstellen, wird man immer mit **GLOBAL** arbeiten. Die Möglichkeit mit **EXT** einem Hauptprogramm ein Unterprogramm bekanntzugeben ist hier aufgeführt, da in der KRL-Vorgängerversion dieses ausschließlich mit **EXT** möglich war, d.h. **GLOBAL** nicht existierte.

In Abb. 35 ist der Unterschied zwischen "normalen" globalen, "GLOBAL" globalen und lokalen Unterprogrammen bzw. Funktionen dargestellt: **PROG** ist ein Hauptprogramm, **LOCAL** ist ein lokales Unterprogramm und **LOCALFUN** eine lokale Funktion des Hauptprogrammes **PROG**. **PROG\_2FUN** und **PROG\_3** sind eine "GLOBAL" globale Funktion und ein "GLOBAL" globales Unterprogramm. **PROG\_1** ist eine "normal" globales Unterprogramm.



**Abb. 35** Unterschied zwischen lokalen, globalen und GLOBAL globalen Unterprogrammen

In dieser Abbildung ist zu erkennen, daß das Programm Prog\_1, daß nicht als GLOBAL vereinbart ist, im Hauptprogramm Prog mit EXT kenntlich gemacht werden muß. Die als GLOBAL vereinbarten Programme/Funktionen Prog\_2 und Prog\_3 hingegen können ohne EXT im Hauptprogramm aufgerufen werden

## 8.2 Aufruf und Parameterübergabe

Der Aufruf eines Unterprogrammes erfolgt einfach durch die Angabe des Unterprogramm-Namens und runden Klammern. Er sieht somit aus wie eine Anweisung (s. Kap. 2.1), z.B.:

```
UNTERPROG1 ( )
```

Ein Funktionsaufruf ist eine besondere Form einer Wertzuweisung. Eine Funktion kann daher nie alleine stehen, sondern der Funktionswert muß stets im Rahmen eines Ausdrucks einer Variablen vom gleichen Datentyp zugewiesen werden, z.B.:

```
INTVAR = 5 * INTFUNKTION( ) + 1
REALVAR = REALFUNKTION( )
```

Parameterli-  
ste

In lokalen Unterprogrammen und Funktionen sind alle in der Datenliste des Hauptprogramms deklarierten Variablen bekannt. In globalen Unterprogrammen dagegen, sind diese Variablen nicht bekannt, außer die Variablen wurden als GLOBAL vereinbart. Über eine Parameterliste können aber auch Werte an globale Unterprogramme und Funktionen übergeben werden.

Die Übergabe mit Parameterlisten ist oft auch in lokalen Unterprogrammen und Funktionen sinnvoll, da so eine klare Trennung zwischen Hauptprogramm und Unterprogramm/Funktion vorgenommen werden kann: Im Hauptprogramm (SRC-File) deklarierte Variablen werden nur dort verwendet, alle Übergaben in Unterprogramme und Funktionen (lokal und global) erfolgen mittels Parameterlisten. Durch diese strukturierte Programmierung reduzieren sich Programmierfehler deutlich.

Zur Parameterübergabe gibt es zwei unterschiedliche Mechanismen:

- **Call by value (IN)**  
Bei dieser Übergabeart wird ein **Wert** aus dem Hauptprogramm an eine Variable des Unterprogramms oder der Funktion übergeben. Der übergebene Wert kann eine Konstante, eine Variable, ein Funktionsaufruf oder ein Ausdruck sein. Bei unterschiedlichen Datentypen wird, wenn möglich, eine Typanpassung durchgeführt.
- **Call by reference (OUT)**  
Durch "Call by reference" wird nur die **Adresse** einer Variablen des Hauptprogramms an das Unterprogramm bzw. die Funktion übergeben. Das aufgerufene Unterprogramm bzw. die Funktion kann nun über einen eigenen Variablennamen den Speicherbereich überschreiben und somit auch den Wert der Variablen im Hauptprogramm verändern. Die Datentypen müssen daher identisch sein, eine Typanpassung ist in diesem Fall nicht möglich.

In Abb. 36 ist der Unterschied zwischen den beiden Methoden aufgezeigt. Während die Variable x bei "Call by value" im Hauptprogramm aufgrund der getrennten Speicherbereiche unverändert bleibt, wird sie bei "Call by reference" durch die Variable ZAHLE in der Funktion überschrieben.

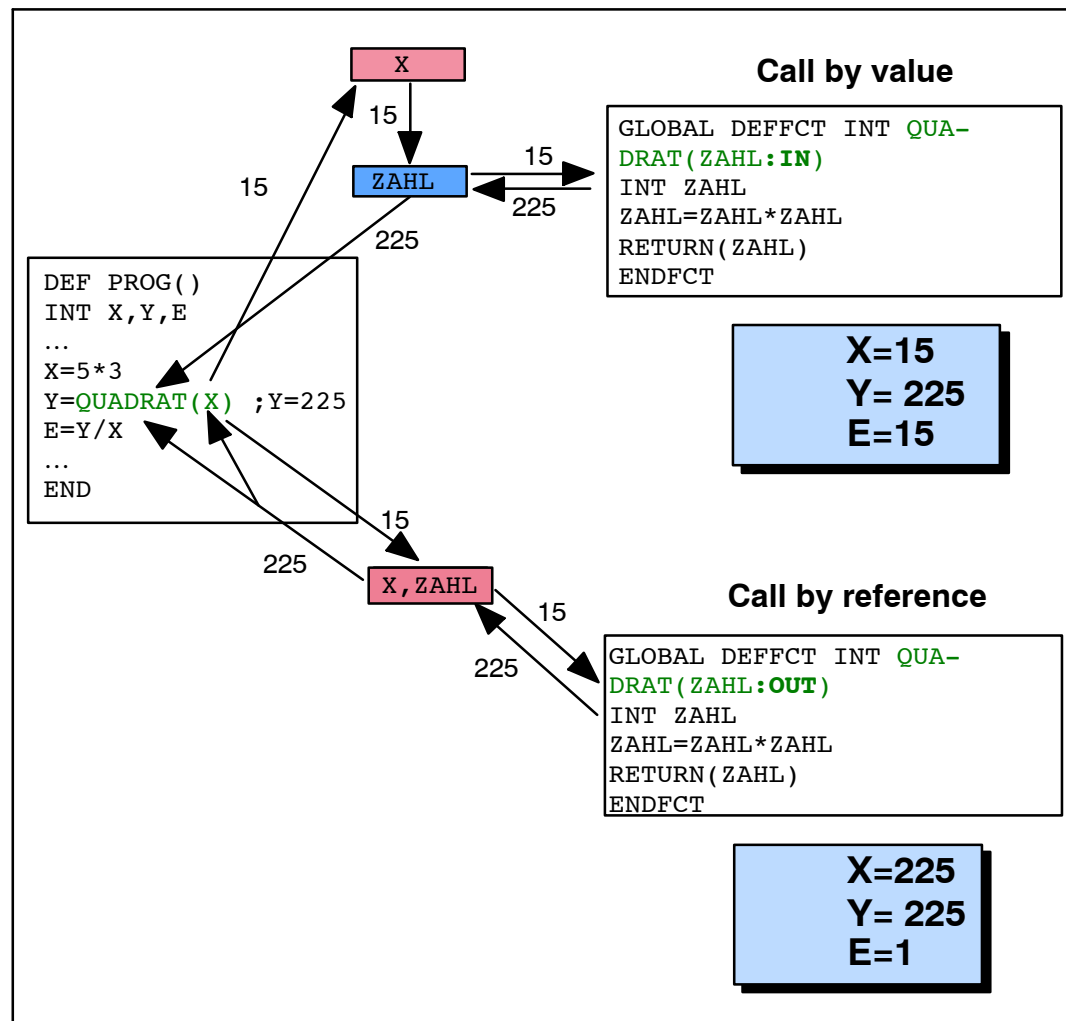


Abb. 36 Unterschied zwischen "Call by value" und "Call by reference"

"Call by value" wird im Unterprogramm- oder Funktionskopf durch das Schlüsselwort **IN** hinter jeder Variablen in der Parameterliste angegeben. "Call by reference" erhält man durch Angabe von **OUT**. **OUT** ist auch die Default-Einstellung. Beispiel:

```
DEF RECHNE (X:OUT, Y:IN, Z:IN, B)
```

Ist das aufzurufende globale Unterprogramm oder die globale Funktion nicht als **GLOBAL** deklariert, so muß bei der Extern-Vereinbarung im Hauptprogramm angegeben werden, welchen Datentyp die jeweiligen Variablen haben und welcher Übergabemechanismus verwendet werden soll. **OUT** ist wieder die Default-Einstellung. Beispiel:

```
EXTFCT REAL FUNKT1 (REAL:IN, BOOL:OUT, REAL, CHAR:IN)
```

Die Verwendung von **IN** und **OUT** soll mit folgendem Beispiel verdeutlicht werden. Das Unterprogramm und die Funktion seien global.



```

DEF PROG ( )
EXT RECHNE (INT:OUT,INT:IN,INT:IN)
EXTFCT REAL FUNKT1 (REAL:IN,REAL:OUT,REAL:OUT,REAL:IN,REAL:OUT)
INT A,B,C
REAL D,E,F,G,H,X

A = 1
B = 2
C = 3
D = 1
E = 2
F = 3
G = 4
H = 5

RECHNE (A,B,C)
    ;A ist nun 11
    ;B ist nun 2
    ;C ist nun 3

X = FUNKT1(H,D,E,F,G)
    ;D ist nun 3
    ;E ist nun 8
    ;F ist nun 3
    ;G ist nun 24
    ;H ist nun 5
    ;X ist nun 15
END

DEF RECHNE(X1:OUT,X2:IN,X3:IN)                                ;globales UP
INT X1,X2,X3
X1=X1+10
X2=X2+10
X3=X3+10
END

DEFFCT REAL FUNKT1(X1:IN,X2:OUT,X3:OUT,X4:IN,X5:OUT);globale Fkt.
REAL X1,X2,X3,X4,X5
X1 = X1*2
X2 = X2*3
X3 = X3*4
X4 = X4*5
X5 = X5*6
RETURN(X4)
ENDFCT

```

Bei der Übergabe eines Feldes muß das Feld im Unterprogramm oder der Funktion ebenfalls neu deklariert werden, jedoch ohne Index. Dazu folgendes Beispiel, indem die Werte eines Feldes x[ ] verdoppelt werden (die Funktion ist global):





```

DEF  ARRAY ( )
EXT  BAS (BAS_COMMAND:IN,REAL:IN)
INT  X[5]      ;feldvereinbarung
INT  I
EXT  DOPPEL (INT[:OUT)

BAS (#INITMOV,0)

FOR I=1 TO 5
    X[I]=I      ;array X[] initialisieren
ENDFOR        ;X[1]=1,X[2]=2,X[3]=3,X[4]=4,x[5]=5

DOPPEL (X[]) ;unterprogramm mit feldparameter aufrufen
                ;X[1]=2,X[2]=4,X[3]=6,X[4]=8,X[5]=10
END

DEF  DOPPEL (A[:OUT)
INT A[]      ;erneute vereinbarung des feldes
INT I
FOR I=1 TO 5
    A[I]=2*A[I] ;verdoppelung der feldwerte
ENDFOR
END

```

Bei der Übergabe von mehrdimensionalen Felder werden ebenfalls keine Indizes angegeben, allerdings muß die Dimension des Feldes durch Angabe von Kommas spezifiziert werden, Beispiele:

A[ , ]        für zweidimensionale Felder  
A[ , , ]      für dreidimensionale Felder



## NOTIZEN:



## 9 Interrupt-Behandlung

Beim Einsatz von Robotern in komplexen Fertigungsanlagen besteht die Notwendigkeit, daß der Roboter auf bestimmte externe oder interne Ereignisse gezielt und sofort reagieren kann und parallel zum Roboterprozeß Aktionen ausgeführt werden können. Das heißt, ein laufendes Roboterprogramm muß unterbrochen und ein Unterbrechungsprogramm bzw. -funktion gestartet werden. Nach Abarbeitung des Unterbrechungsprogramms soll, wenn nichts weiteres vereinbart ist, das unterbrochene Roboterprogramm wieder fortgesetzt werden.

Dieses gezielte Unterbrechen bzw. Starten eines Programms wird durch die Interrupt-Anweisung ermöglicht. Damit hat der Anwender die Möglichkeit, per Programm auf ein Ereignis zu reagieren, welches zeitlich nicht synchron zum Programmablauf auftritt.

Interrupts können von

- Geräten, wie Sensoren, Peripherieeinheiten, etc.,
- Fehlermeldungen
- dem Anwender oder durch
- Sicherheitsschaltungen

ausgelöst werden. Beispielsweise kann nach Drücken des Not-Aus-Schalters eine Interrupt-Routine aufgerufen werden, die bestimmte Ausgangssignale zurücksetzt (vorbereitetes Programm `IR_STOPM.SRC`).



NOTIZEN:

## 9.1 Deklaration

Bevor ein Interrupt aktiviert werden kann, müssen zunächst einmal die möglichen Unterbrechungsursachen und die jeweilige Reaktion des Systems darauf definiert werden.

**INTERRUPT** Dies geschieht mit der Interrupt-Deklaration, bei der sie jeder Unterbrechung eine Priorität, ein Ereignis und die aufzurufende Interrupt-Routine zuordnen müssen. Die vollständige Syntax lautet:

**INTERRUPT DECL** Priorität **WHEN** Ereignis **DO** Unterprogramm

Zur Bedeutung der Argumente siehe Tab. 13.

Argument	Datentyp	Bedeutung
Priorität	INT	Arithmetischer Ausdruck, der die Priorität der Unterbrechung angibt. Es stehen die Prioritätsebenen 1...39 und 81...128 zur Verfügung. Die Werte 40...80 sind für eine automatische Prioritätsvergabe durch das System reserviert. Die Unterbrechung der Ebene 1 hat die höchste Priorität.
Ereignis	BOOL	Logischer Ausdruck, der das Unterbrechungseignis definiert. Zugelassen sind: <ul style="list-style-type: none"> <li>eine boolsche Konstante</li> <li>eine boolsche Variable</li> <li>ein Signalname</li> <li>ein Vergleich</li> </ul>
Unterprogramm		Name des Interruptprogramms, das bei Auftreten des Ereignisses abgearbeitet werden soll.

**Tab. 13** Argumente in der Interrupt-Deklaration

### Die Anweisung

```
INTERRUPT DECL 4 WHEN $IN[3]==TRUE DO UP1()
```

deklariert z.B. einen Interrupt der Priorität 4, der das Unterprogramm UP1 ( ) aufruft sobald der Eingang 3 auf High geht.

Die Interrupt-Deklaration ist eine Anweisung. Sie darf daher nicht im Vereinbarungsteil stehen!

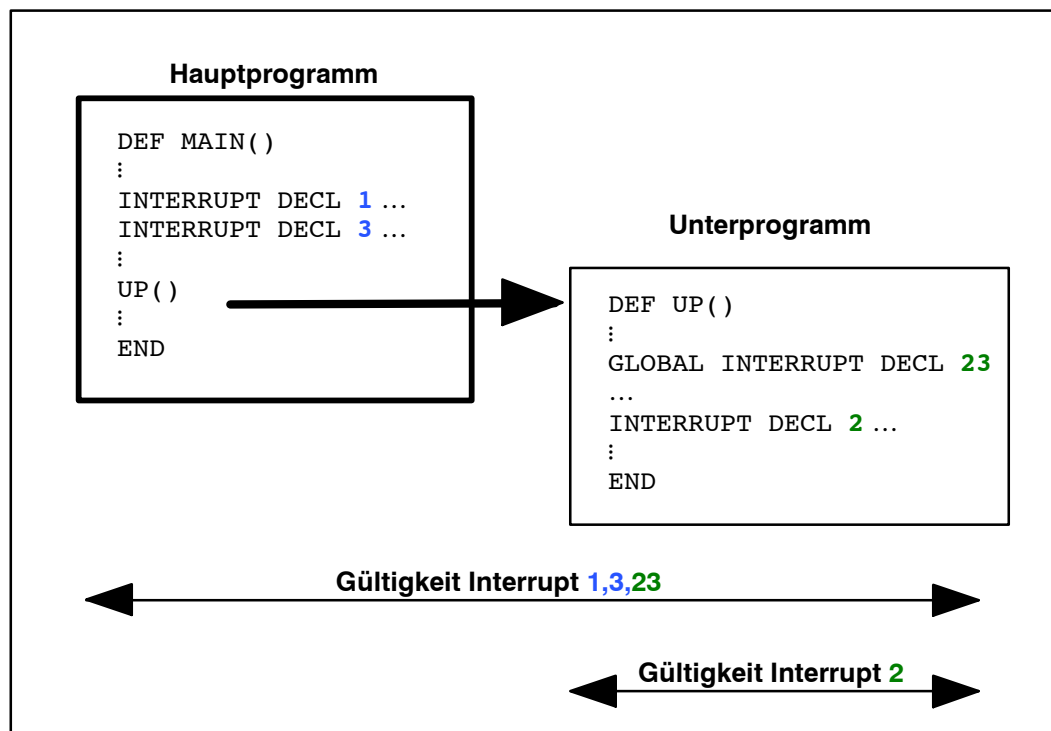
Ein Interrupt wird erst ab der Programmebene erkannt, in der er deklariert ist. In darüber liegenden Programmebenen wird der Interrupt trotz Aktivierung nicht erkannt. Das heißt, ein in einem Unterprogramm deklarierter Interrupt, ist im Hauptprogramm nicht bekannt (s. Abb. 37).

**GLOBAL** Wird der Interrupt hingegen als GLOBAL vereinbart so kann er in einem beliebigen Unterprogramm deklariert werden und verliert seine Gültigkeit beim Verlassen dieser Ebene nicht (s. Abb. 37).

```
GLOBAL INTERRUPT DECL 4 WHEN $IN[3]==TRUE DO UP1()
```



- Eine Deklaration kann jederzeit durch eine neue überschrieben werden.
- Ein GLOBALER Interrupt unterscheidet sich von einem normalen Interrupt dadurch, daß er nach Verlassen des Unterprogramms in dem er vereinbart wurde weiterhin gültig ist.
- Zur gleichen Zeit dürfen maximal 32 Interrupts deklariert sein.
- In der Interruptbedingung darf nicht auf Strukturvariablen oder –komponenten zugegriffen werden.
- Es dürfen keine Laufzeitvariablen als Parameter der Interruptroutine übergeben werden, außer es handelt sich um GLOBAL oder in der Datenliste vereinbarte Variablen.



**Abb. 37** Gültigkeitsbereiche eines Interrupts abhängig vom Deklarationsort und -art



**NOTIZEN:**

## 9.2 Aktivieren von Interrupts

Interrupt einschalten	<p>Nach der Deklaration ist ein Interrupt zunächst ausgeschaltet. Mit der Anweisung <code>INTERRUPT ON 4</code> wird der Interrupt mit der Priorität 4, mit <code>INTERRUPT ON</code> werden alle Interrupts eingeschaltet. Erst nach dem Einschalten kann eine Reaktion auf die definierte Unterbrechung erfolgen. Das Unterbrechungsereignis wird nun zyklisch überwacht.</p>
flankengetriggert	<p>Die Überprüfung des Ereignisses erfolgt dabei flankengetriggert, das heißt, ein Interrupt wird nur ausgelöst, wenn die logische Bedingung vom Zustand <code>FALSE</code> in den Zustand <code>TRUE</code> wechselt, nicht jedoch, wenn die Bedingung beim Einschalten bereits <code>TRUE</code> war.</p>

Aus Rechenzeitgründen dürfen zur gleichen Zeit nur 16 Interrupts eingeschaltet sein. Dies ist insbesondere bei der globalen Aktivierung aller Interrupts zu beachten.

Interrupt ausschalten	<p>In gleicher Weise wie das Einschalten funktioniert auch das Ausschalten einzelner oder aller Interrupts:</p> <p><code>INTERRUPT OFF 4</code></p> <p>oder</p> <p><code>INTERRUPT OFF</code></p>
-----------------------	---

Sperren / Freigeben	<p>Mit den Schlüsselworten <code>ENABLE</code> und <code>DISABLE</code> können eingeschaltete Interrupts einzeln oder global freigegeben oder gesperrt werden.</p> <p>Die Sperranweisung ermöglicht einen Schutz von bestimmten Programmteilen vor Unterbrechung. Ein gesperrter Interrupt wird zwar erkannt und gespeichert, aber nicht ausgeführt. Erst wenn eine Freigabe erfolgt, werden die aufgetretenen Interrupts in der Reihenfolge ihrer Priorität bearbeitet.</p>
---------------------	--

`DISABLE 4`

oder

`DISABLE`

Auf ein gespeichertes Ereignis wird nicht mehr reagiert, wenn der Interrupt vor der Auslösung ausgeschaltet wird. Falls ein Interrupt während er gesperrt ist mehrmals auftritt, wird er nach seiner Freigabe nur einmal ausgeführt.

Die Voraussetzungen für das Auslösen eines Interrupts sind:

- Der Interrupt muß deklariert sein (`INTERRUPT DECL ...`)
- Der Interrupt muß eingeschaltet sein (`INTERRUPT ON`)
- Der Interrupt darf nicht gesperrt sein
- Das zugehörige Ereignis muß eingetreten sein (flankengetriggert)

Priorität	<p>Bei gleichzeitigem Auftreten von Interrupts wird erst der Interrupt höchster Priorität bearbeitet, dann die Interrupts niedriger Priorität. Die Prioritätsstufe 1 hat dabei die höchste Priorität, Stufe 128 die niedrigste.</p>
-----------	---

Bei Erkennen eines Ereignisses wird die aktuelle Ist-Position des Roboters gespeichert und die Interrupt-Routine aufgerufen. Der aufgetretene Interrupt sowie alle Interrupts niedriger Priorität werden für die gesamte Dauer der Abarbeitung gesperrt. Bei der Rückkehr aus dem Interruptprogramm wird die sogenannte implizite Sperre, auch für den aktuellen Interrupt, wieder aufgehoben.

Der Interrupt kann nun also bei erneutem Auftreten (auch während des Interruptprogramms) wieder abgearbeitet werden. Soll dies verhindert werden, so muß der Interrupt explizit vor dem Rücksprung gesperrt oder ausgeschaltet werden.

Ein Interrupt kann nach dem ersten Befehl im Interruptprogramm durch Interrupts höherer Priorität unterbrochen werden. Im ersten Befehl hat der Programmierer somit die Möglichkeit, dies durch Sperren/Ausschalten eines oder aller Interrupts zu verhindern. Schaltet sich ein Interrupt im Interruptprogramm selbst ab, so wird dieses natürlich zu Ende abgearbeitet.

Nach Beendigung eines höher priorien Interrupts wird das unterbrochene Interruptprogramm an der Stelle fortgesetzt, an der es unterbrochen wurde.



- An ein Interruptprogramm können IN-Parameter übergeben werden.
- Soll ein lokales Interruptprogramm einen Parameter zurückliefern, so muß die Variable in der Datenliste des Hauptprogramms deklariert sein. Bei globalen Interruptprogrammen muß mit der Datenliste \$CONFIG.DAT gearbeitet werden.
- Änderungen von \$TOOL und \$BASE im Interruptprogramm sind nur dort wirksam (Kommandobetrieb).
- Im Interruptprogramm gibt es keinen Rechnervorlauf, da es auf Kommandoebene läuft, d.h. es wird Satz für Satz abgearbeitet ( $\Rightarrow$  \$ADVANCE-Zuweisungen sind nicht zulässig). Ein Überschleifen ist somit nicht möglich.

#### Sonderfälle:

- Interrupts auf die Systemvariablen \$ALARM\_STOP und \$STOPMESS werden auch im Fehlerfall bearbeitet, das heißt, trotz Roboterstop laufen die Interruptanweisungen ab (keine Bewegungen).
- Während eines Bedienstopps kann jeder deklarierte und aktivierte Interrupt erkannt werden. Nach einem erneuten Start werden die aufgetretenen Interrupts ihrer Priorität entsprechend abgearbeitet (falls sie freigegeben sind) und anschließend wird das Programm fortgesetzt.

Eine laufende Roboterbewegung wird beim Aufruf eines Interruptprogramms nicht abgebrochen. Während das Interruptprogramm bearbeitet wird, werden noch alle im unterbrochenen Programm aufbereiteten Bewegungen abgefahren. Wenn das Interruptprogramm während dieser Zeit komplett abgearbeitet ist, wird das unterbrochene Programm ohne Bewegungsstillstand, d.h. ohne Verlängerung der Bearbeitungszeit, fortgesetzt. Ist dagegen die Interruptaktion noch nicht beendet, bleibt der Roboter stehen bis nach dem Rücksprung die nächste Bewegung aufbereitet und fortgesetzt wird.

Befinden sich im Interruptprogramm selbst Bewegungsanweisungen, so hält das Interruptprogramm solange auf der ersten Bewegungsanweisung an bis der Vorlauf des Hauptprogramm abgearbeitet ist.

Den Einsatz von Interruptanweisungen und die Verwendung der speziellen Systemvariablen soll das folgende Beispiel erläutern. Hierin werden während einer Linearbewegung ständig zwei Sensoren (an den Eingängen 1 und 2) überwacht. Sobald ein Sensor einen Teil detektiert (auf High geht), wird ein Interruptunterprogramm aufgerufen, indem die Position des Teiles gespeichert wird und als Anzeige ein entsprechender Ausgang gesetzt wird. Die Roboterbewegung wird dabei nicht unterbrochen. Danach werden die erkannten Teile nochmals angefahren.



```

DEF  INTERRUPT ( )

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

;----- Initialisierung -----
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
FOR I=1 TO 16
    $OUT[I]=FALSE ;alle Ausgaenge ruecksetzen
ENDFOR
INTERRUPT DECL 10 WHEN $IN[1]==TRUE DO SAVEPOS (1 )
INTERRUPT DECL 11 WHEN $IN[2]==TRUE DO SAVEPOS (2 )

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

PTP  {X 1320,Y 100,Z 1000,A -13,B 78,C -102}

INTERRUPT ON ;alle Interrupts aktivieren
LIN  {X 1320,Y 662,Z 1000,A -13,B 78,C -102} ;Suchstrecke
INTERRUPT OFF 10 ;Interrupt 10 ausschalten
INTERRUPT OFF 11 ;Interrupt 11 ausschalten

PTP  HOME

FOR I=1 TO 2
    IF $OUT[I] THEN
        LIN  TEIL[I] ; erkanntes Teil anfahren
        $OUT[I]=FALSE
        PTP  HOME
    ENDIF
ENDFOR

END

;----- Interruptprogramm -----
DEF  SAVEPOS (NR :IN ) ;Teil erkannt
INT NR
$OUT[NR]=TRUE          ;Merker setzen
TEIL[NR]=$POS_INT      ;Position speichern
END
    
```



Neben dem Basis-Paket (BAS.SRC) ist standardmäßig auch eine Datei IR\_STOPM( ) auf der Steuerung vorhanden. Dieses Unterprogramm führt im Fehlerfall einige grundlegende Anweisungen aus. Dazu gehört neben einigen technologiespezifischen Aktionen auch das Rückpositionieren des Roboters auf die Bewegungsbahn. Während nämlich der Roboter nach Drücken des NOT-AUS-Schalters auf der Bahn bleibt, wird bei Schutzeinrichtungen, die direkt den Bediener betreffen (z.B. Schutztür), hardwareseitig ein nicht bahntreuer Stop ausgelöst.

Sie sollten daher im Initialisierungsteil Ihrer Programme stets die folgende Sequenz implementieren (befindet sich standardmäßig im Fold BAS INI):

```

INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT  ON 3
    
```



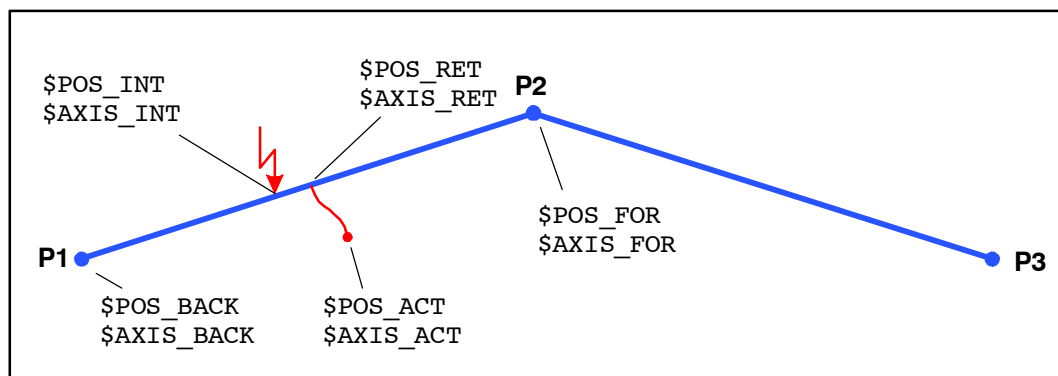
In der Datei `IR_STOPM( )` wird mit der Anweisung `PTP $POS_RET` das Rückpositionieren veranlaßt und somit wieder SAK hergestellt.

Weitere nützliche Systemvariablen für das Arbeiten mit Interrupts sind in Tab. 14 aufgeführt. Die Positionen beziehen sich immer auf die im Hauptlauf aktuellen Koordinatensysteme.

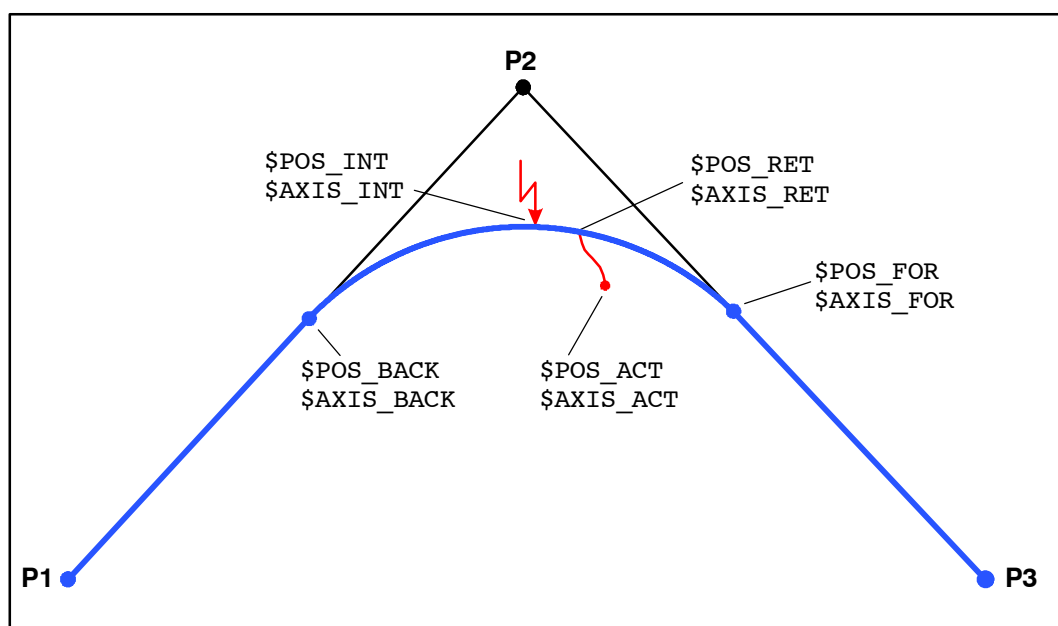
achsspezifisch	kartesisch	Beschreibung
<code>\$AXIS_INT</code>	<code>\$POS_INT</code>	Position, an der der Interrupt ausgelöst wurde
<code>\$AXIS_ACT</code>	<code>\$POS_ACT</code>	aktuelle IST-Position
<code>\$AXIS_RET</code>	<code>\$POS_RET</code>	Position, an der die Bahn verlassen wurde
<code>\$AXIS_BACK</code>	<code>\$POS_BACK</code>	Position des Startpunktes der Bahn
<code>\$AXIS_FOR</code>	<code>\$POS_FOR</code>	Position des Zielpunktes der Bahn

**Tab. 14** Nützliche Systemvariablen bei der Interrupt-Behandlung

Die Positionen `..._BACK` und `..._FOR` sind bei Überschiefbewegungen davon abhängig wo sich der Hauptlauf befindet. Siehe dazu Abb. 38 bis Abb. 39.



**Abb. 38** Interrupt-Systemvariablen bei Genauhaltpunkten



**Abb. 39** Interrupt-Systemvariablen bei Interrupt in einem Überschiefbereich

### 9.3 Laufende Bewegungen anhalten

**BRAKE** Sollen noch laufende Roboterbewegungen beim Eintreten eines Interrupts angehalten werden, so bedient man sich der **BRAKE**-Anweisung im Interruptprogramm. Programmiert man ohne Parameter

**BRAKE**

so bewirkt dies ein Abbremsen der Bewegung mit den programmierten Bahn- bzw. Achsbeschleunigungswerten. Das Verhalten ist das gleiche wie beim Betätigen der STOP-Taste. Die programmierte Bewegungsbahn wird dabei nicht verlassen.

Mit der Anweisung

**BRAKE F**

(brake fast) erreichen Sie ein kürzeren Bremsweg. Der Roboter wird dabei mit der maximalen bahntreuen Verzögerung abgebremst.

Die **BRAKE** - Anweisung darf nur in einem Interruptprogramm stehen. In anderen Programmen führt sie zu einem Fehlerstop.

Die **BRAKE**-Anweisung muß nicht direkt nach dem Aufruf, sondern kann an beliebiger Stelle im Interruptprogramm erfolgen. Ihre Wirkung hängt davon ab, ob zum Zeitpunkt ihrer Abarbeitung noch eine Bewegung des unterbrochenen Programms ausgeführt wird. Wenn der Roboter steht, hat die Anweisung keine Auswirkung. Eine noch laufende Bewegung des unterbrochenen Programms wird mit dem programmierten Bremsmodus angehalten. Der **BRAKE**-Befehl ersetzt allerdings nicht die **HALT**-Anweisung, wenn der Programmlauf angehalten werden soll. Die Bearbeitung des Interruptprogramms wird erst nach erfolgtem Stillstand des Roboters mit der darauffolgenden Anweisung fortgesetzt.



**Nach dem Rücksprung in das unterbrochene Programm wird eine im Interruptprogramm mit **BRAKE** oder **BRAKE F** angehaltene Bewegung fortgesetzt!**



#### NOTIZEN:

## 9.4 Abbrechen von Interrupt-Routinen

In Beispiel 8.1 werden während der Roboterbewegung durch 2 Initiatoren maximal 2 Gegenständen detektiert und deren Positionen zum späteren Anfahren aufgezeichnet.

Die Suchstrecke wird dabei auf jeden Fall komplett abgefahren, auch wenn die beiden Gegenstände schon erkannt sind. Um Zeit zu sparen, ist es wünschenswert die Bewegung sofort abubrechen, wenn die maximale Anzahl von Teilen erkannt wurde.

**RESUME** Das Abbrechen einer Roboterbewegung ist in der KR C1 mit der  
RESUME

Anweisung möglich. RESUME bricht alle laufenden Interruptprogramme und auch alle laufenden Unterprogramme bis zu der Ebene ab, in der der aktuelle Interrupt deklariert wurde.

Wie die BRAKE-Anweisung, ist auch RESUME nur in einem Interruptprogramm zulässig. Zum Zeitpunkt der RESUME-Anweisung darf der Vorlaufzeiger nicht in der Ebene stehen, in der der Interrupt deklariert wurde, sondern mindestens eine Ebene tiefer.

Da die Suchstrecke mit RESUME abgebrochen werden soll, muß die Suchbewegung in einem Unterprogramm programmiert werden. Dies geschieht im folgenden Beispiel in MOVEP ( ), das Interruptunterprogramm heißt IR\_PROG ( ).

Wichtig ist, daß in Unterprogrammen, die mit RESUME abgebrochen werden sollen, vor der letzten Zeile der Vorlauf angehalten wird. Nur dann ist gewährleistet, daß der Vorlaufzeiger bei RESUME nicht in der Ebene steht, in welcher der Interrupt deklariert wurde. In MOVEP ( ) wurde dies mit der \$ADVANCE=0 Zuweisung realisiert.

Im Interruptprogramm selbst wird – sobald 4 Teile durch einen Sensor an Eingang 1 erkannt sind – die Suchbewegung mittels BRAKE angehalten, und schließlich mit der RESUME-Anweisung abgebrochen (da neben IR\_PROG ( ) auch MOVEP ( ) beendet wird). Ohne die BRAKE-Anweisung würde zunächst noch die Suchbewegung im Vorlauf abgearbeitet werden.

Nach RESUME wird das Hauptprogramm mit der Anweisung nach dem Unterprogrammaufruf, also \$ADVANCE=3 (Vorlauf zurückstellen), fortgesetzt.



```

DEF SEARCH ( )
;----- Deklarationsteil -----
EXT BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
;----- Initialisierung -----
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3 ;standardmaessige Fehlerbehandlung
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
INTERRUPT DECL 11 WHEN $IN[1] DO IR_PROG ( )
I[1]=0 ;vordefinierten Zaehler auf 0 setzen
;----- Hauptteil -----
PTP HOME ;SAK-Fahrt
INTERRUPT ON 11
MOVEP ( ) ;Abfahren der Suchstrecke
$ADVANCE=3 ;Vorlauf zurueckstellen
INTERRUPT OFF 11
GRIP ( )
PTP HOME
END
;----- Unterprogramm -----
DEF MOVEP ( ) ;Unterprogramm zum Abfahren der Suchstrecke
PTP {X 1232,Y -263,Z 1000,A 0,B 67,C -90}
LIN {X 1232,Y 608,Z 1000,A 0,B 67,C -90}
$ADVANCE=0 ;Vorlauf anhalten
END ;
;----- Interruptprogramm -----
DEF IR_PROG ( ) ;Teileposition speichern
;INTERRUPT OFF 11
I[1]=I[1]+1
POSITION[I]=$POS_INT ;Abspeichern der Position
IF I[1]==4 THEN ;4 Teile werden erkannt
BRAKE ;Anhalten der Bewegung
RESUME ;Abbrechen von IR_PROG & MOVE
ENDIF
;INTERRUPT ON 11
END
;----- Unterprogramm -----|

DEF GRIP ( ) ;Greifen der erkannten Teile
INT POS_NR ;Zaehlvariable
FOR POS_NR=I[1] TO 1 STEP -1
POSITION[POS_NR].Z=POSITION[POS_NR].Z+200
LIN POSITION[POS_NR] ;200mm ueber Teil fahren
LIN_REL {Z -200} ;Teil senkrecht anfahren
; Teil greifen
LIN POSITION[POS_NR] ;wieder hoch fahren
LIN {X 634,Y 1085,Z 1147,A 49,B 67,C -90}
; Teil ablegen
ENDFOR
END

```



Wenn die Gefahr besteht, daß ein Interrupt fälschlicherweise durch empfindliche Sensorik zweimal ausgelöst wird ("Tastenprellen"), so können Sie dies durch Ausschalten des Interrupts in der ersten Zeile des Interruptprogramms verhindern. Allerdings kann dann auch ein tatsächlich auftretender Interrupt während der Interruptverarbeitung nicht mehr erkannt werden. Vor dem Rücksprung muß der Interrupt – wenn er weiter aktiv sein soll – wieder eingeschaltet werden.



Falls, wie in obigem Beispiel, eine Bewegung mit `RESUME` abgebrochen wird, sollte die nachfolgende Bewegung keine `CIRC`-Bewegung sein, da der Anfangspunkt jedes mal ein anderer ist ( $\Rightarrow$  unterschiedliche Kreise).

Bei der in Beispiel 9.2 programmierten Suchaktion werden die Eingänge im Interpolations-takt (z.Z. 12ms) abgefragt. Dabei besteht eine maximale Ungenauigkeit von ca. 12ms mal Bahngeschwindigkeit.

"Schnelles  
Messen"

Soll diese Ungenauigkeit vermieden werden, so darf man den Initiator nicht an die Anwenderingänge anschließen, sondern muß spezielle Eingänge (4 Stück) am Peripheriestecker X11 benutzen. Diese Eingänge können über die Systemvariablen `$MEAS_PULSE[ 1 ] ... MEAS_PULSE[ 4 ]` angesprochen werden (Reaktionszeit 125µs).

Beim Einschalten des Interrupts darf der Meßimpuls nicht anliegen, sonst erscheint die entsprechende Fehlermeldung.



---

**NOTIZEN:**

## 9.5 Verwendung zyklischer Flags

In der Anweisung zur Interrupt-Deklaration sind keine logischen Verknüpfungen zulässig. Um kompliziertere Ereignisse definieren zu können, müssen Sie daher mit zyklischen Flags arbeiten, da nur diese eine stetige Aktualisierung ermöglichen.

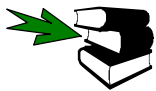
Mit der Sequenz

```

:
$CYCFLAG[3] = $IN[1] AND ([ $IN[2] OR $IN[3] )
INTERRUPT DECL 10 WHEN $CYCFLAG[3] DO IR_PROG()
INTERRUPT ON 10
:

```

können Sie somit gleichzeitig 3 Eingänge überwachen und logisch miteinander verknüpfen.



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



### NOTIZEN:

## 10 Trigger – Bahnbezogene Schaltaktionen

Im Gegensatz zu den bewegungsunabhängigen Interruptfunktionalitäten erfordern manche Anwendungsfälle auch Schaltaktionen, die in Abhängigkeit von der Bewegungsbahn ausgelöst werden. Solche Anwendungsfälle sind z.B.:

- Schließen bzw. Öffnen der Schweißzange beim Punktschweißen
- Ein-/Ausschalten des Schweißstromes beim Bahnschweißen
- Zu-/Abschalten des Volumenstromes beim Kleben oder Abdichten

In der KR C1 sind diese bahnbezogenen Schaltaktionen mit der TRIGGER-Anweisung möglich. Parallel zur nächsten Roboterbewegung kann mit TRIGGER in Abhängigkeit von einem Wegkriterium ein Unterprogramm abgearbeitet werden, eine Wertzuweisung an eine Variable oder eine PULSE-Anweisung erfolgen oder ein Ausgang gesetzt werden.

### 10.1 Schaltaktion am Start- oder Zielpunkt der Bahn

**TRIGGER** Ist eine Schaltaktion bezüglich des Start- oder Zielpunktes einer Bewegungsbahn gewünscht, so programmieren Sie vor der betreffenden Bewegungsanweisung (PTP, LIN oder CIRC) eine TRIGGER-Anweisung mit folgender Syntax:

**TRIGGER WHEN DISTANCE=Schaltpunkt DELAY=Zeit DO Anweisung**  
**<PRIO=Priorität>**

Die Argumente sind in folgender Tabelle näher beschrieben.

Argument	Datentyp	Bedeutung
Schaltpunkt	INT	Bei <b>Einzelsätzen</b> bezeichnet DISTANCE=0 den Startpunkt und DISTANCE=1 den Zielpunkt der nachfolgenden Bewegung. Bei <b>Überschleifsätzen</b> markiert die Angabe DISTANCE=1 die Mitte des nachfolgenden Überschleifbogens. Ist der Vorgängersatz bereits ein Überschleifsatz, so markiert DISTANCE=0 den Endpunkt des vorhergehenden Überschleifbogens.
Zeit	INT	Mit der DELAY-Angabe ist es möglich, den Schaltpunkt um eine bestimmte Zeit zu verzögern oder vorwegzunehmen. Der Schaltpunkt kann dabei aber immer nur soweit verschoben werden, daß er immer noch innerhalb des jeweiligen Satzes liegt. Die Einheit ist <b>Millisekunden</b> .
Anweisung		Die Anweisung kann <ul style="list-style-type: none"> <li>• ein Unterprogrammaufruf</li> <li>• eine Wertzuweisung an eine Variable</li> <li>• eine OUTPUT-Anweisung (auch Pulse) sein.</li> </ul>
Priorität	INT	Jeder TRIGGER-Anweisung mit Unterprogrammaufruf muß eine Priorität zugeordnet werden. Zulässig sind Werte von 1...39 und 81...128. Es handelt sich dabei um die selben Prioritäten wie bei Interrupts (s. Abschnitt 9). Die Werte 40...80 sind für eine automatische Prioritätsvergabe durch das System reserviert. Programmieren Sie dazu PRIO=-1.

**Tab. 15** Argumente in der TRIGGER-Anweisung

Mit der Anweisungsfolge

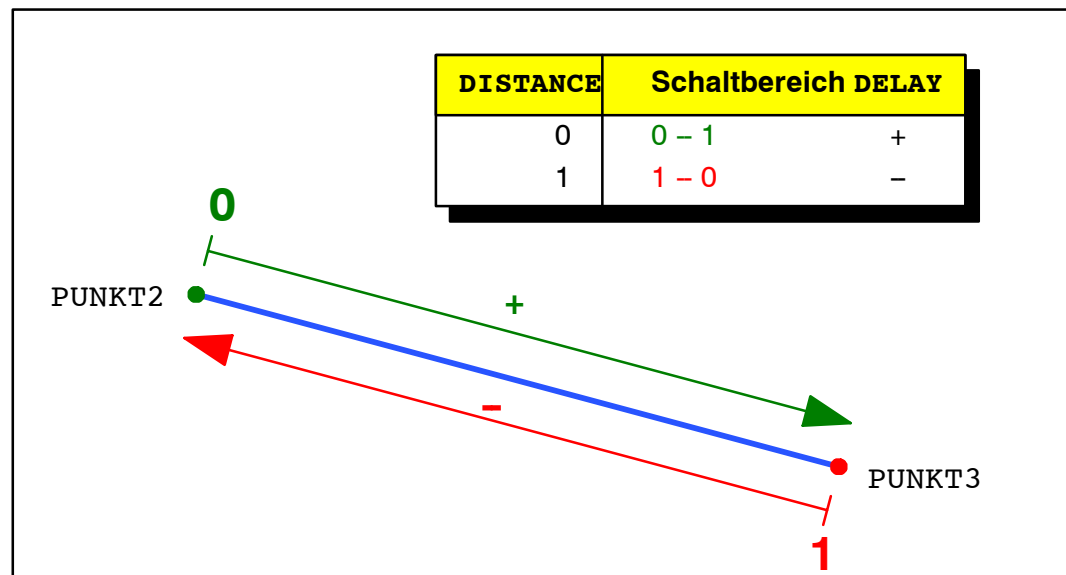
```

:
LIN PUNKT2
:
TRIGGER WHEN DISTANCE = 0 DELAY=20 DO $OUT[4]=TRUE
TRIGGER WHEN DISTANCE = 1 DELAY=-25 DO UP1() PRIO=-1
LIN PUNKT3
:
LIN PUNKT4
:

```

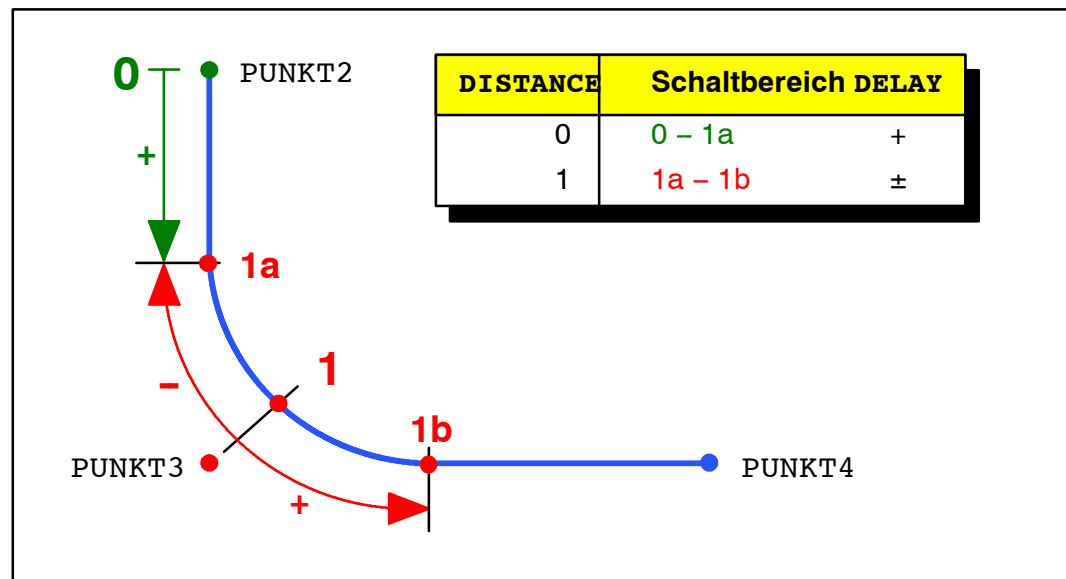
wird somit während der Linearbewegung zu PUNKT3 der Ausgang 4 um 20 Millisekunden nach dem Start der Bewegung gesetzt und 25 Millisekunden vor Erreichen des Zielpunktes das Unterprogramm UP1() aufgerufen. Die Prioritätsvergabe erfolgt automatisch durch das System.

Zur Erläuterung der unterschiedlichen Wirkung der DISTANCE-Angabe bei Einzel- und Überschleifsätzen siehe Abb. 40 – Abb. 43.

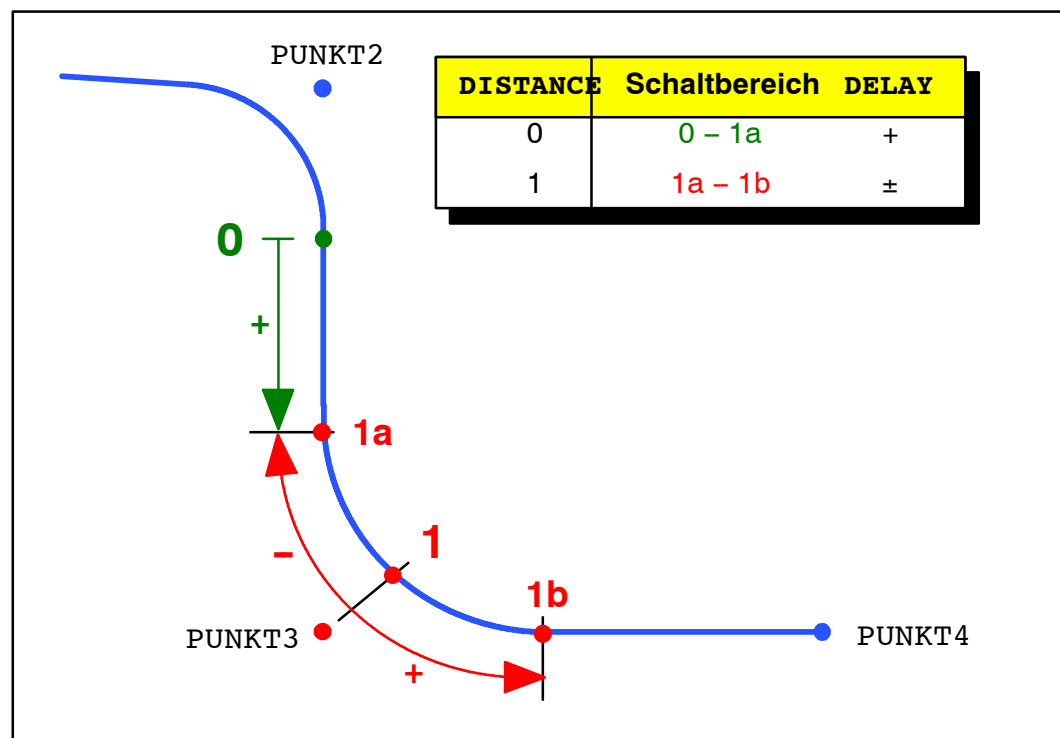


**Abb. 40** Schaltbereiche und mögliche Verzögerungswerte, wenn Start- und Zielpunkt Genauhaltpunkte sind.

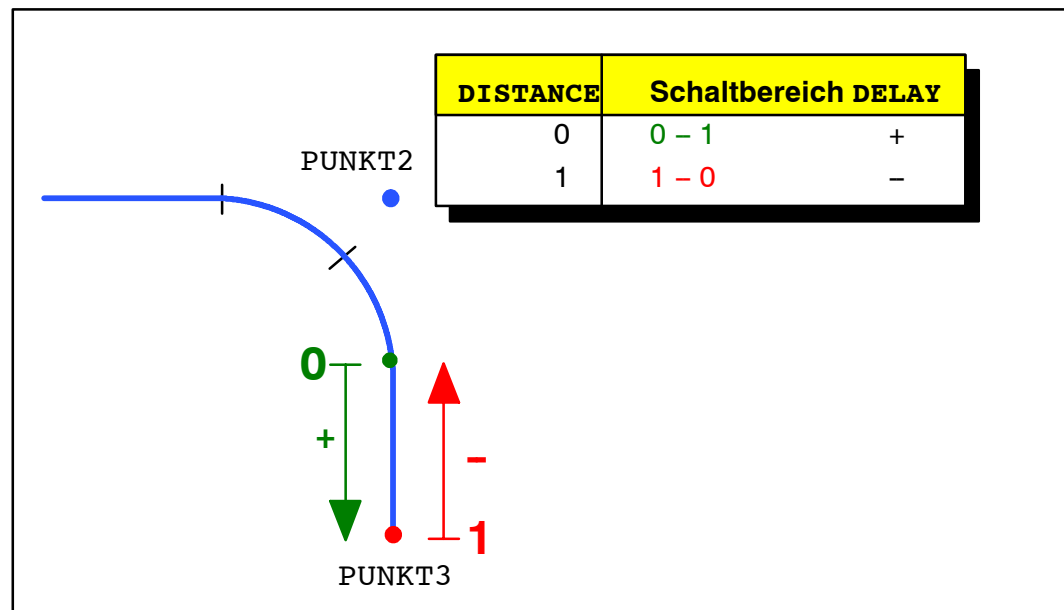




**Abb. 41** Schaltbereiche und mögliche Verzögerungswerte, wenn Startpunkt ein Genauhaltspunkt und Zielpunkt ein Überschleifpunkt ist.



**Abb. 42** Schaltbereiche und mögliche Verzögerungswerte, wenn Start- und Zielpunkt-Überschleifpunkte sind.



**Abb. 43** Schaltbereiche und mögliche Verzögerungswerte, wenn Startpunkt ein Überschleifpunkt und Zielpunkt ein Genauhaltpunkt ist.



NOTIZEN:

## 10.2 Schaltaktion beliebig auf der Bahn

Bei der wegbezogenen TRIGGER-Anweisung können Sie mit einer Entfernungsangabe die Schaltaktion an einer beliebigen Stelle auf der Bahn auslösen. Diese kann zusätzlich noch einmal – wie bei Schaltaktionen an Start- oder Zielpunkt – zeitlich verschoben werden.

Die wegbezogene Schaltaktion ist nur für Bahnbewegungen (LIN oder CIRC) erlaubt. Die TRIGGER-Anweisung bezieht sich dabei auf den nachfolgend programmierten Bewegungssatz, und hat die Syntax:

**TRIGGER WHEN PATH = Strecke DELAY = Zeit DO Anweisung (PRIO=Priorität**

Die Argumente sind in folgender Tabelle näher beschrieben.

Argument	Datentyp	Bedeutung
Strecke	INT	<p>Mit <b>Strecke</b> geben Sie die gewünschte Entfernung vom nach dem Trigger programmierten Zielpunkt an.</p> <p>Ist dieser Zielpunkt ein <b>überschliffener Punkt</b>, so gibt <b>Strecke</b> die gewünschte Entfernung der Schaltaktion von der dem Zielpunkt am nächsten liegenden Position des Überschleifbereichs an.</p> <p>Der Schalterpunkt kann durch eine negative <b>Strecke</b> bis zum Startpunkt vorgezogen werden. Ist der Startpunkt ein Überschleifpunkt, so kann der Schalterpunkt bis zum Anfang des Überschleifbereichs verschoben werden.</p> <p>Mit einer positiven Angabe von <b>Strecke</b> ist eine Verschiebung bis zum nächsten nach dem Trigger programmierten Genauhaltpunkt möglich.</p> <p>Die Einheit ist <b>Millimeter</b>.</p>
Zeit	INT	<p>Mit der <b>DELAY</b>-Angabe ist es möglich, den Schalterpunkt relativ zur <b>PATH</b>-Angabe um eine bestimmte Zeit zu verzögern (+) oder vorwegzunehmen (-).</p> <p>Der Schalterpunkt kann dabei aber immer nur im oben genannten Schaltbereich verschoben werden (soweit, daß er bis zum nächsten Genauhaltpunkt reicht. Bei Überschleifbewegungen kann der Schalterpunkt höchstens bis zum Überschleifbeginn des Startpunktes nach vorne versetzt werden).</p> <p>Die Einheit ist <b>Millisekunden</b>.</p>
Anweisung		<p>Die Anweisung kann</p> <ul style="list-style-type: none"> <li>• ein Unterprogrammaufruf</li> <li>• eine Wertzuweisung an eine Variable</li> <li>• eine OUTPUT-Anweisung (auch PULS) sein.</li> </ul>
Priorität	INT	<p>Jeder TRIGGER-Anweisung mit Unterprogrammaufruf muß eine Priorität zugeordnet werden. Zulässig sind Werte von 1...39 und 81...128. Es handelt sich dabei um die selben Prioritäten wie bei Interrupts (s. Abschnitt 9).</p> <p>Die Werte 40...80 sind für eine automatische Prioritätsvergabe durch das System reserviert. Programmieren Sie dazu <b>PRIO=-1</b>.</p>

**Tab. 16** Argumente in der TRIGGER-Anweisung

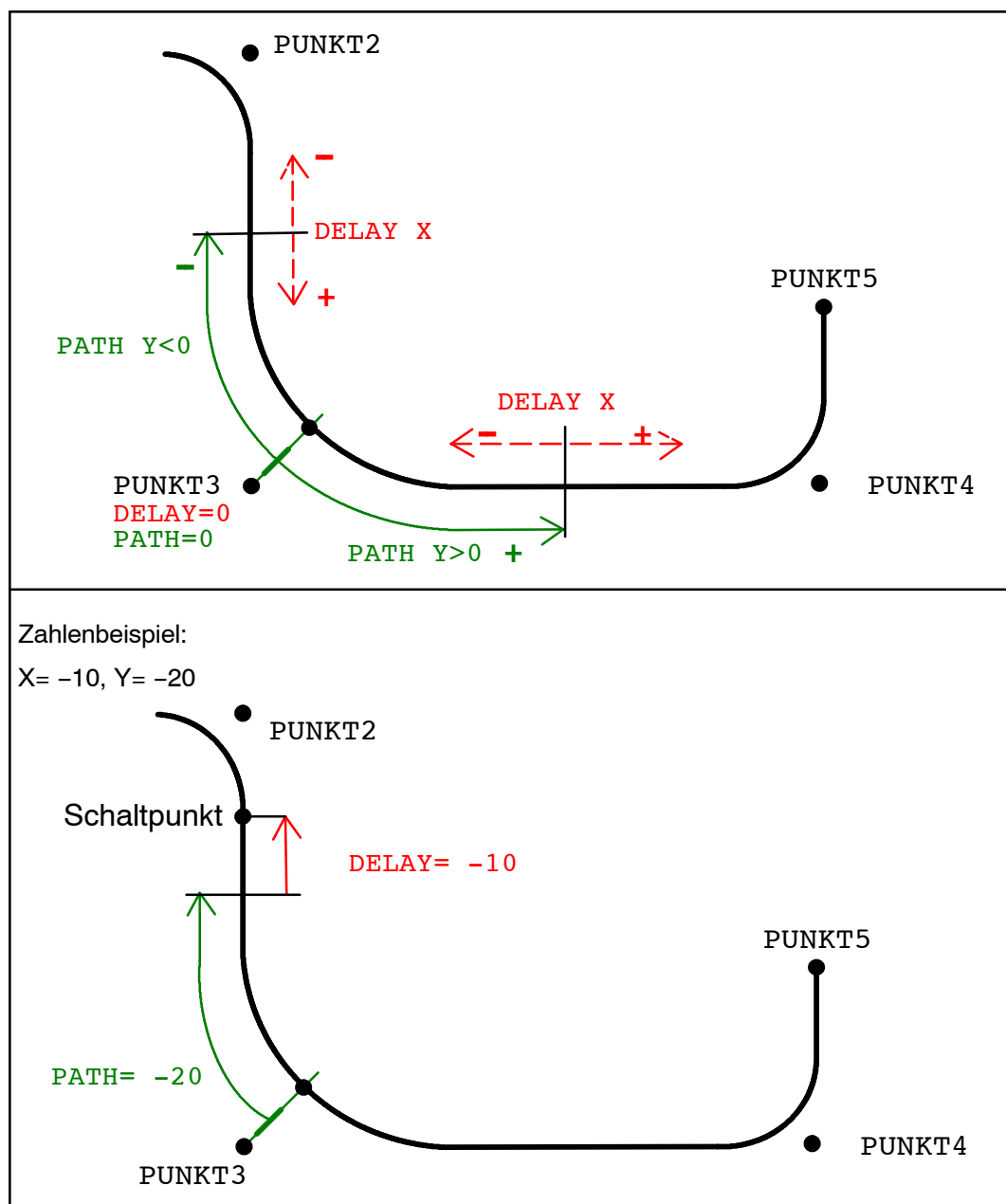
Anweisungsfolge:

```

:
LIN PUNKT2 C_DIS
TRIGGER WHEN PATH = Y DELAY= X DO $OUT[2]=TRUE
LIN PUNKT3 C_DIS
LIN PUNKT4 C_DIS
LIN PUNKT5
:

```

Da der Schalterpunkt ab dem Bewegungspunkt, vor dem er programmiert wurde, über alle nachfolgenden Überschleifpunkte hinweg, bis zum nächsten Genauhaltspunkt verschoben werden kann, ist eine Verschiebung des Schalterpunkts vom Überschleifanfang PUNKT2 bis zu PUNKT5 möglich. Wäre in dieser Anweisungsfolge PUNKT2 nicht überschleift, so würde der Schalterpunkt nur bis zum Genauhaltspunkt PUNKT2 verschoben werden können.



**Abb. 44** Schaltbereiche wenn Startpunkt ein Überschleifpunkt ist

**Sonderfälle:**

- **SAK–Fahrt**

Wird eine Satzanwahl auf eine Bahnbewegung durchgeführt, so findet diese als SAK–Fahrt statt. Da bei dieser SAK–Fahrt der Startpunkt beliebig ist, kann er kein sinnvoller Startpunkt für eine Entfernungsangabe sein. Sind vor einer solchen Bewegung also TRIGGER–Befehle mit PATH–Angabe programmiert, und findet eine Satzanwahl auf diese Befehle statt, so werden diese alle am Zielpunkt ausgeführt.

- **Überschleifen nicht möglich**

Ist ein Überschleifen nicht möglich, so findet an dieser Stelle eine Genauhaltbewegung statt. Diese wird in diesem Zusammenhang jedoch wie eine Überschleifbewegung behandelt. Im weiteren Verlauf auf der Bahn liegende Schaltaktionen bleiben gespeichert und werden an entsprechender Stelle ausgelöst. Allerdings werden sie in der Regel nicht mehr exakt stimmen, da sich jetzt eine andere Bahn und somit eine andere Bahnlänge ergibt. Schaltaktionen, die durch einen negativen PATH–Wert auf die erste Hälfte des Überschleifbereichs gelegt wurden, können jetzt frühestens am Genauhaltpunkt ausgelöst werden:

```

:
LIN P1 C_DIS
TRIGGER WHEN PATH=-120 DELAY=0 DO UP1( ) PRIO=-1
TRIGGER WHEN PATH=-70 DELAY=0 DO $OUT[2]=TRUE
LIN P2 C_DIS
:

```

Im obigen Beispiel soll die Entfernung zwischen Start– und Zielpunkt 100 mm betragen. Kann bei P1 überschleift werden, so wird der Unterprogrammaufruf UP1( ) 20 mm vor dem Erreichen des Bahnpunktes, der dem Überschleifpunkt P1 am nächsten ist, ausgeführt. Das Setzen von Ausgang 2 wird 30 mm nach diesem Bahnpunkt ausgeführt. Könnte die Überschleifbewegung bei P1 nicht ausgeführt werden, so läuft die Bahn durch den Punkt P1, in dem auch positioniert wird. Der Unterprogrammaufruf UP1( ) wird jetzt unmittelbar nach dem Verlassen von P1 ausgeführt, das Setzen von Ausgang 2 erfolgt in 30 mm Abstand von P1.

- **Abbrechen einer Bewegung**

Wird eine Bewegung z.B. durch Satzanwahl oder Reset abgebrochen und nicht mehr zu Ende geführt, so werden wie bei der DISTANCE–Angabe die noch nicht ausgeführten Schaltaktionen auch nicht mehr ausgeführt, sondern gelöscht.

- **Wegbezogenen TRIGGER –Anweisung für eine PTP –Bewegung**

Wird eine PATH–TRIGGER –Anweisung mit Wegangabe zu einer PTP –Bewegung programmiert, so wird dies vom Interpreter bei der Ausführung abgelehnt.

- **PTP–Bahnüberschleifen**

Wird eine PATH–TRIGGER –Anweisung, zu einer Bewegung programmiert, deren Startpunkt ein PTP–Bahn–Überschleifpunkt ist, so kann – da jetzt der ganze Überschleifbereich noch PTP gefahren wird – die Schaltaktion frühestens am Ende dieses Überschleifbereichs stattfinden.

Bei einem Bahn–PTP–Überschleifbereich werden alle noch aktiven TRIGGER –Anweisung, die bis dahin nicht geschaltet haben, am Anfangspunkt des Überschleifbereichs ausgelöst. Denn ab hier wird dann PTP gefahren und es ist keine Bahnzuordnung mehr möglich.

Im nächsten Beispiel sind sowohl Schaltaktionen mit **DISTANCE**-Angabe als auch mit **PATH**-Angabe programmiert. Die einzelnen Schaltpunkte und die Bewegungsbahn sind in Abb. 47 dargestellt.

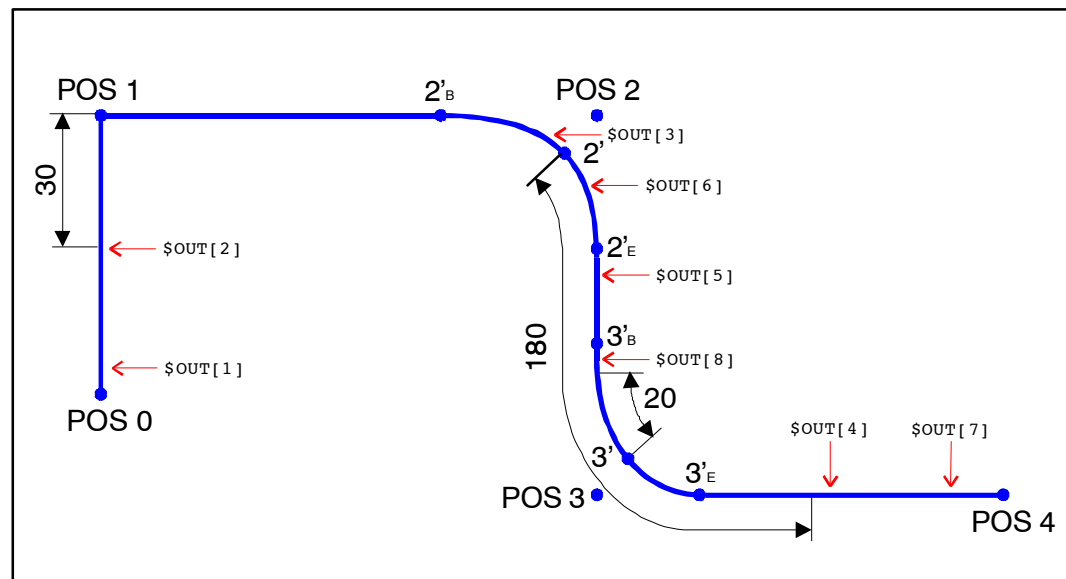


```

DEF TRIG ( )
;----- Deklarationsteil -----
EXT BAS (BAS_COMMAND :IN,REAL :IN)
DECL AXIS HOME
INT I
SIGNAL KLEBER $OUT[3]
;----- Initialisierung -----
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT ON 3
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
$APO.CDIS=35      ;Überschleifdistanz festlegen
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
POS0={POS: X 1564,Y -114,Z 713,A 128,B 85,C 22,S 6,T 50}
POS1={X 1383,Y -14,Z 713,A 128,B 85,C 22}
POS2={X 1383,Y 200,Z 713,A 128,B 85,C 22}
POS3={X 1527,Y 200,Z 713,A 128,B 85,C 22}
POS4={X 1527,Y 352,Z 713,A 128,B 85,C 22}
FOR I=1 TO 16
    $OUT[I]=FALSE
ENDFOR
;----- Hauptteil -----
PTP HOME ;SAK-Fahrt
LIN POS0
TRIGGER WHEN DISTANCE=0 DELAY=40 DO $OUT[1]=TRUE
TRIGGER WHEN PATH=-30 DELAY=0 DO UP1(2) PRIO=-1
LIN POS1
TRIGGER WHEN DISTANCE=1 DELAY=-50 DO KLEBER=TRUE
TRIGGER WHEN PATH=180 DELAY=55 DO PULSE($OUT[4],TRUE,0.9)
TRIGGER WHEN PATH=0 DELAY=40 DO $OUT[6]=TRUE
LIN POS2 C_DIS
TRIGGER WHEN DISTANCE=0 DELAY=40 DO PULSE ($OUT[5],TRUE,1.4 )
TRIGGER WHEN PATH=-20 DELAY=-15 DO $OUT[8]
LIN POS3 C_DIS
TRIGGER WHEN DISTANCE=1 DELAY=-15 DO UP1 (7 ) PRIO= -1
LIN POS4
PTP HOME
END
DEF UP1 (NR :IN )

INT NR
IF $IN[1]==TRUE THEN
    $OUT[NR]=TRUE
ENDIF
END

```



**Abb. 45** Schaltpunkte und Bewegungsbahn zum vorangegangenen Beispiel



NOTIZEN:





## 11 Datenlisten

### 11.1 Lokale Datenlisten

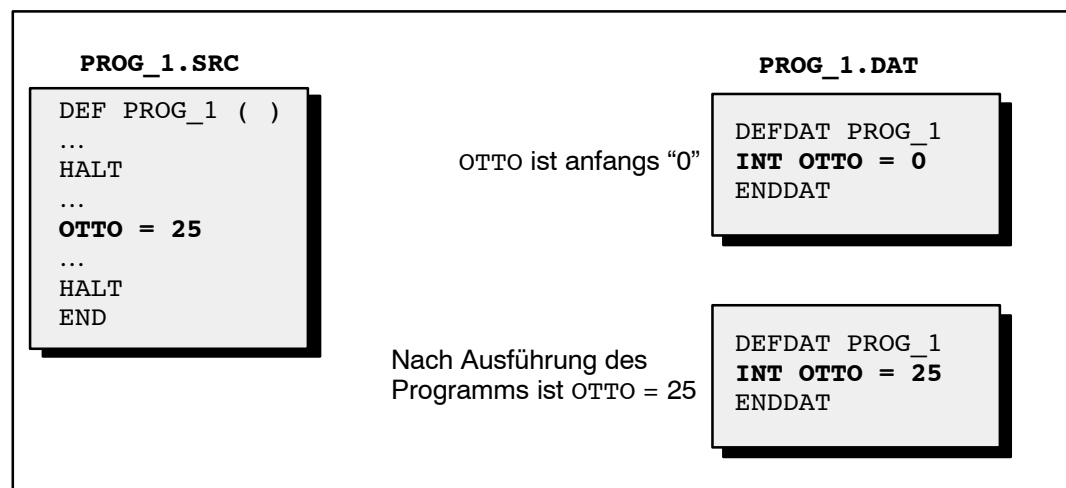
Datenlisten dienen dazu, programmspezifische oder auch übergeordnete Vereinbarungen bereitzustellen. Hierzu gehören auch die Punktinformationen, z.B. Koordinaten:

- Zu jeder SRC-Datei kann eine Datenliste erstellt werden. Diese trägt den selben Namen wie die SRC-Datei und endet mit der Erweiterung ".DAT".
- Die Datenliste ist lokal, obwohl sie eine eigene Datei ist.
- In einer Datenliste dürfen **nur** Deklarationen und Initialisierungen stehen.
- Es kann in einer Zeile deklariert und initialisiert werden.
- Es werden keine Systemvariablen akzeptiert.

DEFDAT

Die Vereinbarung von Datenlisten erfolgt analog zu Vereinbarung von SRC-Dateien: Die Deklaration wird mit dem Schlüsselwort **DEFDAT** und dem Programmnamen eingeleitet und mit dem Schlüsselwort **ENDDAT** abgeschlossen.

Die Initialisierung von Variablen erfolgt durch eine Wertzuweisung an die betreffende Variable direkt in der Deklarationszeile.



**Abb. 46** Initialisierung und Wertzuweisung an in Datenlisten deklarierte Variablen

Durch Deklaration und Initialisierung in der Datenliste entfällt diese im Hauptprogramm. Wird der Variablen **OTTO** im Hauptprogramm ein neuer Wert zugewiesen, so wird dieser auch in der Datenliste eingetragen und bleibt permanent gespeichert (s. Abb. 46).

Nach Steuerung AUS/EIN wird also mit dem "neuen" Wert gearbeitet. Dies ist unerlässlich für Online-Korrektur oder andere Programmkorrekturen.

Soll ein Hauptprogramm immer mit dem selben Wert starten, so muß man die entsprechende Variable im Hauptprogramm mit dem gewünschten Wert vorbesetzen.

In Datenlisten dürfen folgende Vereinbarungen stehen:

- Externvereinbarungen für Unterprogramme und Funktionen, die in der SRC-Datei benutzt werden.
- Importvereinbarungen für importierte Variablen.
- Deklarationen und Initialisierungen von Variablen, die in der SRC-Datei benutzt werden.
- Vereinbarungen von Signal- und Kanalnamen, die in der SRC-Datei benutzt werden.
- Deklarationen von Daten- und Aufzählungstypen (Struc, Enum), die in der Datenliste oder in der SRC-Datei benutzt werden.

## 11.2 Globale Datenlisten

In einer Datenliste definierte Variablen können einem “fremden” Hauptprogramm zugänglich gemacht werden.

**PUBLIC** Dazu muß die Datenliste mit dem optionalen Schlüsselwort **PUBLIC** in der Kopfzeile als “öffentlich zugänglich” definiert werden. Jetzt gibt es zwei Möglichkeiten die Variable zu Deklarieren:

**IMPORT**

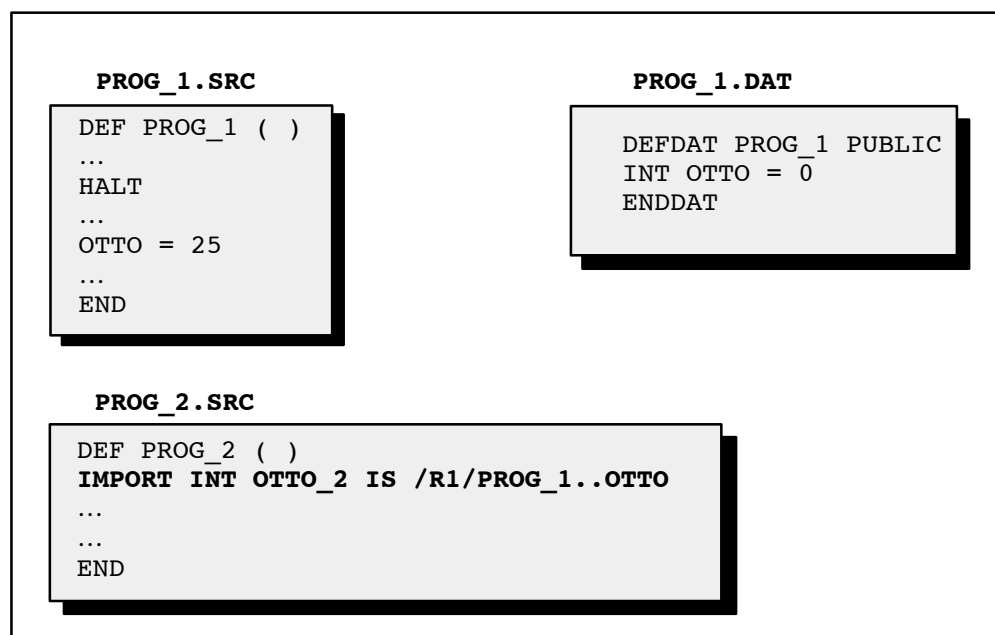
- Eine Variable wird z.B. mit **INT OTTO = 0** in der Datenliste definiert, und in dem fremden Hauptprogramm muß diese Variable mit dem Befehl **Import** importiert werden, damit sie zugänglich ist.

Einer importierten Variablen kann im Hauptprogramm ein anderer Name gegeben werden als in der Datenliste, aus der sie importiert wurde.

Wollen Sie also in einem Programm **PROG\_2 ( )** die Variable **OTTO** aus obiger Datenliste **PROG\_1** verwenden, so programmieren Sie neben dem Schlüsselwort **PUBLIC** in der Datenliste, folgende Importvereinbarung im Programm **PROG\_2 ( )**:

```
IMPORT INT OTTO_2 IS /R1/PROG_1..OTTO
```

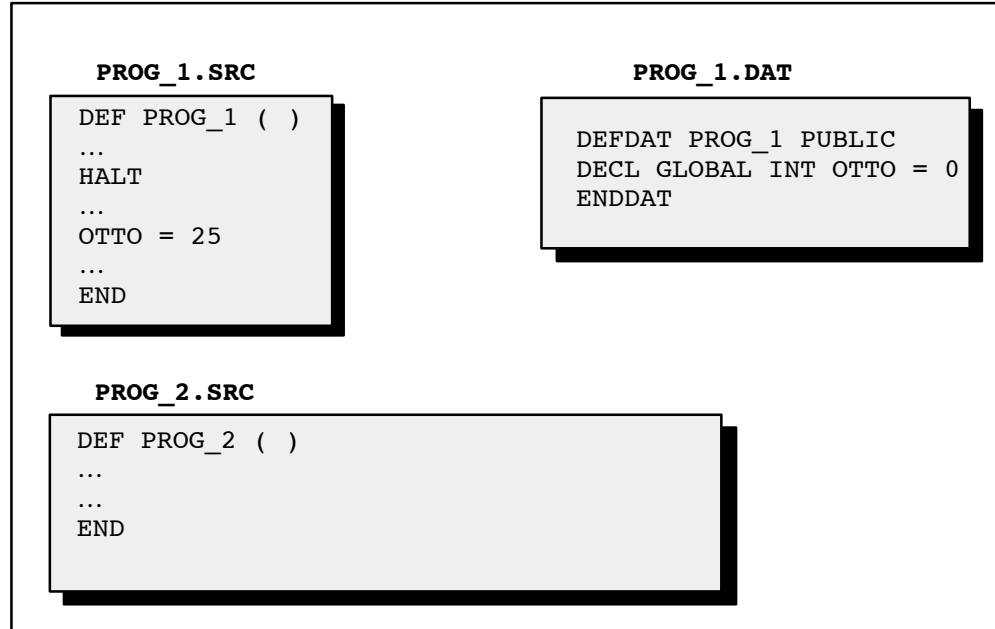
Die Variable **OTTO** aus der Datenliste **PROG\_1.DAT** im Verzeichnis **/R1** ist nun unter dem Namen **OTTO\_2** auch im Programm **PROG\_2 ( )** bekannt (s. Abb. 47).



**Abb. 47** Importieren von Variablen aus “fremden” Datenlisten mit **Import**

- Global
- Die Variable wird als "Globale Variable" deklariert z.B. `DECL GLOBAL INT OTTO = 0` und ist jedem fremden Hauptprogramm ohne Importbefehl zugänglich.

Wenn eine globale Variable deklariert wurde ist es nicht möglich in einem fremden Hauptprogramm den Namen der Variable zu ändern.

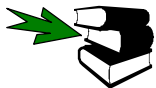


**Abb. 48** Importieren von Variablen aus "fremden" Datenlisten ohne `Import`



Die Deklaration einer globalen Variable ist nur in Datenlisten zulässig, wird sie in SRC- oder SUB-Files verwendet, erfolgt eine Fehlermeldung.

In der vordefinierten, globalen Systemdatenliste `$CONFIG.DAT` können Variablen, Strukturen, Kanäle und Signale definiert werden, die längere Zeit gültig sind und für viele Programme von übergeordneter Bedeutung sind. Die Variablen in der `$CONFIG.DAT` müssen nicht mit `IMPORT` vereinbart werden, da sie allen Anwenderprogrammen automatisch bekannt sind.



Weitere Informationen zur `$CONFIG.DAT` finden Sie im Kapitel **[Variablen und Vereinbarungen]**, Abschnitt **[Systemvariablen und Systemdateien]**.



---

## 12 Externer Editor

Dieses Zusatzprogramm erweitert die Software des Roboters um Funktionen, die auf der Bedienoberfläche nicht verfügbar sind.

### **Programm kopieren und unter anderem Namen speichern**

Diese Funktion wird z.B. zur Werkzeugvermessung nach einer Kollision benötigt.

### **Programm bereinigen**

Nicht referenzierte Bahnpunkte und Bewegungsparameter werden gelöscht.

### **Setzen und Verschieben von Endschaltern**

#### **Blockmanipulationen**

- Block markieren und kopieren, löschen oder ausschneiden.
- Bahn des Roboters im markierten Bereich umkehren, d.h. der vormals zuerst programmierte Punkt des markierten Bahnabschnitts wird zuletzt angefahren – der zuletzt programmierte Punkt jedoch zuerst.
- Bahn des Roboters im markierten Bereich in der X–Z–Ebene des Welt–Koordinatensystems spiegeln.
- Bewegungsparameter ( Geschwindigkeit, Beschleunigung etc. ) im markierten Bereich ändern.
- Verschieben aller Punkte innerhalb des markierten Bahnabschnittes im Werkstück–(BASE)–, Werkzeug–(TOOL)– oder Welt–(WORLD)–Koordinatensystem. Die Verschiebung bzw. Verdrehung kann durch die manuelle Eingabe eines Offsets im jeweiligen Koordinatensystem oder durch Teachen von Referenzpunkten erfolgen.
- Achsspezifisches Verschieben aller Punkte im markierten Bereich.

#### **Bahnpunkte anpassen**

- an ein anderes Werkzeug–, bzw.
- an ein anderes Werkstück–Koordinatensystem

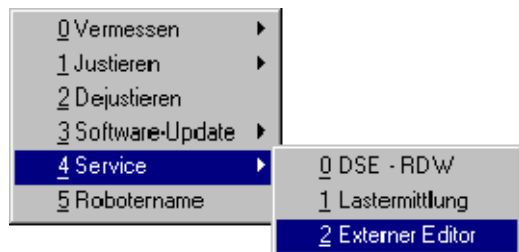
#### **Bahnpunkte anpassen**

- während ein Programm auf der Steuerung läuft, können Punktkoordinaten im Tool–, Base– und World–Koordinatensystem verschoben werden.

## 12.1 Starten des externen Editors

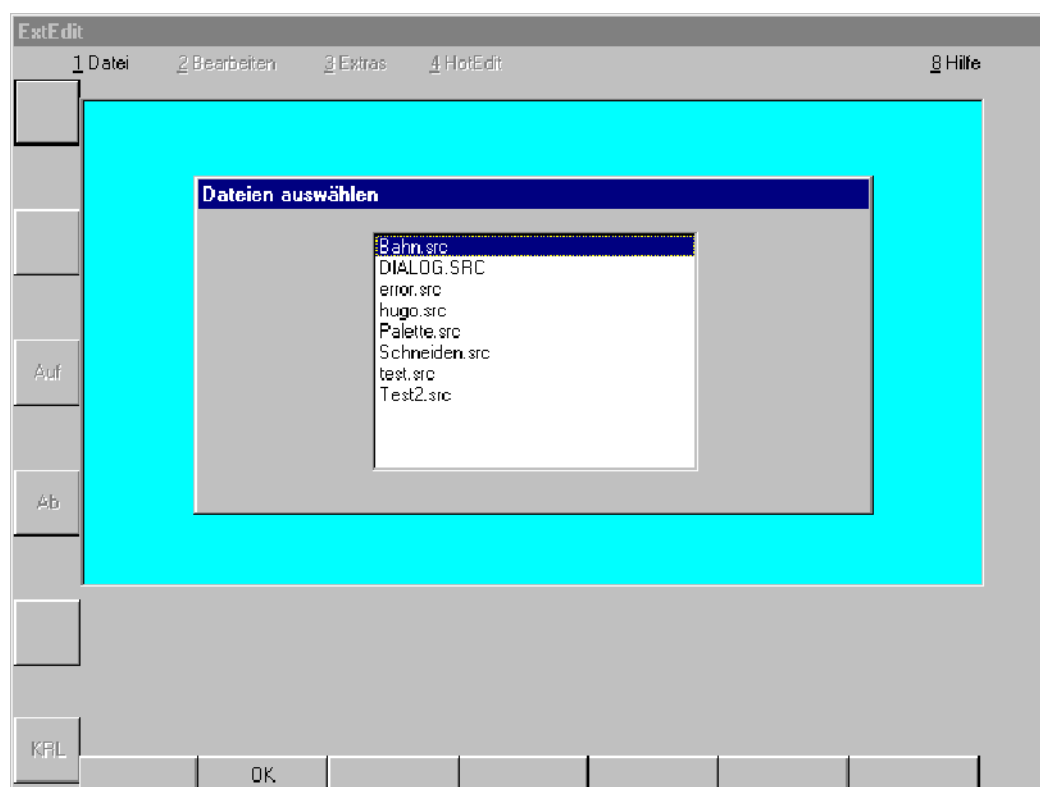
**Inbetriebn.**

Rufen Sie über das Menü "Inbetriebnahme" und die dort angebotene Option "Service" den externen Editor auf.



Ist der Menüpunkt "Inbetriebnahme" nicht verfügbar, so müssen Sie das angewählte Programm abwählen. Zur Erinnerung: Wählen Sie dazu aus dem Menü "Bearbeiten" die Option "Programm abwählen" aus.

Haben Sie den externen Editor auf diese Weise gestartet, so erscheint sein Programmfenster über der Bedienoberfläche der Robotersoftware. Diese tritt wieder in den Vordergrund, sobald eine Meldung über das Meldungsfenster ausgegeben wird.



Beim Start des Editors erscheint der Dialog "Dateien auswählen". Wählen Sie hier mit den Cursortasten "↓" und "↑" die zu editierende Datei aus. Es stehen alle SRC-Files aus dem R1-Verzeichnis zur Verfügung, bis auf jene Standarddateien die in der HotEdit.ini (C:\Krc\Util\ ) definiert sind.

Haben Sie die gewünschte Datei auf diese Weise markiert, so wird sie nach Druck auf den Softkey "OK" oder "Enter" in den Editor geladen. Die Dauer dieses Ladevorgangs hängt da-

bei von der Größe der Datei ab.



Erscheint die Meldung “\*.DAT-Datei nicht gefunden”, so haben Sie versucht, ein Programm in den Editor zu laden, zu dem keine Datenliste existiert. Diese Funktion ist in der vorliegenden Version des Editors nicht verfügbar. Quittieren Sie diese Meldung mit Druck auf die Eingabetaste. Öffnen Sie das Menü “Datei” und wählen die Option “Datei öffnen” aus, um an das Fenster “Dateien auswählen” zu gelangen.

Wurde die Datei erfolgreich in den Editor geladen, wird Ihnen das Programm ähnlich wie im Editor der Bedienoberfläche angezeigt.

```
ExtEdit [ PALETTE1 ]
1 Datei 2 Bearbeiten 3 Extras 4 HotEdit 8 Hilfe

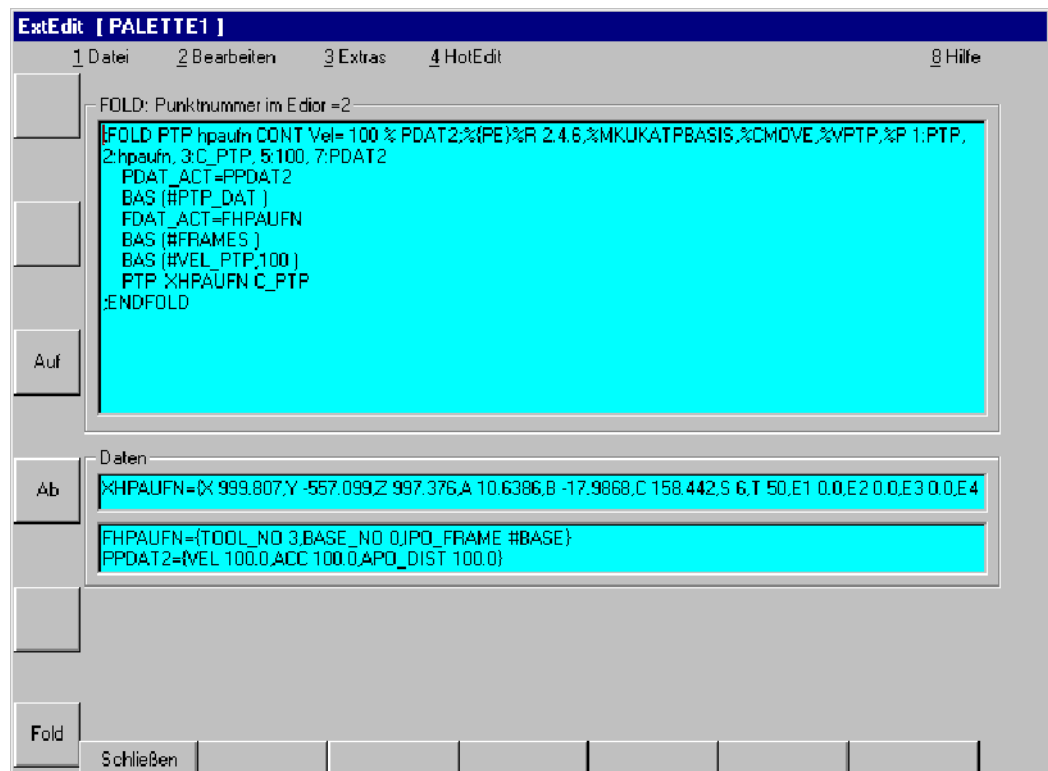
POS ABSETZPOS[4.4]
POS UEBER_ABS[4.4]
INT ZEILE,SPALTE
REAL DX,DY,DZ ,werkstueckdaten
INT X

INI
DX=80
DY=80
DZ=80

1: PTP HOME Vel= 100 % DEFAULT
SET GRP 1 State= OPN GDAT1
FOR ZEILE=1 TO 4
FOR SPALTE=1 TO 4
ABSETZPOS[ZEILE,SPALTE]~XGELERNT
ENDFOR
ENDFOR

KRL
Schließen
```

Mit den Statustasten “Auf”, bzw. “Ab” können Sie diese Markierung zeilenweise in Richtung Listenanfang, bzw. –ende bewegen. Betätigen Sie die Statustaste “KRL”, so wird Ihnen das Programm wie im Expertenmodus mit den Einstellungen “Alle Fold öffnen” und “Limited Visibility off” angezeigt.



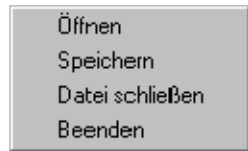
Im Fenster "Daten" werden Ihnen die zum programmierten Punkt gehörenden Daten angezeigt. Durch Betätigung der Statustaste, nun mit "Fold" beschriftet, kehren Sie zur Darstellung des gesamten Programmes zurück.



---

## 12.2 Menü “Datei”

Im diesem Menü stehen Ihnen Funktionen zum Umgang mit Dateien zur Verfügung.

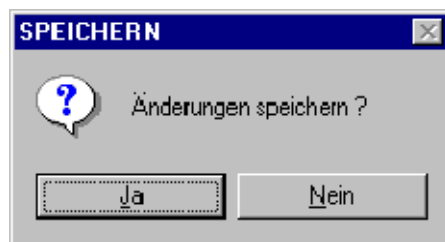


### 12.2.1 Öffnen

Bei Betätigung dieser Option erscheint das Fenster “Dateien auswählen” wie bereits im vorhergehenden Abschnitt 12.1 beschrieben.

### 12.2.2 Speichern

Wurde der Inhalt der im Editor geladenen Datei mit den angebotenen Funktionen verändert, so erfolgt vor dem Speichern der neuen Version dieser Datei, bzw. vor dem Beenden der Editorsitzung eine Rückfrage, ob die Änderungen übernommen werden sollen.



Möchten Sie die ausgeführten Änderungen übernehmen, so genügt ein Druck auf die Eingabetaste, um die Vorauswahl zu bestätigen.



Die Inhalte der bereits bestehenden Version der editierten Datei gehen dabei, zumindest teilweise, unwiderruflich verloren.

Um die Aktion abzubrechen, muß zuerst durch Druck auf eine der Cursortasten der sog. 'Eingabefokus' auf die Schaltfläche mit der Beschriftung “Nein” gelegt werden. Danach kann diese Auswahl durch Druck auf die Eingabetaste übernommen werden. Die bereits bestehende Version der editierten Datei bleibt weiterhin unverändert bestehen.

### 12.2.3 Datei schließen

Hier ist die gleiche Bedienprozedur erforderlich, wie schon im vorhergehenden Abschnitt 12.2.2 beschrieben wurde.

Nachdem die im Editor befindliche Datei geschlossen wurde, kann über den Menüpunkt “Datei” und die darin angebotene Option “Öffnen” ( siehe auch Abschnitt 12.2.1 ) eine andere Datei in den Editor geladen werden.

### 12.2.4 Beenden

Hier ist die gleiche Bedienprozedur erforderlich, wie schon im vorhergehenden Abschnitt 12.2.2 beschrieben wurde.

Das Fenster des Editors wird geschlossen und die Bedienoberfläche der Robotersoftware erscheint wieder.



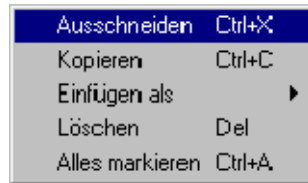
---

### NOTIZEN:

---

## 12.3 Menü “Bearbeiten”

Dieses Menü beinhaltet eine Reihe von Blockfunktionen.



Das Markieren von Blöcken erfolgt durch Betätigung einer der beiden Statustasten “Auf”, bzw. “Ab” bei gedrückt gehaltener “SHIFT”-Taste.

### 12.3.1 Ausschneiden

Löscht den markierten Bahnabschnitt aus dem Programm und hält ihn in der Zwischenablage zum Einfügen bereit.

### 12.3.2 Kopieren

Kopiert den markierten Bahnabschnitt in die Zwischenablage und hält ihn dort zum Einfügen bereit.

### 12.3.3 Einfügen als ...

#### 12.3.3.1 ursprüngliche Bahn

Fügt den Inhalt der Zwischenablage in gleicher Reihenfolge wie ausgeschnitten, bzw. kopiert wieder ein.

#### 12.3.3.2 Rückwärtsbahn

Fügt den Inhalt der Zwischenablage in umgekehrter Reihenfolge wieder ein.

Der letzte Bahnpunkt des markierten und ausgeschnittenen, bzw. kopierten Bahnabschnitts wird an den Anfang des eingefügten Blockes gesetzt; der erste Bahnpunkt an das Ende.

#### 12.3.4 Löschen

Der markierte Bahnabschnitt wird ohne Speicherung in der Zwischenablage gelöscht.



Das Löschen eines markierten Bahnabschnittes kann nicht rückgängig gemacht werden. Haben Sie versehentlich Teile des Programms gelöscht, so schließen Sie bitte die Datei, ohne die Änderungen zu speichern und öffnen sie anschließend erneut.

#### 12.3.5 Alles markieren

Das gesamte, in den Editor geladene Programm wird markiert.



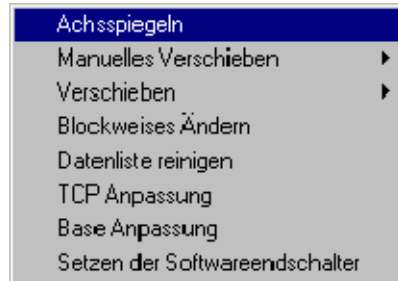
---

NOTIZEN:

---

## 12.4 Menü “Extras”

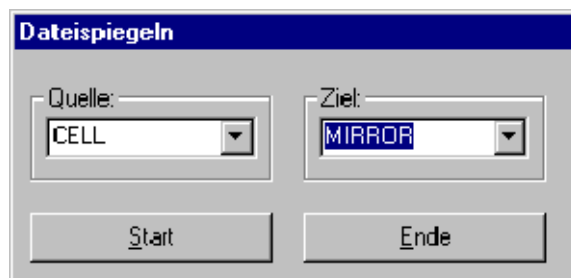
In diesem Menü sind Optionen zur geometrischen Manipulation des Programmes untergebracht.



### 12.4.1 Achsspiegeln

Mit Hilfe dieser Funktion können Sie die Lage der programmierten Bahnpunkte in der X-Z-Ebene des \$ROBROOT-Koordinatensystems spiegeln. Voraussetzung ist, daß die Bewegungssätze markiert sind.

Nach Auswahl der Option wird ein Dialogfenster eingeblendet, in dem Sie den Namen der Datei angeben müssen, in der das geladene Programm mit gespiegelten Bahnpunkten gespeichert wird.

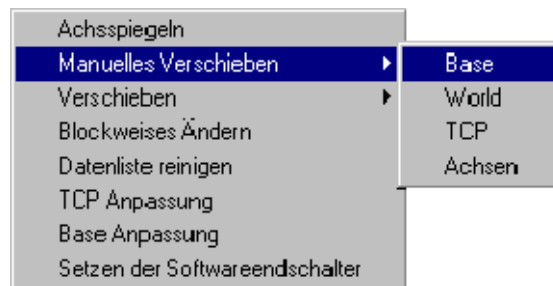


Durch Druck auf die Eingabetaste wird der Vorgang gestartet. Wollen Sie die Aktion jedoch abbrechen, so drücken Sie bitte die Taste “ESC”.

Die erfolgreiche Konvertierung der Datei wird Ihnen mit folgender Meldung angezeigt. Bestätigen Sie diese Meldung durch Druck auf die Eingabetaste.



## 12.4.2 Manuelles Verschieben



Mit Hilfe dieser Funktion können Sie die Lage der markierten Bahnpunkte im:

- Werkstück– ( BASE ) –Koordinatensystem
- Welt– ( WORLD ) –Koordinatensystem
- Werkzeug– ( TOOL ) –Koordinatensystem oder
- achsspezifisch

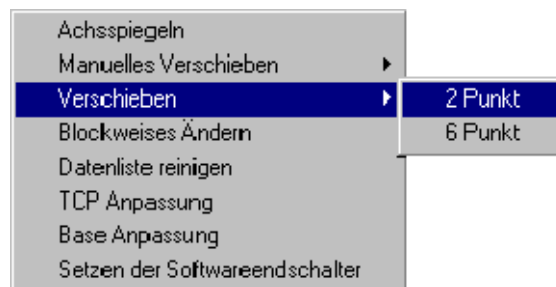
verschieben.

Nach Auswahl einer der Optionen erscheint am unteren Rand des Displays ein Dialogfenster, in dem Dialogfenster kann die Verschiebung (X, Y, Z) und Verdrehung (A, B, C) bzgl. des jeweiligen Koordinatensystems eingegeben werden. Das hier abgebildete Dialogfenster steht beispielhaft für die anderen Dialogfenster.



Mit den Cursortasten “↑” oder “↓” bewegen Sie die Einfügemarke von Eingabefeld zu Eingabefeld. Haben Sie alle für die Verschiebung erforderlichen Angaben gemacht, so drücken Sie bitte den Softkey “Ok”. Sie können die Funktion aber auch jederzeit durch Betätigen des Softkey “Abbrechen” beenden.

### 12.4.3 Verschieben



Mit dieser Funktion können Sie die Lage der markierten Bahnpunkte unter Zuhilfenahme eines Vektors verschieben ( 2 Punkt ), bzw. verschieben und gleichzeitig das zu Grunde liegende Koordinatensystem verdrehen ( 6 Punkt ).

Für die Funktionalität 2-Punkt-Verschieben muß ein Referenzfile (beliebiger Name; .src, .dat ) geteacht werden. In diesem Referenzfile werden zwei Punkte abgelegt. Diese Punkte definieren den Verschiebevektor, um den die markierten Punkte verschoben werden. Die Funktionalität 6-Punkt-Verschieben arbeitet nach dem gleichen Prinzip. In diesem Referenzfile müssen sechs Punkte geteacht werden. Die ersten drei definieren ein Quell-Base, die zweiten drei ein Ziel-Base. Die Verschiebung und Verdrehung zwischen diesen zwei Base bestimmt den Wert, um den die markierten Punkte geshiftet werden.



Der Vektor muß bereits vor dem Aufruf der Funktion in einer Datei gespeichert worden sein.

Weiterhin sind die zu verschiebende Bewegungssätze markiert werden.

Nach Anwahl einer der angebotenen Optionen öffnet sich ein Dialogfenster zur Auswahl der Referenzdatei. Wählen Sie hier mit den Cursortasten "↓" und "↑" die Datei aus, in welcher der Vektor gespeichert wurde.

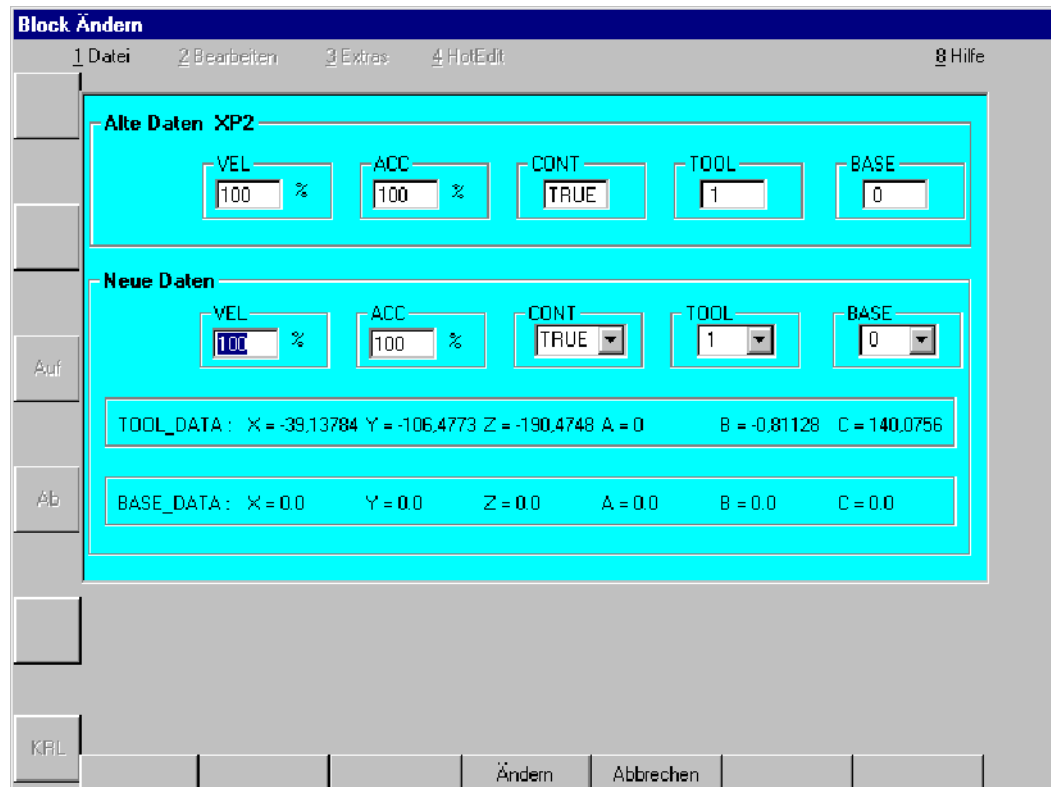


Haben Sie die gewünschte Datei markiert, so wird die Verschiebung durch Druck auf die Eingabetaste gestartet. Während dieser Prozeß läuft, wird im unteren Teil des Displays eine Fortschrittsanzeige eingeblendet.

#### 12.4.4 Blockweises Ändern

Mit Hilfe dieser Funktion können in markierten Programmteilen Bewegungsdaten geändert werden. Zur Erinnerung: Das Markieren von Blöcken erfolgt durch Betätigung einer der beiden Statustasten "Auf", bzw. "Ab" bei gedrückt gehaltener "SHIFT"-Taste.

Nach dem Aufruf der Funktion erscheint das unten abgebildete Fenster:



Mit den Cursortasten "→" oder "←" bewegen Sie die Einfügemarke von Eingabefeld zu Eingabefeld.



Listenfelder werden mit der Tastenkombination "ALT"+"↓", bzw. "ALT"+"↑" geöffnet.

Haben Sie die gewünschten Änderungen eingegeben, so drücken Sie bitte den Softkey "Ändern". Sie können die Funktion jederzeit durch Betätigen des Softkey "Abbrechen" verlassen.

#### 12.4.5 Datenliste reinigen

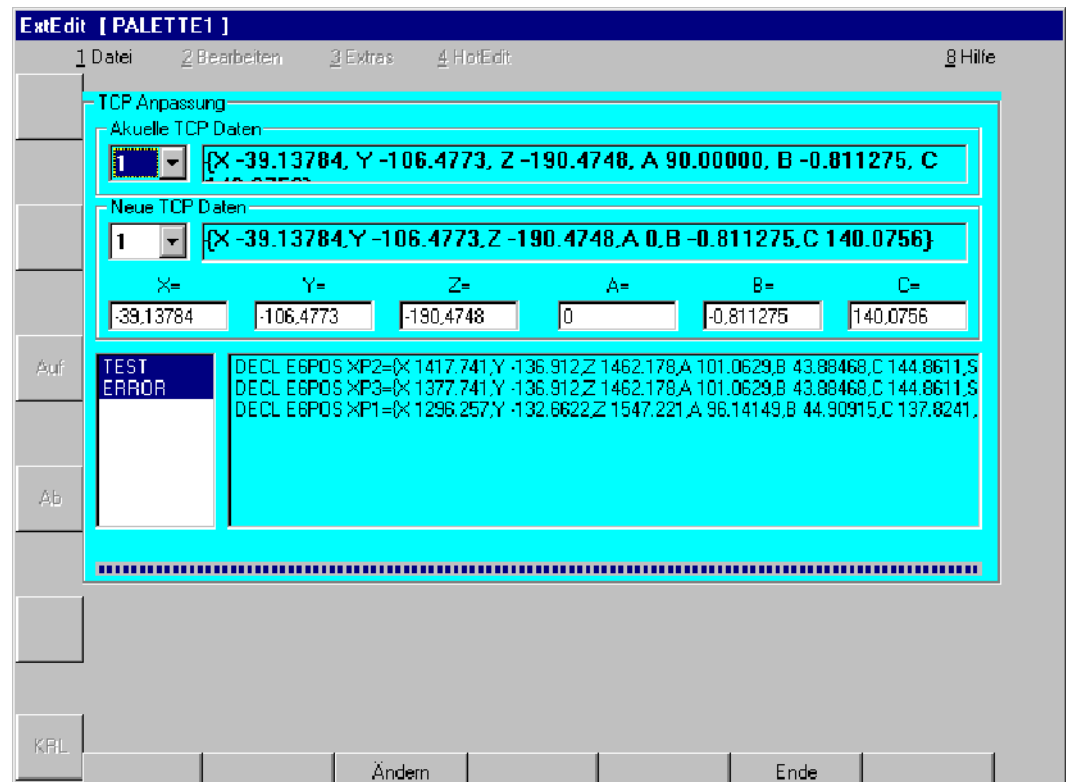
Nach dem Aufruf dieser Funktion werden nicht referenzierte Bahnpunkte und Bewegungsparameter aus der zum Programm gehörenden Datenliste ( \*.DAT ) gelöscht.



### 12.4.6 TCP-, bzw. BASE-Anpassung

Mit Hilfe dieser Funktion können Sie das im Editor geladene Programm an ein anderes Werkzeug-( TOOL )-, bzw. Werkstück-( BASE )-Koordinatensystem anpassen.

Nach dem Aufruf der Funktion erscheint das unten abgebildete Fenster. Als Beispiel wurde hier die TCP-Anpassung gewählt.



Es werden alle Programme mit zugehörigen Punkten angezeigt, die von der Tool- bzw. Baseanpassung betroffen sind.

Um die Einfügemarke von Feld zu Feld bewegen zu können, müssen die Cursor-Steuernfunktionen des Nummernfeldes aktiviert sein. Betätigen Sie dazu kurz die "NUM"-Taste. Anschließend können Sie mit der Tastenkombination "SHIFT"+"TAB" die Einfügemarke von Feld zu Feld springen lassen.



Listenfelder werden mit der Tastenkombination "ALT"+"↓", bzw. "ALT"+"↑" geöffnet.

Wählen Sie aus dem Listenfeld im Bereich "Neue TCP Daten" das Werkzeug aus, an welches das geladene Programm angepaßt werden soll. In den Eingabefeldern "X=", "Y=", "Z=", "A=", "B=" und "C=" können die neuen Werkzeugdaten auch manuell eingegeben werden.

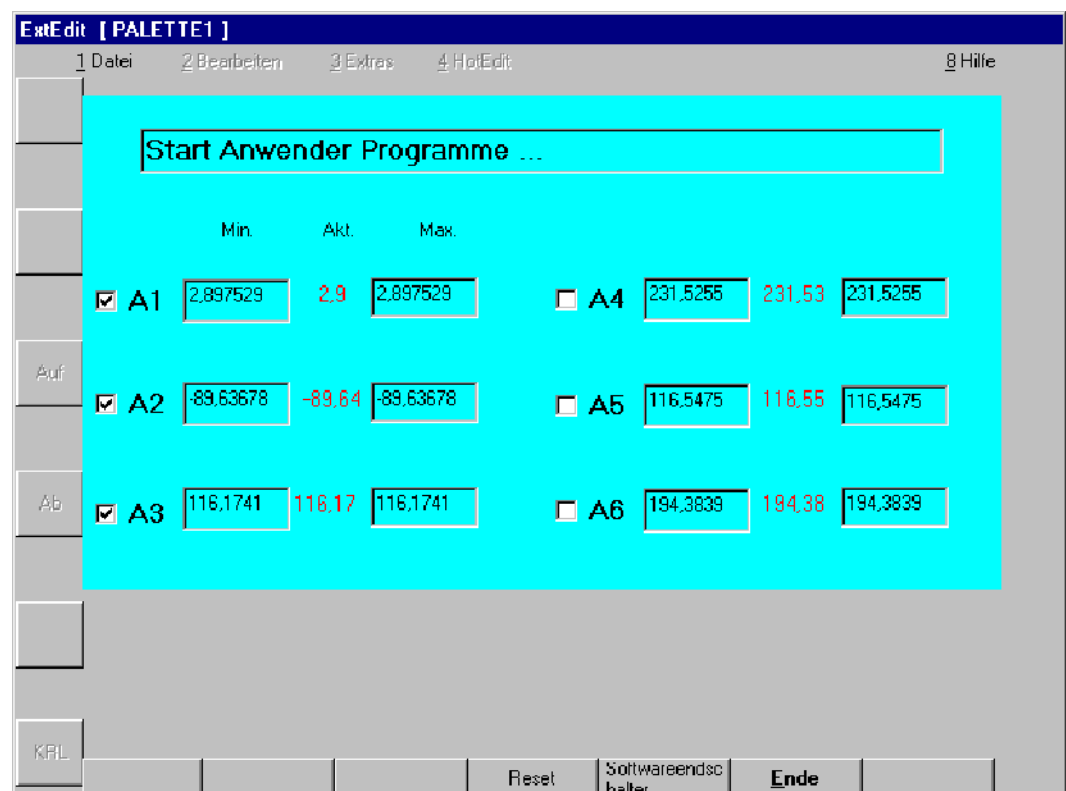
Haben Sie die gewünschten Änderungen eingegeben, so drücken Sie bitte den Softkey "Ändern". Sie können die Funktion jederzeit durch Betätigen des Softkey "Ende" verlassen.

## 12.4.7 Setzen der Softwareendschalter

Mit Hilfe dieser Funktion können die Software-Endschalter der einzelnen Achsen an die Bewegungsprogramme angepaßt werden.

Zu diesem Zweck starten Sie die Funktion und wechseln dann mit der Fenstertaste zurück zur Bedienoberfläche der Robotersoftware. Dort starten Sie die Bewegungsprogramme. Sind alle für die Steuerung relevanten Programme durchlaufen, rufen Sie die Funktion erneut auf.

Mit dieser Funktion werden die maximalen Achspositionen der Programme bestimmt, die laufen, solange die Funktion gestartet ist. Die ermittelten Werte können als Softwareendschalter gesetzt werden. Mit einer Checkbox können die Achsen, für die dies erfolgen soll, gekennzeichnet werden. Mit *Reset* können die Softwareendschalter auf ihre ursprünglichen Werte zurückgesetzt werden.



	Min.	Akt.	Max.
<input checked="" type="checkbox"/> A1	2,897529	2,9	2,897529
<input checked="" type="checkbox"/> A2	-89,63678	-89,64	-89,63678
<input checked="" type="checkbox"/> A3	116,1741	116,17	116,1741
<input type="checkbox"/> A4	231,5255	231,53	231,5255
<input type="checkbox"/> A5	116,5475	116,55	116,5475
<input type="checkbox"/> A6	194,3839	194,38	194,3839

In den Anzeigefeldern "Min." und "Max." der Achsen A1 bis A6 wurden die Minimal-, bzw. Maximalwerte eingetragen, die während des Programmlaufes gemessen wurden.



Zu den gemessenen Achswerten wird ein Puffer von 5° addiert.

Um den sog. "Eingabefokus" von Kontrollkästchen zu Kontrollkästchen bewegen zu können, müssen die Cursor-Steuerfunktionen des Nummernfeldes aktiviert sein. Betätigen Sie dazu kurz die "NUM"-Taste. Anschließend können Sie mit der Tastenkombination "SHIFT" + "TAB" die Einfügemarke von Kästchen zu Kästchen springen lassen.

Sie können nun die jeweils angewählte Achse durch Betätigen der Leertaste an- oder abwählen.

Durch Druck auf den Softkey "Softwareendschalter" werden die Software-Endschalter der angewählten Achsen neu gesetzt. Diese Aktion kann jederzeit, auch zu einem späteren Zeitpunkt, durch Druck auf den Softkey "Reset" wieder rückgängig gemacht werden.

---

## 12.5 Menü “HotEdit”

Das Menü “HotEdit” ermöglicht es Punktkoordinaten im Tool-, Base-, und Worldkoordinatensystem während des Programmlaufs online zu verschieben. Der Menüpunkt “Limits” gestattet eine Eingrenzung der Verschiebung.



### 12.5.1 Limits

Bei Anwahl des Menüpunktes “Limits” erscheint das Fenster in dem die Verschiebetoleranzen eingetragen sind und verändert werden können.



Um eine Veränderung vorzunehmen, müssen Sie mit den Pfeiltasten “↑” oder “↓” den Cursor an die gewünschte Stelle bringen, und dort mit den Nummernblock die neue Toleranz eingeben. Betätigen Sie den Softkey “OK” um die Werte abzuspeichern und die Aktion zu beenden. Möchten Sie ohne abzuspeichern beenden, betätigen Sie “Cancel”.

### 12.5.2 Tool, Base und World

Gehen Sie mit dem Cursor auf den zu verschiebenden Bewegungssatz, bzw. markieren Sie mehrere Bewegungssätze.

Nach Auswahl einer der Optionen erscheint am unteren Rand des Displays ein Dialogfenster, in dem die Verschiebung (X, Y, Z) und Verdrehung (A, B, C) bzgl. des jeweiligen Koordinatensystems eingegeben werden kann. Das hier abgebildete Dialogfenster steht beispielhaft für die anderen Dialogfenster.



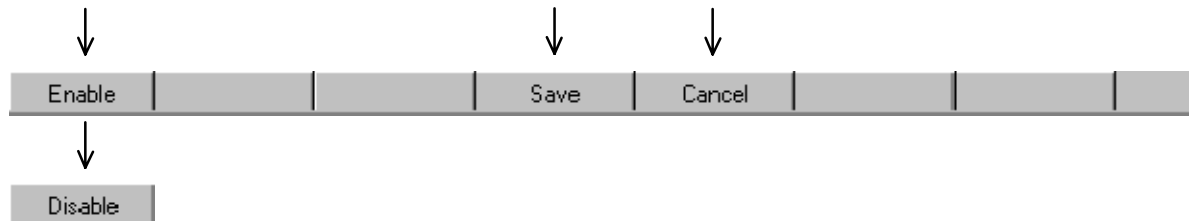
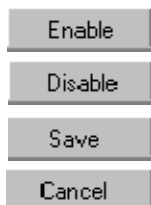
Mit den Cursortasten “↑” oder “↓” bewegen Sie die Einfügemarke von Eingabefeld zu Eingabefeld. Haben Sie alle für die Verschiebung erforderlichen Angaben (ganze Zahlen) gemacht, so drücken Sie bitte den Softkey “Ok”. Sie können die Funktion aber auch jederzeit durch Betätigen des Softkey “Abbrechen” beenden.

Haben Sie mit “Ok” bestätigt, erscheint die angestrebte Verschiebung unter dem externen Editorfenster:

Shifting: BASE {X 7, Y 2, Z 0, A 0, B 0, C 0}

Liegt einer der eingegebenen Werte außerhalb der Toleranzen, erscheint im Programmverschiebungsfenster ein Hinweis bzw. eine Fehlermeldung, daß der eingegebene Wert die Toleranzen überschreitet.

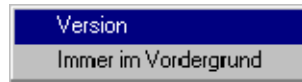
Bis zu diesem Zeitpunkt ist noch keine Verschiebung vorgenommen worden. Dies kann nun über die neue Softkeyleiste geschehen.

Durch Drücken des Softkeys “Enable” ist die Verschiebung aktiv, aber nicht gespeichert. Nun ist es möglich mit dem Softkey “Disable” die Verschiebung zurückzunehmen, also den Urzustand wieder herzustellen. Das ist gerade von Vorteil, wenn ein Programm am Laufen ist und die Wirkung der Verschiebung unmittelbar zu verfolgen ist. Verließ die Verschiebung erfolgreich, so können Sie durch Drücken des Softkeys “Save” die neuen Punktkoordinaten im DAT-File sichern. Sind die Daten gesichert, schließt diese Anwendung. War die Verschiebung nicht zufriedenstellend drücken Sie den Softkey “Cancel” um den Vorgang abbrechen und erneut mit “HotEdit” zu starten.

---

## 12.6 Menü “Hilfe”



### 12.6.1 Version

Nach Auswahl dieser Option wird Ihnen in der Mitte des Displays ein Meldungsfenster angezeigt, in dem die Versionsnummer des externen Editors angegeben wird.

Bestätigen Sie diese Meldung durch Druck auf die Eingabetaste.

### 12.6.2 Immer im Vordergrund

Nach Auswahl dieser Option tritt die Bedienoberfläche der Robotersoftware bis zum Beenden des Editors nicht mehr in den Vordergrund.

Dies betrifft auch das Verhalten bei der Ausgabe von Meldungen über das Meldungsfenster.



**Zeichen**

.ERR, 15  
!, 110  
!-Zeichen, 108  
?, 111  
:, 112  
# – Zeichen, 45  
#BASE, 69, 84  
#CONSTANT, 83  
#INITMOV, 83, 84  
#PATH, 84  
#TCP, 69  
#VAR, 83  
\$ – Zeichen, 59  
\$ACC.CP, 82  
\$ACC.ORI1, 82  
\$ACC.ORI2, 82  
\$ACC\_AXIS, 70  
\$ADVANCE, 91  
\$ALARM\_STOP, 59, 60  
\$APO.CDIS, 98, 101  
\$APO.CORI, 98, 101  
\$APO.CPTP, 95  
\$APO.CVEL, 98, 101  
\$APO\_DIS\_PTP, 95  
\$BASE, 67, 74  
\$CIRC\_TYPE, 84, 87  
\$CONFIG.DAT, 10, 61  
\$CUSTOM.DAT, 60  
\$CYCFLAG, 59  
\$FLAG, 58  
\$IBUS\_ON, 60  
\$IPO\_MODE, 69  
\$MACHINE.DAT, 60  
\$MASCHINE.DAT, 10  
\$NULLFRAME, 74  
\$NUM\_AX, 60  
\$ORI\_TYPE, 83, 87  
\$POS\_ACT, 68, 69  
\$PRO\_MODE, 27  
\$PSER, 60  
\$ROBCOR.DAT, 10, 61  
\$ROBROOT, 67, 74  
\$TIMER, 58  
\$TIMER\_FLAG, 58  
\$TIMER\_STOP, 58  
\$TOOL, 67, 74  
\$VEL.CP, 82

\$VEL.ORI1, 82  
\$VEL.ORI2, 82  
\$VEL\_AXIS, 70  
\$WORLD, 67, 74

**Zahlen**

2-dimens., 40  
3-dimens., 41

**A**

A10.DAT, 10  
A10.SRC, 10  
A10\_INI.DAT, 10  
A10\_INI.SRC, 10  
A20.DAT, 10  
A20.SRC, 10  
A50.DAT, 10  
A50.SRC, 10  
ABS(X), 56  
Abweisende Schleife, 124  
Achsbeschleunigung, 72  
Achsgeschwindigkeit, 72  
achsspezifisches Koordinatensystem, 63  
Achsspiegeln, 187  
ACOS(x), 56  
Aggregat, 42  
aktuelle FOLD öff/schl, 22  
alle FOLDS öffnen, 22  
alle FOLDS schließen, 22  
Analoge Ausgänge, 139  
Analoge Eingänge, 141  
Ändern von Programmen, 17  
ANIN, 141  
ANIN OFF, 141  
ANIN ON, 141  
ANIN/ANOUT, 139  
ANOUT OFF, 139  
ANOUT ON, 139  
Anweisung, 12  
Anweisungsteil, 12  
Anwender, 7  
ARCSPS.SUB, 10  
Arcuscosinus, 56  
Arcussinus, 56  
Arcustangens, 56  
Arithmetische Operatoren, 46  
ATAN2(Y,X), 56  
Aufbau und Struktur von Programmen, 9

Aufzählungstypen, 44  
Ausblenden von Bits, 54  
Ausschneiden, 18  
automatischer Vorlaufstop, 92  
AXIS, 44, 64, 111

**B**

B\_AND, 53, 55  
B\_EXOR, 53, 55  
B\_NOT, 53, 55  
B\_OR, 53, 55  
Bahnbewegungen, 82  
BAS.SRC, 10, 76  
BASE, 69  
BASE-Anpassung, 191  
basisbezogene Interpolation, 68  
Basiskoordinatensystem, 68  
Bedingte Verzweigung, 120  
Benutzergruppe Experte, 7  
Beschleunigung, 82  
Beschränkung der Informationsmenge, 24  
Betrag, 56  
Bewegungsanweisungen, 63  
Bewegungsprogrammierung, 63  
Bin Dez, 38  
Binäre Ein-/Ausgänge, 132  
Binärsystem, 37  
Binden, 15  
Binder, 15  
Bit-Operatoren, 53  
Blockfunktionen, 17  
Blockkennung, 121  
Blockweises Ändern, 190  
BOOL, 37, 38  
BOSCH.SRC, 10  
BRAKE, 160

**C**

C\_DIS, 98, 101, 113  
C\_ORI, 98, 101, 113  
C\_PTP, 95, 113  
C\_VEL, 98, 101, 113  
CA, 89  
CELL.DAT, 10  
CELL.SRC, 10, 11  
CHAR, 37, 39  
CIRC, 89, 117

CIRC !, 108  
CIRC REL, 117  
CIRC-CIRC-Überschleifen, 101  
CIRC-LIN-Überschleifen, 101  
CIRC\_REL, 89  
Compiler, 15  
CONFIRM, 130  
Continuous Path, 82  
CONTINUE, 93  
COS(X), 56  
CP-Bewegungen, 82  
CSTEP, 27  
CTRL-C, 17  
CTRL-V, 18  
CTRL-X, 18

**D**

Dateikonzept, 12  
Dateiliste, 12  
Dateistruktur, 12  
Datenliste, 12  
Datenliste reinigen, 190  
Datenlisten, 175  
Datenmanipulation, 46  
Datenobjekte, 35  
Datentyp, 35  
DECL, 35  
DEF, 12, 145  
DEF-Zeile, 26  
DEFDAT, 175  
DEFFCT, 146  
Detailansicht, 24  
Dezimalsystem, 37  
DIGIN, 143  
DIGIN OFF, 143  
DIGIN ON, 143  
Digitale Ein-/Ausgänge, 135  
Digitaleingänge, 143  
DISTANCE, 166  
Distanzkriterium, 98  
Dreidimensionales Feld, 41

**E**

E6AXIS, 44, 111  
E6POS, 111  
Editier-Cursor, 109  
Editieren, 15  
Editor, 17



Ein-/Ausgabeeweisungen, 131  
Einblenden von Bits, 54  
Eindimensionales Feld, 39  
Einfache Datentypen, 37  
Einfügen, 18  
ELSE, 120  
END, 12  
ENDFOLD, 22  
ENDFOR, 122  
ENDLOOP, 127  
Endlosschleife, 127  
ENDWHILE, 124  
ENUM, 44  
Ersetzen, 19  
Erstellen eines neuen Programms, 14  
EXIT, 127  
Exklusive ODER-Verknüpfung, 53  
Experte, 7  
EXT, 76  
Externer Editor, 179

**F**

Fehlerbehandlung, 29  
Felder, 39  
Feldindex, 39  
Flags, 59, 164  
flankengetriggert, 156  
Fliegendes Messen, 163  
FLT\_SERV.DAT, 10  
FLT\_SERV.SRC, 10  
FOLD, 22  
FOR, 122  
FRAME, 44, 111  
Frameverknüpfung, 47  
Funktionen, 12, 145

**G**

Geometrische Datentypen, 44  
geometrische Datentypen, 44  
Geometrischer Operator, 47  
Geometrischer Operator ":", 112  
Geschwindigkeit, 82  
Geschwindigkeitskriterium, 98  
GLOBAL, 177  
global, 146  
Globale Datenlisten, 176  
Globale Variable, 177

GO, 27  
GOTO, 119  
Greiferbezogene Interpolation, 69  
Grundbereich, 78

**H**

H50.SRC, 10  
H70.SRC, 10  
HALT, 129  
Handwurzelpunkt, 78  
Hauptlauf, 91  
Herausfiltern von Bits, 54  
Hex Dez, 38  
Hexadezimalsystem, 37  
Höheres Fahrprofil, 71  
Home-Fahrt, 77

**I**

IF, 120  
Impulsausgänge, 137  
Index, 39  
INI, 76  
INT, 37  
INTERRUPT, 154  
Interrupt ausschalten, 156  
Interrupt einschalten, 156  
Interrupt-Behandlung, 153  
Invertierung, 53  
IR\_STOPM.SRC, 10  
ISTEP, 27

**K**

kartesische Koordinatentransformation, 64  
Kinematik-Singularität, 77  
Kinematische Kette, 68, 69  
Kommentare, 32  
Kompilieren, 15  
konstant + bahnbezogen, 84  
konstant + raumbezogen, 86  
Koordinatensysteme, 63, 74  
Koordinatentransformation, 64  
Kopieren, 17  
Kreisbewegungen, 89  
Kreiswinkel, 89  
KRL, 7  
KRL-Assistent, 109  
KUKA-Robot-Language, 109

**L**

Lebensdauer, 34  
LIN, 88, 115  
LIN !, 108  
LIN REL, 115  
LIN-CIRC Überschleif, 103  
LIN-LIN-Überschleifen, 98  
LIN\_REL, 88  
Linearbewegungen, 88  
Logische Operatoren, 52  
Logische Verknüpfung, 52  
lokal, 146  
Lokale Datenlisten, 175  
LOOP, 127  
Löschen, 18

**M**

Manuelles Verschieben, 188  
Maschinensprache, 15  
Mechanische Nullstellung, 72  
Mehrdeutige Roboterkinematiken, 78  
Mehrdeutigkeit, 78  
MERKER, 119  
MSG\_DEMO.SRC, 10  
MSTEP, 27

**N**

Namen, 33  
NEW\_SERV.SRC, 10  
Nicht abweisende Schleife, 125

**O**

ODER-Verknüpfung, 53  
Operand, 46  
Operator, 46  
Orientierungsführung, 83  
Orientierungskriterium, 98

**P**

P00.DAT, 10  
P00.SRC, 10  
Parameterliste, 148  
Parameterübergabe, 148  
Paßwort, 7  
PERCEPT.SRC, 11  
Platzhalter, 110

POS, 44, 111  
Positionsangabe, 111  
Priorität, 55, 156, 165, 169  
Prioritäten von Operatoren, 55  
Programm-Korrektur, 17  
Programmablaufarten, 27  
Programmablaufkontrolle, 119  
Programme erstellen und editieren, 14  
Programmverzweigungen, 119  
PROKOR, 17  
PSTEP, 27  
PTP, 70, 73, 113  
PTP !, 108  
PTP REL, 113  
PTP-Bahnüberschleifen, 104  
PTP-PTP-Überschleifen, 95  
PTP\_REL, 73  
PUBLIC, 176  
PULSE, 137  
Punkt-Separator, 42  
Punkt-zu-Punkt Bewegungen, 70

**R**

REAL, 37, 38  
Rechnervorlauf, 91  
REPEAT, 125  
RESUME, 161  
Roboterkoordinatensystem, 67  
Rückwärtstransformation, 64

**S**

S und T, 77  
SAK, 77  
Satzkoinzidenz, 77  
Schaltaktion, 169  
Schleifen, 122  
SIGNAL, 132  
SIN(X), 56  
Sinus, Cosinus, Tangens, 56  
Softwareendschalter, 192  
Sperren / Freigeben, 156  
Sprunganweisung, 119  
SPS.SUB, 10  
SQRT(X), 56  
Status, 74, 77  
STRUC, 42  
Strukturen, 42  
SWITCH, 121

Synchron-PTP, 70  
Systemdateien, 58  
Systemvariablen, 58

**T**

TAN(X), 56  
TCP, 69  
TCP-Anpassung, 191  
Teachen von Punkten, 108  
Timer, 58  
Tool Center Point, 70  
Touch Up, 111  
Translationen, 66  
TRIGGER, 165  
Trigger, 165  
Turn, 74, 77

**U**

Überkopfbereich, 78  
Überschleif Bahn – PTP, 105  
Überschleif PTP – Bahn, 104  
Überschleifbeginn, 98  
Überschleifbewegungen, 94  
Überschleifen, 94  
Überschleifkontur, 94  
UND-Verknüpfung, 53  
Unterprogramme, 12, 145  
UNTIL, 125  
USER\_GRP.DAT, 11  
USER\_GRP.SRC, 11  
USERSPOT.SRC, 11

**V**

VAR, 110  
variabel + bahnbezogen, 84  
variabel + raumbezogen, 87  
Variablen und Namen, 33  
Variablen und Vereinbarungen, 33  
Vereinbarung, 12  
Vereinbarungsteil, 12  
Vergleichsoperatoren, 51  
Verstecken von Programmteilen, 22  
vordefinierte Strukturen, 44  
Vorlauf, 91  
Vorlaufstop, 92, 131  
Vorwärtstransformation, 64

**W**

WAIT, 128  
Warteanweisungen, 128  
WEAV\_DEF.SRC, 11  
Weltkoordinatensystem, 67  
Werkzeug, feststehendes, 69  
Werkzeugkoordinatensystem, 67  
Werkzeugwechsel, 107  
Wertzuweisung, 33  
WHILE, 124  
Wurzel, 56

**Z**

Zählschleife, 122  
Zeichenketten, 42  
Zweidimensionales Feld, 40  
Zyklische Flags, 59

