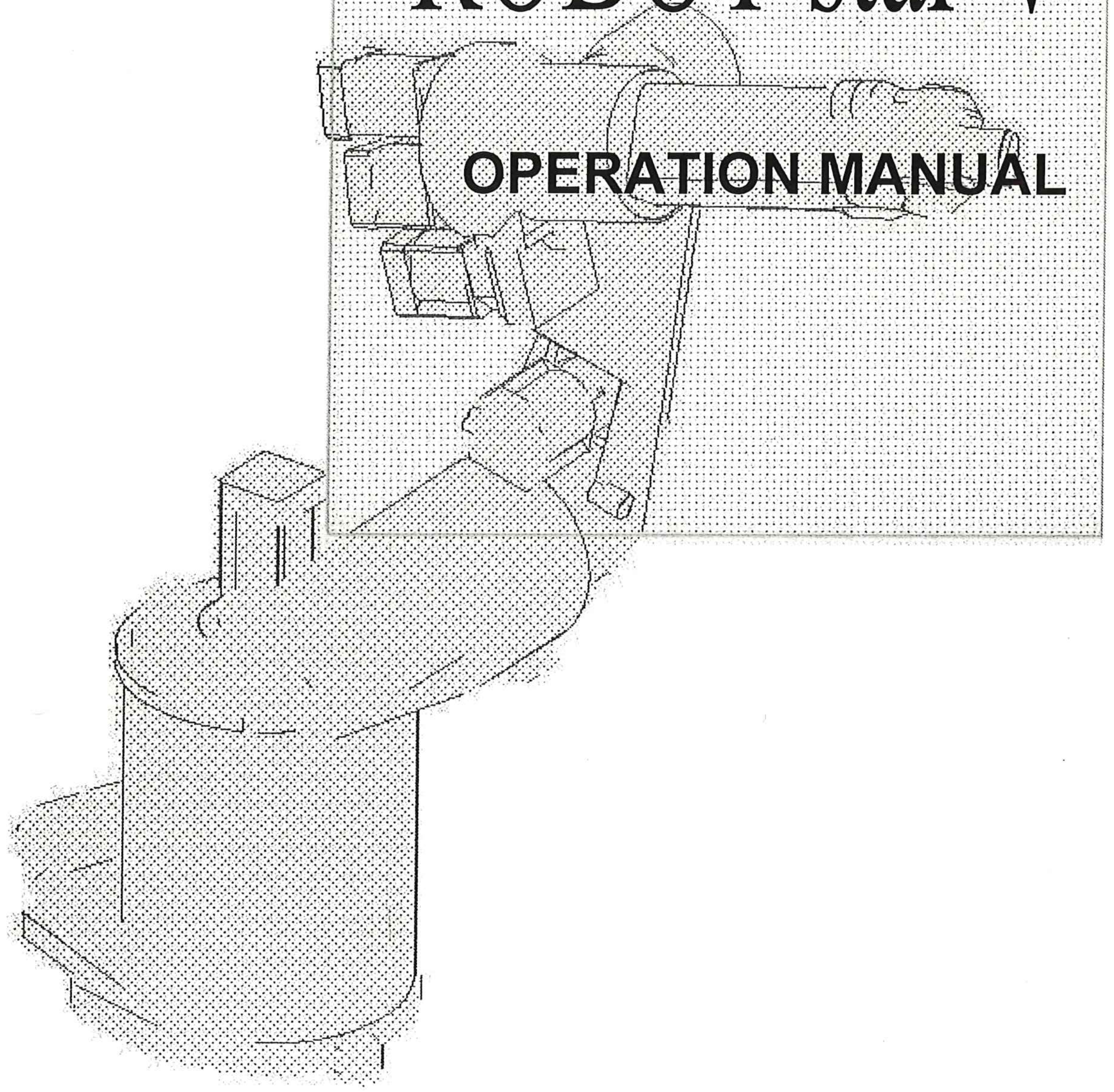
REIS ROBOTICS

ROBOT star V

OPERATION MANUAL



Version: RSV V7.x 1.2000 GB

REIS ROBOTICS

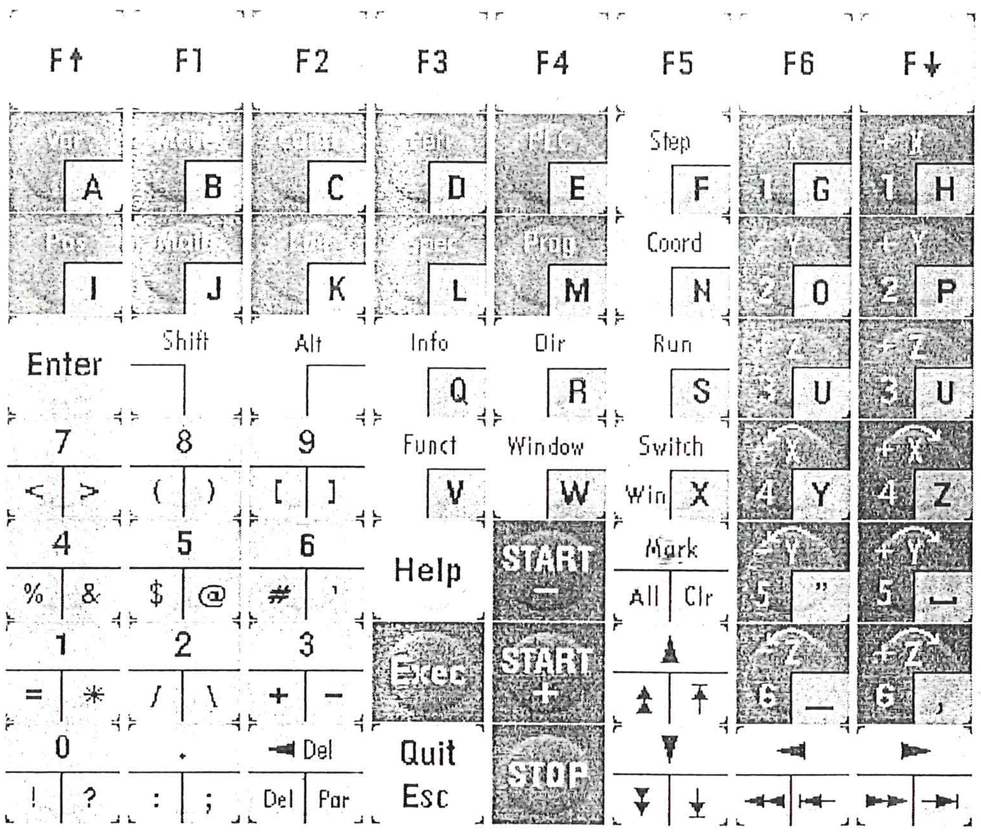
Operating manual ROBOTstar V

BDA-RSV/07.98/E

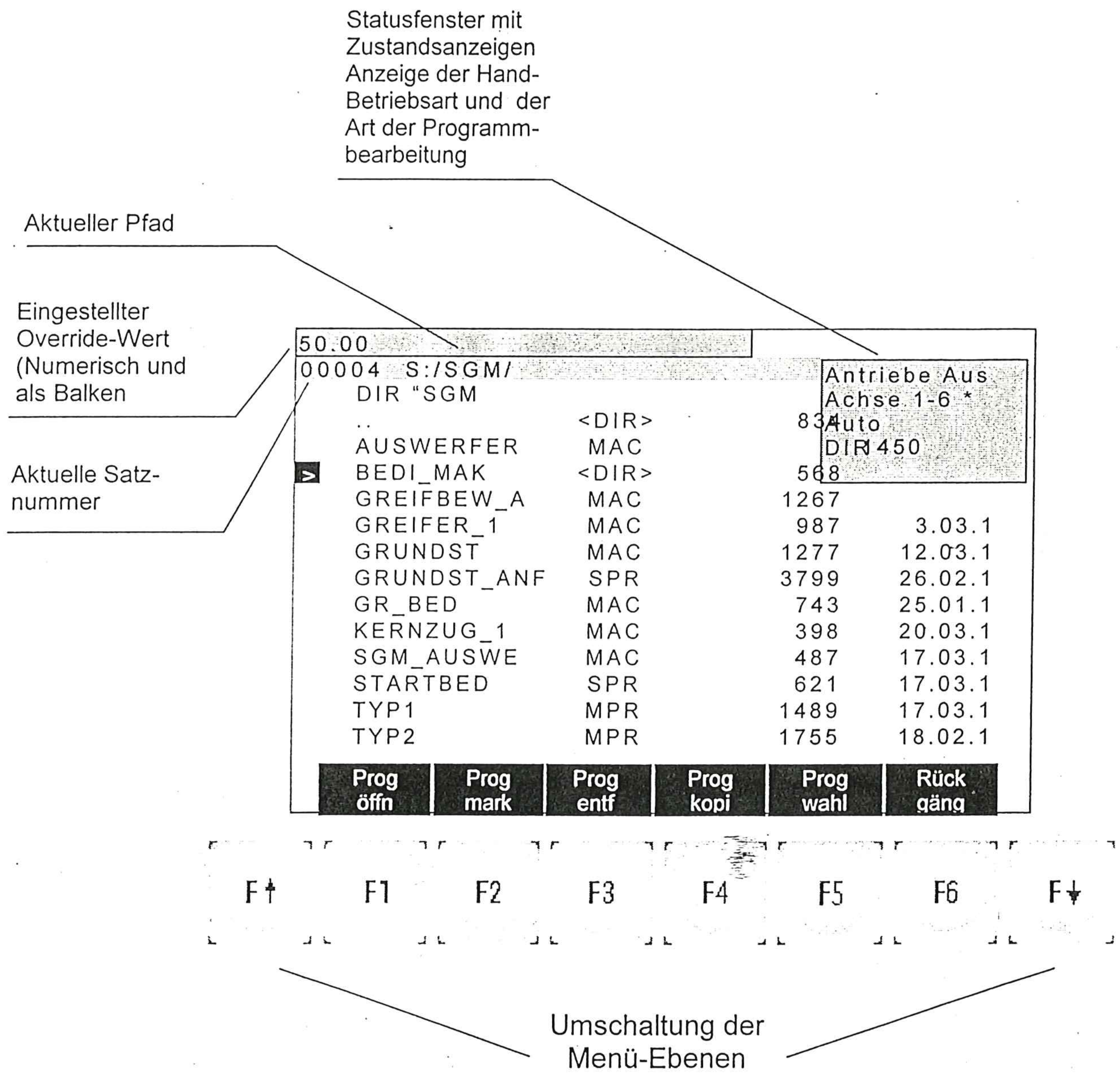
REIS ROBOTICS

1 Portable teach pendant PHG

1.1 Arrangement of the keyboard on the PHG



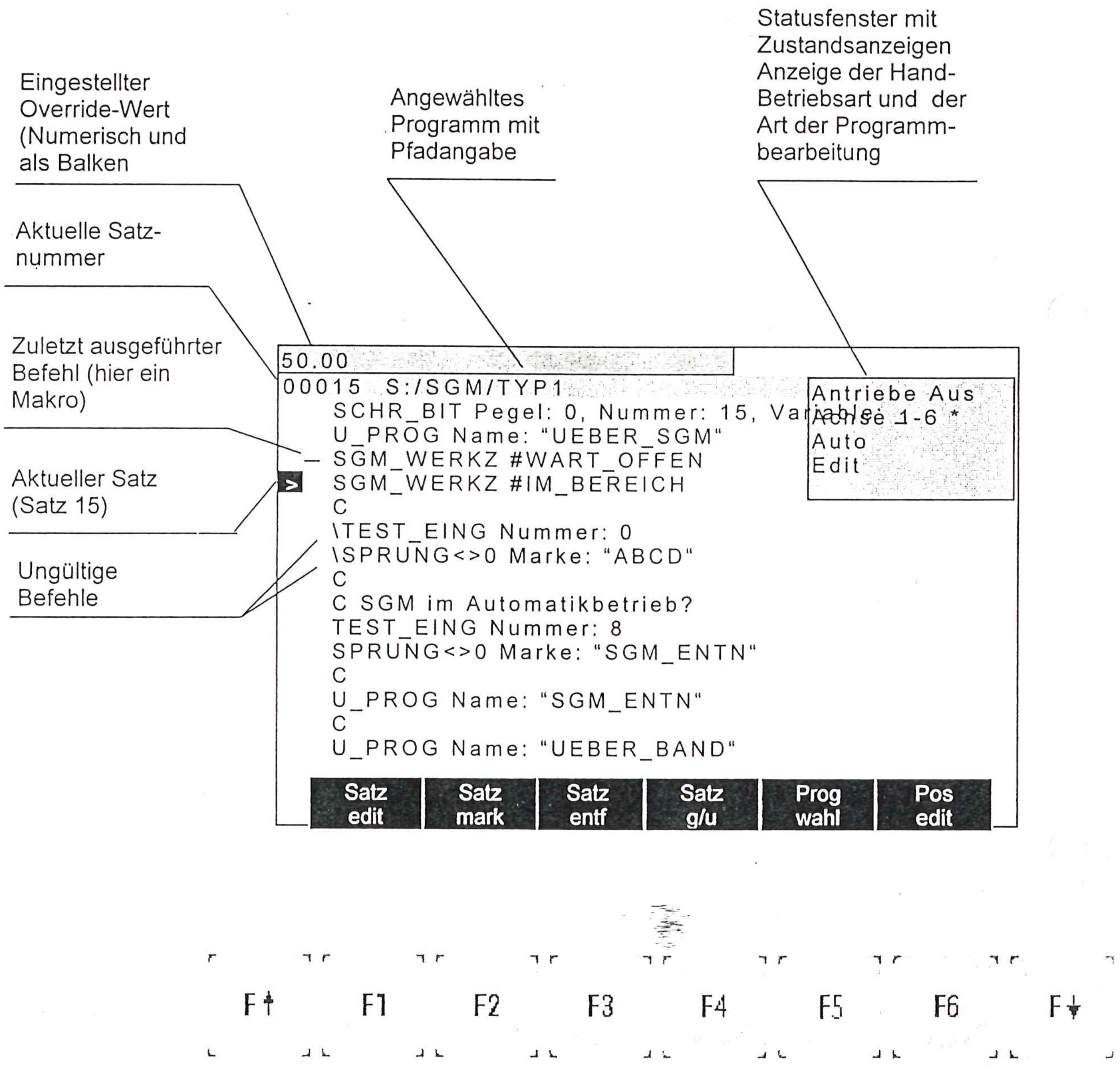
1.2 Display of the PHG DIR-Mode



```
50.00
00004  S:/SGM/                                     Antriebe Aus
     DIR "SGM                                      Achse 1-6 *
     ..                  <DIR>             834     Auto
     AUSWERFER            MAC                      DIR 450
>    BEDI_MAK            <DIR>             568
     GREIFBEW_A           MAC             1267
     GREIFER_1            MAC              987      3.03.1
     GRUNDST              MAC             1277     12.03.1
     GRUNDST_ANF          SPR             3799     26.02.1
     GR_BED               MAC              743     25.01.1
     KERNZUG_1            MAC              398     20.03.1
     SGM_AUSWE            MAC              487     17.03.1
     STARTBED             SPR              621     17.03.1
     TYP1                 MPR             1489     17.03.1
     TYP2                 MPR             1755     18.02.1
```

Statusfenster mit Zustandsanzeigen Anzeige der Hand-Betriebsart und der Art der Programm-bearbeitung
Aktueller Pfad
Eingestellter Override-Wert (Numerisch und als Balken
Aktuelle Satz-nummer
Prog öffn
Prog mark
Prog entf
Prog kopi
Prog wahl
Rück gäng
F↑
F1
F2
F3
F4
F5
F6
F↓
Umschaltung der Menü-Ebenen

1.3 Display of the PHG EDIT- Mode


Eingestellter Override-Wert (Numerisch und als Balken)
Angewähltes Programm mit Pfadangabe
Statusfenster mit Zustandsanzeigen Anzeige der Hand-Betriebsart und der Art der Programm-bearbeitung
Aktuelle Satz-nummer
Zuletzt ausgeführter Befehl (hier ein Makro)
Aktueller Satz (Satz 15)
Ungültige Befehle
Satz edit
Satz mark
Satz entf
Satz g/u
Prog wahl
Pos edit
F↑
F1
F2
F3
F4
F5
F6
F↓

```
50.00
00015  S:/SGM/TYP1
     SCHR_BIT Pegel: 0, Nummer: 15, Variable:
     U_PROG Name: "UEBER_SGM"
   — SGM_WERKZ #WART_OFFEN
>    SGM_WERKZ #IM_BEREICH
     C
     \TEST_EING Nummer: 0
     \SPRUNG<>0 Marke: "ABCD"
     C
     C SGM im Automatikbetrieb?
     TEST_EING Nummer: 8
     SPRUNG<>0 Marke: "SGM_ENTN"
     C
     U_PROG Name: "SGM_ENTN"
     C
     U_PROG Name: "UEBER_BAND"
```

```
Antriebe Aus
Achse 1-6 *
Auto
Edit
```

2 Operating modes of the control ROBOTstar V

2.1 Operating modes

The last line in the status window shows the current operating mode of the control. The mode changes according to the operating state. Depending on the active mode, there will be various possibilities for operation.

The two main modes are the following ones:

2.1.1 DIR - mode

The display of the teach pendant (PHG) shows the inputs of a directory. You reach the DIR-mode by operating the key "Dir".

2.1.2 EDIT - mode

The display of the teach pendant (PHG) shows the content of a program. You reach the EDIT-mode by selecting a program.

Change to further modes, in order to select operating functions, is possible from the DIR- as well as from the EDIT-mode. After selection of the function the control returns to the original mode.

2.1.3 RUN - mode

Is activated by operating the key "Run"
The following settings are possible in RUN-mode:

- kind of program processing (automatic, test)
- start of the integrated PLC
- switching on the VKE- and status display for the integrated PLC
- approach of the reference points
- synchronization resp. calibration of the robot axes

2.1.4 COOR - mode

Is activated by operating the key "Coord". This mode determines in which manner the robot is moved manually.

2.1.5 INFO - mode

Is activated by operating the key "Info".
The following information can be called in Info-mode:

- actual states of the robot
- contents of variables and PLC markers
- suppressed error messages
- actual states of binary inputs and outputs
- status window (switching on and off)

2.1.6 FKT - mode

Is activated by operating the key "Funct" .
This mode is used for the following operations:

- programming the system timer
- modification of the program protection
- interruption of archival storage
- reset of the DNC-interface

2.1.7 CWIN - mode

Is activated by operating the key "Window".
With this mode the display can be divided into two windows, resp. the division can be reset.

3 Movement modes of the robot

3.1 Movement modes of the robot

3.1.1 Keyboard

Depending on the selected movement mode the robot can be moved via the movement keys (red double row on the right hand side of the keyboard). Movement is executed in dead man operation, i.e. the robot only moves so long as the movement key remains operated. Simultaneous operation of several key is generally possible, in order for instance to move several axes at the same time.

The speed of movement can even be changed during movement with the override regulator.

3.1.2 Mouse

For intuitive movement of the tool into any random direction it is possible to have connected a 6D-mouse. This mouse, depending on the point of fixture, must be calibrated. For this option as a standard feature there is provided a fixture point on the PHG and up to two fixture points directly at the tool. Direct guiding of the tool is possible with the mouse.

3.1.3 Position Control
Only effective with movement mode Cartesian

With the special key "I" under the override regulator it is possible to switch over to Position Control . The first line in the display of the PHG is represented in green and the number value that was adjusted with the override regulator before (e.g. 54.00 %) gets the unit millimeter (e.g. 54.00 mm). If now for instance the key "+ X" is operated, the TCP can be moved - by turning at the override regulator - into X-direction (to the right to X plus, to the left to X minus). The number value in the first line indicates by how many millimeters the TCP moves with one turn of the override regulator. The speed can directly be influenced by the rotation speed at the override regulator. With Position Control also the orientation of the tool can be changed.

Position Control is also possible in conjunction with the 6D-mouse.

With the special key "O" switch over to the override mode is executed again.

4 Program types

4.1 Program types of the ROBOTstar V

4.1.1 MPR Main program

One step after the other is processed in automatic operation. The last step - "END" - includes the command for return to step 1, i. e. the program will be repeated. In the test operating modes running in sequential step mode, return to step 1 is executed after the end and the program stops.

Subprograms (command "CALL"), main programs (command "PROGRAM") and macros can be called in a main program.

4.1.2 SPR Subprogram

A subprogram can be called from a main program, from another subprogram or from a macro. Processing is started with step 1. The subprogram is processed step by step. The last step „END“ includes the command for return to the calling program. Return is made to the step following the call.

If a subprogram was selected by the operator, or if the subprogram was stopped and scrolling or another modification was made, no return will be possible with processing of the last step „END“. The following error message is displayed:

Operating error 4017

Further subprograms and macros can be called in a subprogram. (Nesting depth12!)

4.1.3

4.1.4 MAC Macro

A macro can be called from a main program, a subprogram or another macro. The call of a macro is programmed like a system command. The commands in a macro are processed like in a subprogram, but no program change is executed. In the single step modes a macro call is processed like a single command.

A macro must not contain any positions!
Further macros and subprograms can be called in a macro. (Nesting depth 12!).

4.1.5 PAL Pallet program

Contrary to other program types pallet programs have a fixed format. They must contain the following commands in the indicated order:

- four definitions for global or local integer variables
- three definitions for global or local vector variables
- three definitions for local integer constants
- one tool data step and
- four positions

The names of the variables and constants to be defined are freely selectable, observing the rules for type identification. Commentary lines can be added in the program.

From a pallet program the control calculates the following characteristics of the pallet:

- geometrical dimensions
- alignment of the pallet
- parts number in one row
- number of rows in one layer
- number of the layers
- total number of the parts
- distance of the parts towards each other
- distance of the rows towards each other
- distance of the layers towards each other

The pallet programs only serve for calculation of shifting vectors.

If a pallet shall be processed, the movement sequence for the first part of the pallet must be programmed. This movement sequence must be allocated to the corresponding pallet program. With the palletizing function, during program cycle, the entered positions are shifted by the calculated vector each. Calculation is made depending from internal counters which are modified with each access to the pallet (the four defined integer variables).

4.1.6 PLC PLC program

The control ROBOTstar V has an integrated PLC that is programmed like a robot program. When being once started, the PLC program is cyclically running in the background, independent from the selected operating mode and from state of the robot.

The total of commands approximately corresponds to the Siemens - PLC S5/100U.

The integrated PLC makes access to the same inputs and outputs like the robot program. Communication between PLC and robot program is ensured via system variable. Further system markers contain information about axis actual values of the robot, state of system inputs and system outputs, error states etc.

5 Programming of Movement Sequences

5.1 Programming of Movement Sequences

Movement sequence of the robot is realized via programmed positions in space. During program run the robot approaches the positions in such order as they are in the user program. When doing this, it always keeps the direct way from one position to the next.

5.2 Entering the actual position into the program

TOOL **(instruction set Move)**

Before a position can be taken over into a program, first of all a valid tool variable must be named. This is effected in the instruction set "Move" by selection of the command "TOOL" and subsequent input of the tool variable the name of which must begin with the letter "T" .

POSITION **(instruction set Pos)**

In order to enter a position, the TCP is moved to the required position in operating mode "AXIS 1 - 6" or "CARTESIAN". Additionally, orientation of the tool can be adjusted and optimized. For entering this position into the program it is sufficient to select the instruction set "Pos" and to press the key "ENTER". Thus, the position is taken over into the program as normal position.

The current actual position can be defined in the instruction set POS as follows:

- with F1 as normal position in robot coordinates	#N
- with F2 as oscillation auxiliary point in robot coordinates	#P

5.2.1 Move modes

5.2.2 INTERPOL (instruction set Move)

There are two totally different movement modes by means of which the TCP moves from one programmed position to the other.

PTP movements and CP movements
PTP - Point to Point; CP = Controlled Path

The movement mode PTP (point to point) is preadjusted as default value. PTP-movement is determined by the movement way of the axes only. They start at the same time and simultaneously stop in the corresponding end position. Therefore, the TCP does not follow a straight line but moves on a curved path (only with articulated arm robots).

CP means that the TCP will approach the next following position or the next following positions on a mathematically defined path. CP movements are interpolation modes "#LINEAR", "#SPLINE", "#ZIRK_MIN_ORI" and "#ZIRK_BAHN_ORI".

After selection of the command "INTERPOL" another menu is opened from which the required movement mode of the robot can be selected.

For movements on a straight line select movement mode "#LINEAR". The TCP will approach the next following position(s) on a straight line.
If orientation in the destination point is different from the start point, readjustment is made continuously during TCP movement.

The two circular interpolation modes differ due to the execution of orientation changes on the circular path. "#ZIRK_MIN_ORI" effects that orientation changes will be executed on the shortest possible way.

With "#ZIRK_BAHN_ORI" orientation change will be effected like on a straight line which has been curved in the shape of a circular path.

In order to enable the control to calculate the circular path, an auxiliary position on the circular path (circular auxiliary point) must be entered after the command "#ZIRK_MIN_ORI" or "#ZIRK_BAHN_ORI". It is approached and entered like a normal position. After the auxiliary position, the input of the end point of the circular path is effected. Several circular segments can be programmed directly one after the other.

After switch-on of the circular interpolation always an even number of positions must have been programmed before commutation of the operation mode is possible.
After "#ZIRK ... " the control interprets each first, third, fifth etc. position as circular auxiliary point, each second, fourth, sixth etc. position as end point of a circular path and simultaneously as start point of the next circular segment, until the movement mode will be switched over.

In the movement mode #SPLINE the TCP moves on a path that connects the programmed positions without any corners. (For detailed description please refer to the documentation "path control)

5.2.3 Example for circular interpolation

```
S1     MPR              BEWEGUNG (INTERPOL)
S2     WERKZEUG         T



S9     POSITION         A
S10    POSITION         B
S11    BEWEG_ART        #LINEAR
S12    POSITION         C
S13    BEWEG_ART        #ZIRK_BAHN_ORI
S14    POSITION         D
S15    POSITION         E
S16    POSITION         F
S17    POSITION         G
S18    BEWEG_ART        #PTP
S19    POSITION         H
```

If no movement mode is programmed at the beginning of a program, all following positions will be approached in the default adjustment #PTP. In the example the TCP will approach the positions A and B in movement mode PTP. Position C in step 12 will be approached on a straight line. The movement mode ZIRK_BAHN_ORI in step 13 effects that an arc of a circle will be calculated from the positions C, D, and E on which the TCP approaches position E in step 15. Since circular interpolation is still active, another arc of a circle will be calculated from the positions E, F, and G which is directly attached to the first one. Position G will be approached on this second arc of a circle. Position H and the following ones will be approached in movement mode PTP again.

5.2.4 Velocities and accelerations

PTP_VELOC<value in %>
PTP_ACCEL <value in %>

PATH_VELOC <value in mm/sec>
PATH_ACCEL <value in %>

Speed and acceleration can be programmed for the movement modes PTP and CP. The corresponding instructions are to be found in the instruction set "Move".

The PTP-speed is programmed in percent. The percentage refers to the maximum axis speed of the robot determined in the machine data.

The PTP-speed is no dimension for the speed of the TCP. It only defines the speed by which the pilot axis moves. The pilot axis is the axis having the greatest movement path. The TCP moves more slowly or faster depending on its distance from the rotary point of this axis.

Acceleration for PTP-movements is also programmed in percent. The reference parameter is the maximum axis acceleration which is also defined in the machine data.

Acceleration is not only a dimension for the increase of velocity when starting from a position, but also the dimension for the deceleration by which the axes reduce the speed before reaching a programmed position.

The speed for CP-movements is programmed in millimeters per second. This speed is valid for the TCP.

The path acceleration is also valid for the TCP. It is programmed in percent. The maximum value is defined in a system variable.

6 Displacement of programmed movement sequences

6.1 RELATIVE <name of the vector variable>

Programmed positions can be displaced by a vector. The shifting values must be filed in a vector variable. A vector variable contains the following values:

Component	Description	Unit
1	Frame	
2	shifting in X-direction	[mm]
3	shifting in Y-direction	[mm]
4	shifting in Z-direction	[mm]
5	modification of the orientation angle A	[degree]
6	modification of the orientation angle B	[degree]
7	modification of the orientation angle C	[degree]

The content of the vector variables can be modified with the commands "COPY_OFFS", "VEC_ADD" and "VEC_SUB".

After selection of the command "RELATIVE" of function group "Move" the name of the vector variable has to be entered in which the shifting values are stored.
Before the relative shifting can be activated in the user program, valid values must be allocated to the components of the vector variables.
After treatment of the command "RELATIVE V...", the values contained in the vector variable V... will be added to all following positions, and the robot approaches the positions resulting from this.
The activated shifting is maintained until another vector variable will be activated or until shifting is switched off completely. For switching off the relative shift the following command must be programmed:

"RELATIVE Vector: _VNULL

_VNULL is a system variable and contains the value 0 in all six components.

6.2 CALC_REL <name of the vector variable>

With the command "CALC_REL" the relative shift between the actual position of the TCP and the next programmed position is calculated. The calculated shifting values will be stored in the vector variable which is named in the command.

Calculation of the vector is made as follows:

Relative vector = actual position - programmed position

6.3 Example:

```
S12              .
S13              .
S14              .
S15    POSITION    A
S16    CALC_REL   VEKTOR_1
S17    POSITION   B
S18    POSITION   C
S19    RELATIVE   VEKTOR_1
S20    POSITION   C
S21              .
 .               .
S28    RELATIVE    _VNULL
S29    POSITION   X
                 .
                 .
```

When treating step 15 the robot approaches position A. Step 16 effects the calculation of the shifting vector between positions A and B (values of position A minus values of position B), and the calculated values will be entered into the variable "VEKTOR_1". Position B in step 17 serves for calculation only and is not approached. Subsequently the robot approaches position C in step 18. Displacement is not yet activated, this will only be effected in step 19.
In step 20 position C is approached once again, but shifted by the calculated vector. All following positions will also be approached shifted by the a.m. vector.
In step 28 the relative shift is switched off again. All the following positions won't be displaced any more.

7
Control of the peripheral equipment

Digital signals are used for communication between peripheral equipment and robot control.

User inputs provide the control with information about the peripheral conditions (e.g. finish message of a processing machine, rotary table in correct position, clamping device closed, deposit free etc.).

User outputs give information to the peripheral equipment and execute commutations (e.g. signal to a processing machine, that the robot is moving into the operating range of the machine, switching on or off a conveyor, switching on a spindle, open/close gripper, feed out pallet etc.)

Analog outputs are used for regulation of peripheral equipment by means of a variable direct voltage, e.g. speed of a machine, weld parameters etc.

Also direct voltages coming from peripheral equipment units can be interrogated in the user program.

7.1 Organization of the binary inputs and outputs

Pick-up of the inputs and outputs is effected via two system variable arrays:

Binary inputs:	**_IBIN_IN[10]**
Binary outputs:	**_IBIN_OUT[10]**

Both arrays consist of 10 variables with 4 bytes each.
Each bit of this variable represents an input resp. output.

The inputs and outputs are picked up via byte- and bit numbers. I.e., addressing of 40 bytes is possible. The **bytes** are numbered from **0 to 39.**
In each byte there are available the **bits** number **0 to 7.**

The following instructions are used for direct access to inputs and outputs:

WRITE_BIT	command group Log
TEST_BIT	command group Contr
WAIT_BIT	command group Contr

A system equate has to be entered as first parameter:

#AUSGANG	access to a user output
#EINGANG	access to a user input

Addressing is made in the two following parameters:

Byte:	Byte number of the input resp. output (0 to 39)
Bit_Nr:	Bit number of the a.m. byte (O to 7)

The functions of the inputs and outputs are shown in the circuit diagram of the robot. Numbering of the inputs and outputs in the robot program and in the circuit diagrams are identical.

Example: Setting an output:

0,,,39   0,,,7

WRITE_BIT #AUSGANG, Pegel: 1, Byte: 5, Bit_Nr: 3

With this command that output would be set that is indicated with 5.3 in the circuit diagram and that is shown in the variable _IBIN_OUT[2] in bit 11.

7.1.1 Allocation of the binary inputs and outputs to system variables

by means of the example of the user inputs

Variable	Byte - number			
_IBIN_IN[1]	3 Bit 76543210	2 Bit 76543210	1 Bit 76543210	0 Bit 76543210
_IBIN_IN[2]	7 Bit 76543210	6 Bit 76543210	5 Bit 76543210	4 Bit 76543210
_IBIN_IN[3]	11 Bit 76543210	10 Bit 76543210	9 Bit 76543210	8 Bit 76543210
_IBIN_IN[4]	15 Bit 76543210	14 Bit 76543210	13 Bit 76543210	12 Bit 76543210
_IBIN_IN[5]	19 Bit 76543210	18 Bit 76543210	17 Bit 76543210	16 Bit 76543210
_IBIN_IN[6]	23 Bit 76543210	22 Bit 76543210	21 Bit 76543210	20 Bit 76543210
_IBIN_IN[7]	27 Bit 76543210	26 Bit 76543210	25 Bit 76543210	26 Bit 76543210
_IBIN_IN[8]	31 Bit 76543210	30 Bit 76543210	29 Bit 76543210	28 Bit 76543210
_IBIN_IN[9]	35 Bit 76543210	34 Bit 76543210	33 Bit 76543210	32 Bit 76543210
_IBIN_IN[10]	39 Bit 76543210	38 Bit 76543210	37 Bit 76543210	36 Bit 76543210

7.2 Switching of digital outputs

As already described, switching of digital outputs is effected via the system variables _IBIN_OUT. If a bit is set (level 1) or reset (level 0) in one of the variables, then the CAN bus switches on resp. off the programmed output at the corresponding module (node). 8 or a multitude of outputs can be connected to a CAN bus module, i.e., the connections of a module correspond to one or several bytes of the system variable _IBIN_OUT. Programming therefore is made via setting resp. resetting of the corresponding bytes and their bits. This is executed with the command WRITE_BIT of the command group Log.

7.2.1 Switching on an output

The output 1.6 (acc. to circuit diagram switching on a belt) is to be set.
The following steps must be entered:

Select command "WRITE_BIT"
Select "#AUSGANG" in the opened menu
enter as level a "1"
enter byte number "1"
enter bit number "6"

Then, the command stands in the program as follows:

WRITE_BIT #AUSGANG, Pegel: **1**, Byte: **1**, Bit: **6**

7.2.2 Switching off an output

The a.m. belt shall be stopped. For programming proceed as mentioned above, only a 0 has to be entered as level. The command is as follows:

WRITE_BIT #AUSGANG, Pegel: **0**, Byte: **1**, Bit: **6**

7.3 Inquiry of digital input signals

The input signals are treated in the same manner like the output signals. Bits are set resp. reset according to the node number via the CAN bus. I.e., if a connection is applied with 24 volt on the hardware side, then the corresponding bit is set, if 0 volt are applied, it is reset.

A different reaction on input signals is possible:
Waiting for a signal in the program sequence is possible or a conditional branch can be executed depending on an input signal.

7.3.1 Waiting for input signal

Before a part can be removed from a processing machine, the control waits until the signal "processing finished" is given. Acc. to circuit diagram this signal is active on input 0.5.

The following must be programmed:

WAIT_BIT #EINGANG, Pegel: **1**, Byte: **0**, Bit_Nr: **5**, Max_Zeit: **0.0**, Marke: **X**

The program will stop at this command so long until 24 volt will be applied at the input 0.5. Then, processing of the program will be continued with the next step. The parameter "Max_Zeit" (Max_Time) is indicated with 0.0 here, which means a waiting without time limitation.

Another value can be programmed in the parameter "Max_Zeit":

WAIT_BIT #EINGANG, Pegel: **1**, Byte: **0**, Bit_Nr: **5**, Max_Zeit: **10.0**, Marke: **ABC**

Here, the program also stops and waits for the signal. When the signal is active within the indicated time (10 seconds in the example), processing is continued with the next step. If the programmed time is exceeded without the signal being given, then a program branch is executed to that label that was indicated in the last parameter (ABC in the example).

7.3.2 Conditional branch depending on the input signal

It is possible to program program branches that are only executed when level 1 resp. level 0 is applied to an input.

This is realized with the command "TEST_BIT"

After selection of the command and input of the parameter "#EINGANG the branch condition must be programmed:

#=0	the branch is executed when 0 volt are applied to the input
#=1	the branch is executed when 24 volt are applied to the input

Then, byte and bit number must be indicated, and as last parameter the label name the branch shall be executed to.

If the branch condition is not fulfilled, processing of the program is continued with the next step.

```
TEST_BIT #EINGANG, #=1, Byte: 0, Bit_Nr: 6, Marke: "CDE"
U_PROG Name: "REINIGEN"
MARKE "CDE"
```

In the above example the subprogram REINIGEN (CLEANING) is only executed when no 24 volt are applied to the input 0.6 . Otherwise the subprogram call is omitted.

7.4 Control of the analog outputs

ANA_OUTP <rated voltage in volts, number of the analog output>

In the instruction set Peri the output of a direct voltage can be programmed at an analog output with this command. After selection of this command first the voltage has to be indicated in Volts (-10 volt to +10 volt) and then the number of the analog output has to be entered where the voltage shall be applied.

Voltage can be entered via the numeral keys, but it can also be taken from a real constant or from a real variable; in this case the first parameter to be indicated is the name of the constant or of the variable.

7.5 Inquiry of analog inputs

ANA_INP <number of the analog input, name of the destination variable>

The analog voltage which is applied to an analog input can be filed in a real variable for further treatment.

Selection of the command "ANA_INP" is effected from the function group "Peri". After selection, first of all the number of the analog input the analog voltage shall be taken over from has to be entered, and then, the name of the variable where the voltage value is to be stored has to be indicated.

Example:

```
S1    MPR          ANALOG_TEST
S2    LOK_VAR      RDRUCK
.     .            .           .
S14   POSITION
S15   SCHR_BIT #AUSGANG, 1, 0, 4
S16   MARKE        WIEDERHOLE
S17   ANA_EING     2,RDRUCK
S18   TESTE #VARIABLE, RDRUCK, #<, 5.0, WIEDERHOLE
S19   U_PROG       BESCHICHTE
.     .            .           .
S39   ENDE
```

In step 16 a dosing system is switched on via output 0.4 which gradually builds up the required operating pressure for coating. The current pressure is signalized to analog input 2 via an analog signal.
The applied analog voltage is loaded in the variable RDRUCK in step 17 and is compared with the minimum value 5.0 volt in step 18. The subprogram for coating is only called after this minimum voltage is applied at analog input 2, i.e. when the minimum pressure has been reached.

8 Program calls

8.1 Change into a main program

PROGRAM <name of the main program>

The command "PROGRAM" from the command group „Prog.“ effects a branch from a main program to step 1 of another main program. After selection of the command the name of the main program has to be entered change shall be effected to. If the program with the indicated name does not exist or if the indicated program is no main program, corresponding error messages will be emitted during run in automatic or test mode.

8.2 Call of a subprogram

CALL <name of the subprogram>

After selection of the command "CALL" from the command group „Prog“ the name of the subprogram you want to branch to must be entered. In automatic or test mode branching is effected to step 1 of the indicated subprogram when processing this command. This subprogram will be treated, and after the last step of the subprogram the calling program will be returned to. Program treatment here is continued with the command following the subprogram call.

8.3 Program call via indirect addressing

The indirect program call for RSV is made via a string variable. For definition of this variable (e.g. in program VAR) the string length must be indicated in round brackets. The name starts with S.
Example:

> **VAR** Name: **S_PROG(20)**

With the **COPY**- instruction a string with max. 20 characters can be copied into this variable. The string must stand in inverted commas.
Example:

> **KOPIERE** Quelle: **"BEREICH_1"**, Ziel_Var: **S_PROG**

Then, the program the name of which is contained in the variable "S_PROG" can be called with the instruction **PROGRAM** resp. **CALL.**

It is important that the variable name does **not** stand between inverted commas, because then the control would search for the program with the name S_PROG!
Examples:

> **PROGRAM** Name: **S_PROG** **(no inverted commas!!)**
> (here, now the main program BEREICH_1 would be called)
>
> **KOPIERE** Quelle: **"3344"**, Ziel_Var: **S_PROG**
> **CALL** Name: **S_PROG**
> (here, the subprogram 3344 would be called)
>
> **KOPIERE** Quelle: **"3344"**, Ziel_Var: **S_PROG**
> **CALL** Name: **"S_PROG"**
> (here, the subprogram S_PROG would be searched!)

9 Programmed waiting time

WAIT <time in sec>

Delay times are programmed with the command "WAIT" of the instruction set "Contr" . As parameter that time has to be entered in seconds within which the program shall be stopped before the program cycle will be continued with the next command. The waiting time is effective in automatic mode and in all test operating modes.

10 Branches within a program

Conditional and unconditional (absolute) branches can be programmed within a program. For all branch commands a branch destination must be indicated from where program treatment shall be continued. It does not matter whether the branch shall be executed forward or backward. Branches from one program into another are not possible.

10.1 Branch destination

LABEL <name of the destination mark>

The command for the branch destination is named "LABEL". It is selected from function group "Contr" and after input of a freely selectable name (e.g. LABEL_1) is taken over into the program with key "ENTER". Within one program, another name has to be entered for each command "LABEL". Since branches from one program to another are impossible, same names can be used in different programs.

10.2 Absolute branch

BRANCH <name of the destination mark>

For programming of an absolute (unconditional) branch the command "BRANCH" has to be selected and the name of the label (branch destination) has to be entered (e.g. LABEL_1"). The indicated label must be in the same program as the branch command. When this command is processed in automatic or test mode, processing of program is continued at that branch label that has the same name. If no label with the same name exists in the program, the corresponding error message will be given.

10.3 Conditional branches

Contrary to the absolute branches, with the command for a conditional branch decision is made up whether the branch is executed or whether processing of the program is continued with the next step. The condition may be the state of a single bit or the content of a byte resp. of a variable.

10.3.1 Test of a bit

In order to test a single bit and to execute a branch depending on its state (0 or 1), the command "TEST_BIT" from the group "Contr" must be selected. After selection the following must be entered:

Selection of the operand that contains the bit to be tested.

#EINGANG	test of a user input
#AUSGANG	test of a user output
#MERKER	test of a PLC - marker
#VARIABLE	test of a bit in an integer variable

Selection of the branch condition

#=0	branch, when the bit is not set
#=1	branch, when the bit is set

Input of the byte number, when the input, output or marker was indicated as operand resp. input of the variable name, when variable is programmed as operand.

Input of the number of the bit that has to be tested

Input of the branch destination

Examples:

TEST_BIT #MERKER, **#=1**, Byte: **52**, Bit_Nr: **5**, Marke: "**ABC**"

When the PLC marker 52.5 is set, a program branch to the label ABC is executed, otherwise processing of the program is continued with the next step.

TEST_BIT #VARIABLE, **#=1**, Variable: **I123**, Bit_Nr: **5**, Marke: "**ABC**"

When the bit 5 of the variable I123 is set, a program branch to the label ABC is executed, otherwise processing of the program is continued with the next step.

10.3.2 Test of a byte or of a variable

In order to test the content of a complete byte or a variable and to execute a branch depending from this, the command "TEST" from the group "Contr" must be selected. In principle this command compares the content of the byte resp. of the variable with a number and depending on the result executes the branch or not. After selection the following has to be entered:

Select what shall be tested.

#EINGANG	test of an input byte
#AUSGANG	test of an output byte
#MERKER	test of a marker byte
#VARIABLE	test of a variable

Input of the operand 1 (the number) the operand 2 shall be compared to

Selection of the branch condition

#=	branch when both operands are identical
#<>	branch when both operands are different
#<	branch when operand 1 is smaller than operand 2
#>	branch when operand 1 is higher than operand 2
#<=	branch when operand 1 is equal to or smaller than operand 2
#>=	branch when operand 1 is equal to or higher than operand 2

Enter the byte number, when input, output or marker are the parameters to the tested or enter the variable name, when a variable shall be tested.

Input of the branch destination

When a variable shall be tested, then as first operand its name can be indicated and as second operand the comparison parameter.

Examples:

TESTE #AUSGANG, Op_1: **0**, **#<**, Byte: **1**, Marke: "**ABC**"

When no output is set in the output byte 1 (output 1.0 to 1.7), a branch is executed to the label ABC; when at least one output of this byte is set, processing of the program will be continued with the next step.

TESTE #VARIABLE, Op_1: **I123**, **#<**, Op_2 **10**, Marke: "**ABC**"

When the number indicated in the variable I123 is smaller than 10, a branch is executed to the label ABC , otherwise processing of the program will be continued with the next step.

11 Variables

A variable is a parameter that can be changed. There is a multitude of applications when programming, e.g. counting functions, indirect program call, simultaneous inquiry of several inputs, relative shiftings etc. are realized with variables.

11.1 Variable types

Variables may contain different parameters. The variable type determines the kind of variable content. Identification of the variable type is given with the first letter of the variable name:

I	Integer variable; contains integer number (32 bit)
R	Real variable; contains real number with digits after the comma (64 bit)
V	Vector variable; contains shifting vector
P	Position variable; contains position in space
T	Tool variable; contains tool data
S	String variable; contains a character string

11.1.1 Integer variables

An integer variable contains a binary number with 32 bits. The representable numbers are in the range from -2 147 483 648 to +2 147 483 647.
The bit with the highest value (bit 31) is the sign bit, i.e., as long as this bit is 0, a positive number stands in the variable, if this bit becomes 1, there is a negative number.

If all bits, except bit 31, are set, then the number +2 147 483 647 stands in the variable. If only bit 31 is set and all others are 0, then the variable content corresponds to the decimal number -2 147 483 648. If all 32 bits are set, the variable contents is -1.

11.1.2 Real variables

A real variable consists of 64 bits. The representable numbers are in the range from -1.2 multiplied by 10 to the 38^{th} to +1.2 multiplied by 10 to the 38^{th} . The bit with the highest value (bit 63) also in this case is the sign bit. The remaining digits according to the IEEE format (say EI TRIPPEL I) are converted into a decimal number with digits after the comma.

11.1.3 Vector variable

This variable consists of 7 components. The first component (integer) contains the frame-number (coordinate system). The next three components (real) contain the shifting values in X-, Y- and Z-direction (unit mm). The components 5-7 (real) contain differential values for the orientation angles A, B and C (unit degree).

11.1.4 Position variable

This variable consists of 36 components in total composed of the following: Position type, frame, number of main and additional axes, coordinates of the TCP, tool data and positions of the axes (unit increments).

11.1.5 Tool variable

Consists of 16 components. The components 4, 8 and 12 contain the position of the TCP referring to the tool flange.

11.1.6 String variable

Contains a random character string. The length of the variable must be entered after the name in a bracket, e.g. STRING(20). The variable STRING might contain a maximum of 20 digits.

11.2 Definition of variables

Variables can be defined in any program. Variable definitions must always be made at the beginning of the program. Exceptions are commentary lines, they may stand in front of definitions.

With definition the range of validity (global or local) and the name of the variable are defined, the first character determining the type.

11.2.1 Global variables

A variable is defined as global variable with the command "VAR" from the command group "Var" .

e.g. **VAR** Name: **I123**

Global means, that access to this variable with the name I123 is possible an any program. The name of a global variable must exist only once as variable name in the complete program memory.

11.2.2 Local variables

The command "LOC_VAR" also from the program group "Var" defines a variable as local.

e.g. **LOC_VAR** Name: **IABC**

Local means, that access to this variable is only possible in that program where it is defined, or in the sub-programs that are called from this program.

11.3 Description of variables

The definition of a variable is only a reservation, i.e., the variable does not yet contain any valid values. First, it must be described (initialized) before it can be used.

11.3.1 Description of complete variables

The command "COPY" from the command group "Var" copies a source operand into a variable

e.g. **KOPIERE** Quelle: **100**, Ziel_Var: **I123**

After execution of this command the variable I123 contains the numeric value 10.

KOPIERE Quelle: **T**, Ziel_Var: **T1**

This command effects that the 16 components of the tool variable T are copied into the tool variable T1.

11.3.2 Description of individual components

If individual components of tool-, vector- or position variables are to be described, then the command "COYP_OFFS" from the command group "Var" must be programmed. This command also allows to copy individual components from tool vector or position variables.

5 parameters must be entered after selection of the command COPY_OFFS:

number of the component that shall be copied (source index)
name of the source variable (source)
number of the component of the destination variable (destination index)
name of the destination variable (destination)
number of components that shall be copied (number)
(including named component)

Examples:

COPY_OFFS Quellindex: **2,** Quelle: **V1,** Zielindex: **2,** Ziel: **V2,** Anzahl: **3**

The second, third and fourth component of the vector variable V1 are copied into the variable V2 .

COPY_OFFS Quellindex: **1,** Quelle: **100.0,** Zielindex: **12,** Ziel: **T1,** Anzahl: **1**

Hand length 100 mm is copied into the tool variable T1.

12 Mathematical operations

For arithmetical operations the 4 fundamental operations are available. The commands are in function group "Math" . When executing the arithmetical commands, operand 1 (variable, constant or entered number) is calculated with the content of the destination variable and the result is written into the destination variable.

The commands for arithmetical operations are:

ADD <Quelle, Zielvariable> Zielvariable = dest. variable plus operand 1
SUB <Quelle, Zielvariable> Zielvariable = dest. variable minus operand 1
MUL <Quelle, Zielvariable> Zielvariable = dest. variable multiplied by operand 1
DIV <Quelle, Zielvariable> Zielvariable = dest. variable divided by operand 1
MODULO <Quelle, Zielvariable> Zielvariable =division rest of the division dest. variable by operand 1.
NEG negation of the variable content (negation of sign)

13 Processing of vector variables

VEC_ADD
VEC_SUB
VEC_VALUE

The commands "VEC_ADD" resp. "VEC_SUB" effect that the individual components of two vector variables will be calculated with one another (by addition resp. by subtraction). The result of the executed operation is in the vector variable being named as second operand.

Example:

The vector variables VEC1 and VEC2 contain the following values:
(angles A,B and C are equal in both variables)

	VEC1	VEC2
displacement in X-direction	150	200
displacement in Y-direction	-30	100
displacement in Z-direction	10	30

After the sequence of commands

```
VEC_ADD         VEC1,VEC2
RELATIVE        VEC2
```

the following positions will be approached displaced in X-direction by 350 mm, in Y-direction by 70 mm and in Z-direction by 40 mm.

With the command "VEC_VALUE" the absolute length of a vector can be calculated and entered in a real variable.

Example:

The vector variable "VNAHT_1" contains the following components:

displacement in X-direction	80 mm
displacement in Y-direction	60 mm
displacement in Z-direction	0 mm

After execution of the command

```
VEC_VALUE     VNAHT_1,RLAENGE
```

the variable RLAENGE contains the value 100.0.

With the command "VEC_VALUE" the length of a straight distance can be determined, for instance.

Programming example:
At the end of a programmed linear movement the length of the distance A --> B shall be entered in the variable "RWEG".

```
POSITION        A              <start point>
INTERPOL        #CP_LIN
POSITION        B              <end point>
CALC_REL        V_BAHN
POSITION        A
VEC_VALUE       V_BAHN,RWEG
```

14 Logic Operations

An integer variable contains a 32-digit binary number. This binary number (operand 2) can be linked logically with another number (operand 1). The operation is effected in such a manner that each bit of operand 1 is linked with the equivalent bit of operand 2. The result is again a 32-digit binary number being filed in operand 2.

Contrary to the arithmetical operations, the content of the named integer variable will be interpreted as a number without sign.

Operand 1 may be:

an integer constant	
an integer variable	
a decimal number,	which is entered via the numeral keys
a hexadecimal number,	After the number the letter "H" must be entered.
a binary number,	After the number the letter "B" must be entered.

The content of the variable can be linked via AND-function, OR-function and EXCL_OR-function. The result of the linkage is always in the destination variable (2nd operand).

14.1 AND-linkage

AND <integer value, variable name>

32 bits of one variable are linked simultaneously with the 32 bits of the 1st operand. When a bit of the integer variable is linked with 1 via AND, the value of this bit is taken over into the result. If a zero is in this bit, the result becomes zero. If a 1 is in this bit, the result becomes 1.

With an AND-operation with zero the result is always a zero, independent of the bit status.

The command "AND" is selected from instruction set "Log".

First the 1st operand has to be entered (the number resulting from the 32 individual bits). Then, the variable is named the content of which shall be linked with operand 1.

14.2 OR-linkage

OR <integer value, variable name>

The 32 bits of a variable are linked with the 32 bits of the 1st operand via the OR-function. For each bit of the variable the following is valid:
With an OR-operation with 1 the result is always a 1, independent of the bit status.
With an OR-operation with zero the value of the bit is taken over into the result. If a zero is in this bit, the result is a zero; if a 1 is in this bit, the result is a 1.

The command "OR" is selected in the 1st menu of instruction set "Log" with function key F2. After input of the 1st operand (the number resulting from the 32 individual bits) and a comma, the variable has to be named that shall be linked with the 1st operand.

14.3 EXCL_OR-linkage

EXCL_OR <integer value, variable name>

The 32 bits of a variable are linked with the 32 bits of the 1st operand via the EXCLUSIVE-OR-function. For the individual bits of the variable the following is valid:

With an EXCLUSIVE-OR-operation with 1 the result is zero when the value of the bit is a 1. The result becomes 1 when the value of the bit is a zero.

With an EXCLUSIVE-OR-operation with zero the result becomes 1 when there is a 1 in the bit; the result becomes zero when the value of the bit is a zero.

Briefly summarized - the result is 1 when both bits being linked via EXCLUSIVE-OR have various states; it is zero when both bits have the same value (both bits 0 or both bits 1).

14.4 Inversion of variable contents

INVERT <variable name>

The 32 bits of an integer variable are inverted, i. e., bits the content of which is zero become 1 and bits the content of which is 1 become zero. It must be observed that the highest-value bit (2**31) serves as sign bit. As long as this bit is zero, the variable content is interpreted as positive number, when it is 1, the content is interpreted as negative number. "INVERT" must not be mixed up with the command "NEG" from instruction set "Math".

The command "NEG" only converts the sign of the variable content, i. e., the amount remains the same (conversion in two's complement). The command "INVERT" changes the content of the variable in one's complement, i.e., the amount changes. All bits being logic zero become logic one and vice versa.

14.5 Shifting operations

SHIFT_L <number of digits, variable name>

The 32 bits of an integer variable can be shifted to the left with this command. In the first parameter you must indicate by how many digits shifting shall be effected. Each digit corresponds to a multiplication by 2 (1 digit = *2; 2 digits = *4; 3 digits = *8; 4 digits = *16 a.s.o.). The bits which exceed the 32 bits due to shifting to the left will be ignored and the "gaps" at the right side resulting from shifting to the left will be filled up with zeros.
The highest value bit (2**31) also here is the sign bit.

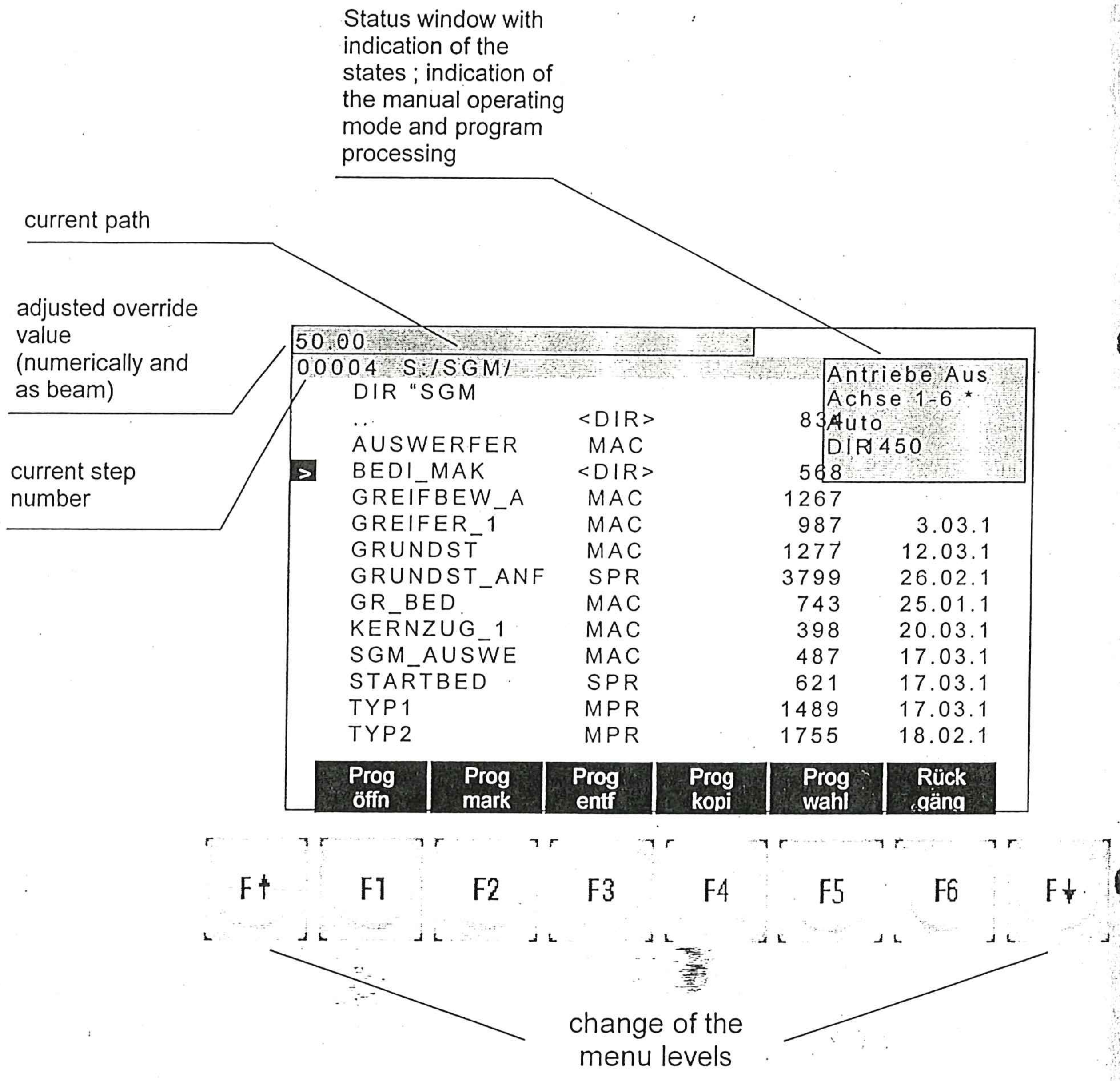
SHIFT_R <number of digits, variable name>

The 32 bits of an integer variable are shifted by n digits to the right. Each digit corresponds to a division by 2 (1 digit = :2, 2 digits = :4, 3 digits = :8 a.s.o.). The bits falling away due to the shift to the right are ignored and the "gaps" resulting on the left side will be filled up with zeros.

After selection of "SHIFT_R" in the 2nd menu of instruction set "Log" with function key F2 the first parameter to be indicated is the number of digits by which shifting shall be effected. After a comma the name of the variable is indicated the content of which is manipulated.

Operation in
DIR - Mode
Key "Dir"

Display of the PHG DIR-Mode



```
50.00
00004  S:/SGM/                                   Antriebe Aus
   DIR "SGM                                      Achse 1-6 *
   ..                 <DIR>          834         Auto
   AUSWERFER           MAC                       DIR 450
>  BEDI_MAK           <DIR>          568
   GREIFBEW_A          MAC          1267
   GREIFER_1           MAC           987      3.03.1
   GRUNDST             MAC          1277     12.03.1
   GRUNDST_ANF         SPR          3799     26.02.1
   GR_BED              MAC           743     25.01.1
   KERNZUG_1           MAC           398     20.03.1
   SGM_AUSWE           MAC           487     17.03.1
   STARTBED            SPR           621     17.03.1
   TYP1                MPR          1489     17.03.1
   TYP2                MPR          1755     18.02.1
```

Status window with indication of the states ; indication of the manual operating mode and program processing
current path
adjusted override value (numerically and as beam)
current step number
Prog öffn
Prog mark
Prog entf
Prog kopi
Prog wahl
Rück gäng
F↑
F1
F2
F3
F4
F5
F6
F↓
change of the menu levels

Description of the indication in the display

The display consists of 20 lines in total.
The first line indicates the set value of the override regulator. This value is indicated as number and as red beam the length of which varies with setting of the regulator.

The second line (blue) contains the number of the step where the cursor is located. Behind the step number the current drive and the directory are indicated the inputs of which are indicated in the lines 3 to 18 . S: is the standard drive of the control (S-RAM), A: means the disk drive.

The lines 19 and 20 contain the selection menus for the function keys.

A directory is a program type. The first step of a directory is named DIR followed by the name, the last step is always named END. The directories form a tree structure. Based on the master directory DIR "/" generation of further sub-directories is possible. The names are freely selectable and may have a length of up to 20 characters. The maximum allowed nesting depth is 6.
The second step of a sub-directory is always named ".." <DIR>. It represents the superimposed directory; i.e. this entry is not available in the master directory.

Programs and directories are listed in alphabetical order in a directory (numbers before letters). The program type is indicated after the name:

MPR	main program
SPR	sub-program
MAC	macro
PAL	pallet program
PLC	PLC - program
<DIR>	sub-directory

The properties of the programs are indicated after the program types:

program length [Byte]
date of generation or last modification
time of generation or last modification
activated program protection (see below)
remark that global definitions exist (see below)

In order to read the rest of a line running out of the window to the right hand side, lateral displacement of the text in the window is possible with the arrow keys on the right and left hand side. Each operation of the key displaces the text by one character, if the key is kept pressed the text is continuously running into the selected direction.

If the key "SHIFT" was operated prior to pressing the arrow key, the text jumps to the right resp. left hand side by 20 characters. With key "ALT" and subsequent operation of the arrow key on the left the display jumps to the beginning of the lines. "ALT" and arrow key on the right places the end of the lines into the middle of the window.

Program protection: The operating possibilities of programs / directories are represented by letters:

L/A/E/S/U

L	The program / directory can be deleted
A	The program can be executed in automatic or test mode
E	Commands can be edited in the program
S	The structure of the program / directory can be modified (commands can be inserted, deleted or invalidated)
U	The program / directory can be renamed.

If one of the letters is replaced by a slash the corresponding operation then is locked.
e.g.: -/A/E/-/-
The program marked in this way can now be executed and edited.

Copying of protected programs remains possible.

Remark concerning global definitions

If variables or constants are defined in a program, "Var" stands after the program protection in the directory.
If the program stands in a sub-directory, then the complete path, i.e. all superimposed directories are marked with „Var“.

Selection of program or directory

Selection of another directory or a program is possible from the directory by setting the cursor on the desired entry. The change is executed by operation of the key "ENTER" . If the cursor stands on step 2 of a sub-directory (".." <DIR>), then change is made into the superimposed directory. If a directory is selected, then the DIR - mode is kept. If jump into a program is made, the mode switches over to "EDIT".

For movement of the cursor in the current directory there are used the keys arrow up and arrow down. The cursor is moved line by line, if the key is kept pressed, the cursor is continuously running up or down.

The combination "SHIFT" and arrow key moves the cursor 15 lines up resp. down. The combination "ALT" arrow key upward sets the cursor on the first line (step 1), the combination "ALT" arrow key downward on the last line of the directory ("END").

Direct jump on a step inside the directory is possible via the key "Step" after entry of a number and operation of the key "ENTER".

Operation in DIR-mode

Selection of the possible operation functions is ensured via the function keys under the display. The keys on the left and right hand side of the function keys are used for switch-over between the menu levels.

<u>First menu</u>

Prog open

Establish new program or sub-directory.
After selection the name of the new program must be entered and be finished with ENTER ; then, the program type is selected with the function keys. After operation of the key ENTER the new program resp. directory is established.

Prog mark

Mark program or directory besides the cursor or cancel its marking.
Marked programs (not directories!) can be copied into another directory or on disk.
Marked programs and directories can be deleted; directories must be empty!

Prog del

Deletion of programs and/or directories. Directories must be empty!
If no marking exists, the following message is displayed:

Acknowledge the program to be deleted:

In the input line below there is indicated the path and name of the program where the cursor stands. By operation of the key ENTER this program is deleted. By input of another name (perhaps with indication of the path) a random other program can be deleted.

If programs / directories are marked, the following message is given:

Do you really want to delete
all marked programs?

After selection of the menu item "YES" with the function key F1 the marked programs are **only deleted in the current directory**

Prog copy

Copying of programs. Directories cannot be copied!
If no marking exists, the following message is given:

Acknowledge the source program:

In the input line below there is indicated the path and name of the program where the cursor stands. By input of another name (perhaps with indication of the path) a random other program can be copied. After operation of the key ENTER the following message is given:

new program name (optionally with
drive and path):

The new program name must be entered. For entry without indication of path the new program is copied into the current directory. After operation of the key ENTER the type of the new program must be selected. Type DIR is not admitted!

If programs are marked, the following message is given:

destination path without program name:

The directory (with complete indication of path) must be entered, into which the marked programs are to be copied.
If the programs are to be stored **on disk,** then only **A:** must be entered! (At the time being it is impossible to copy into subdirectories on the disk!)

Prog sel

Selection of program by input of the program or directory name

If the program to be selected is not in the current directory, the path must be indicated. Directories are indicated by "/" (slash).

If only the slash is entered, direct change to the master directory is made.

Undo

No function yet at the time being.

Second menu

Find repl

Search of a program or directory name **only in the current directory.**
Replacement of character strings in not possible in the DIR-mode!
If programs are marked, the indicated character string is searched for in the marked programs. This character string can be replaced by another one, if it is not component of a program name, a command word, a describer or a system equate.

Disk read

Read in of programs from disk (drive A:).
After selection of the menu item the programs are indicated in the master directory of the disk (no subdirectories!)
The programs to be read in are marked and then copied into the desired directory on the control with "Prog kopi".
If a selected program name already exists in the destination directory, you are asked whether the existing program shall be overwritten.
Overwriting is only possible for programs that are not protected against deletion!

Disk frmt

Formatting of a disk in drive A:
After selection the following message is given:

> Do you really want to
> format disk?

Formatting is started with function key F1 "YES" , with function key F2 "NO" selection is reset.

Mem edit

Not for general use! Only for customer service!

Prog ren

Renaming programs and directories
If no marking exists, the following message is given:

Acknowledge the source program:

In the input line below the path and name of the program / directory on which the cursor stands is indicated. By input of another name (perhaps indication of path) a random other program / directory can be renamed. After operation of the key ENTER the following message is given:

new program name (optionally with
disk and path):

The new program resp. directory name must be entered. For input without indication of path the new program / directory is displaced into the current directory. After operation of the key ENTER the type of the new program must be selected.

If programs / directories are marked, the following message is given:

destination path without program name:

Enter the directory (with complete indication of path) into which the marked programs / directories shall be displaced.

Third menu

Flsh read	Not for general use!
Flsh frmt	Not for general use!

Operation in RUN - Mode Key "Run"

Operation in RUN-Mode

The RUN-mode is activated with the white operating mode key "Run".
The RUN-mode determines in which operating mode program processing will be made (automatic or test mode). Further possibilities of operation are the following ones:

Synchronization of the robot
Approaching reference points
Start of the integrated PLC
Test of the integrated PLC

Selection of the possible operating modes is made with the function keys under the display. The keys on the left and right hand side of the function keys are used for change between the menu levels.

For the TEST-operating modes distinction is made between single step mode and sequential step mode

Single step mode:
Always the current command the cursor points to is executed. When processing of the step is finished, the cursor jumps to the next step, the start key must be released and pressed again, in order to continue processing. If the current step is a programmed position, it will be directly approached in dead man operation, i.e. the movement of the robot stops as soon as the START key is let off. In this manner any position in the program can be directly approached in single step mode after selection of the corresponding step.

Sequential step mode:
a) Set key selector switch on the teach pendant (PHG) to "Auto" or "Auto-Test" (position right or left h.s.):
After operation of the "START" key the program automatically runs to the end and then stops after a return to the beginning of the program. During cycle the program can be stopped with the key "STOP" and started again with the key "START".

b) Set key selector switch on the teach pendant (PHG) to "Cal/Sync" (setting) (medium position)
Also in this case, one step after the other is automatically processed, but programmed positions are approached in dead man operation, i.e., the robot executes movements only so long as the key START will be kept pressed. At the end of the main program the program is also stopped after the return to step 1.

In some TEST-operating modes the input signals must be simulated, i.e., when a command is processed, that has access to a user input, then the function keys F1 and F2 are activated. With F1 simulation of level 0, with F2 simulation of level 1 is possible.

First menu

Test 1

Single step mode macro calls are treated like a system command. (program execution by the interpreter)

Input signals must be simulated. This applies for the commands: WAIT_BIT #EINGANG (INPUT)... and TEST_BIT #EINGANG (INPUT)....
Binary user outputs are not switched
Approximation functions are not active

Test 2

Sequential step mode macro calls are treated like a system command. (program execution by the interpreter)

User inputs and user outputs are treated like in the automatic mode
Approximation functions are active

Test 3

Single step mode macro calls are treated like a system command. (program execution by the interpreter)

User inputs and user outputs are treated like in the automatic mode
Approximation functions are active

Test 4

Sequential step mode macro calls are treated like a system command. (program execution by the interpreter)

Input signals must be simulated. This applies for the commands: WAIT_BIT #EINGANG (INPUT)... and TEST_BIT #EINGANG (INPUT)....
Binary user outputs are not switched
Approximation functions are not active

Test 5

Single step mode; program execution by the executer.
The selected operating mode for the interpreter remains (AUTO; TEST1 to TEST4)
The last line of the status window indicates the mode EXEC. Processing of the current step is made with the key "EXEC". After processing the cursor jumps to the next step. No return to step 1 is executed at the end of a main program. Some commands cannot be executed by the executer:

WAIT_BIT
POSITION
velocities
accelerations
OSCILLATION
WELDING
etc.

Auto

With activated operating mode "AUTO" the current program can be started via the key "START", starting from the current step. The program is processed forward step by step, all valid instructions being executed. During run the program can be stopped at any position with the key "STOP" and restarted with the key "START". When the program has been stopped with the key "STOP", the robot can be moved in operating mode "HAND" after having pressed the key „Quit/Esc". If then the program is started again via "START", it will be continued exactly from that position where interruption took place, i.e. when the robot had been moved, it first of all returns to the position where the movement being just executed had been interrupted and continues the interrupted movement sequence from there. This, however, is no longer valid when the program was scrolled, when a step had been selected via the key "Step", or when the program was modified.

After stopping of the program in operating mode "AUTO" it can be continued in one of the test operating modes and vice versa any time.

Second menu

Cal/ Sync

When the robot became asynchronous (corresponding message in the status window), the menu item "SYNCHRONIZATION" must be selected after selection of this option. The window then faded in shows the asynchronous axes in binary code. This display can be edited; axes that shall be synchronized, are represented with a 1, the other axes with a 0. After having pressed the key ENTER the following message is displayed

synchronization finished

After synchronization the reference points of the synchronized axes must be approached via the function Ref Pos (F3) , and the reference markings must be checked!

The menu item "Cal/Sync" (setting) is used for determination of the resolver offset values. This operation must be made with the first start-up, after replacement of the motor, after replacement of the resolver and after replacement of the gear unit.

Start PLC

Start of the integrated PLC

When a PLC program was written, or when modifications were made in an existing program, then the internal PLC must be started anew. That PLC program is started the name of which is copied in the subprogram

S:/$SYSTEM/$SYSMAC/SYVARINI

into the system variable "_SSTARTPLC". When a new program is to be started, the COPY-command in the program "SYVARINI" must have been executed before!

This program is automatically started with switching on the control.

Ref Pos

Approaching the reference points.

All axes are simultaneously approaching their reference points on the shortest way. The traversing speed is constant, independent from the adjusted override (condition higher than zero). When the last axis has reached its reference position, the following message will be indicated:

reference position reached

Attention! Danger of collision!

Test 6	Single step mode (program execution by the interpreter) Processing is the same as in TEST 1, but macros are also run through in single step mode.
Test PLC	Switching on resp. off the control window in order to check a selected running PLC program. Additionally to the displayed PLC commands, the corresponding interlinkage results and the contents of the accumulators are indicated. Commands that are omitted are marked with "NOP".

Operation in COOR - Mode Key "Coord"

Operation in COOR-Mode

The COOR- mode is activated by the white operating mode key "Coord".
COOR - mode determines the manner of manual movement of the robot. Robot movement generally is made in dead man operation, i.e. the robot only moves so long as the corresponding movement key will be operated. Robot speed can be influenced with the override regulator during movement.

First menu

AXIS 1 - 6	Axes 1 - 6 can be moved in negative or positive direction via the movement keys "1-", "1+" to "6-", "6+". If the keys of several axes are operated at the same time, the corresponding axes are moving simultaneously. During movement the actual values of the individual axes are indicated in a window; for rotation axes in degrees, for linear axes in millimeters
tXYZ rABC	Movement of the tool in the basic coordinate system. Contrary to sub-operating mode „Axis 1 - 6“ the TCP (Tool Center Point) moves on a straight line during cartesian movement and the orientation of the tool remains constant during the movement. The TCP can be moved in the basic coordinate system via the keys X-(1-), X+(1+), Y-(2-), Y+(2+), Z-(3-) and Z+(3+). The orientation angles A, B , and C of the tool can be changed via the keys 4-, 4+, 5-, 5+, 6- and 6+. The TCP keeps its cartesian X, Y, and Z values during the orientation change (presumed that the correct tool data are activated!), i.e., during the robot movements the TCP stands still and only the selected angle changes. During cartesian movement the X-Y-Z-coordinates of the TCP are indicated in mm and the orientation angles A, B, C are indicated in degrees.
tUVW	Movement of the tool in the hand coordinate system. Contrary to sub-operating mode „Axis 1 - 6“ the TCP (Tool Center Point) moves on a straight line during cartesian movement and the orientation of the tool remains constant during the movement. The TCP can be moved in the hand coordinate system via the keys X-(1-), X+(1+), Y-(2-), Y+(2+), Z-(3-) and Z+(3+). During cartesian movement the X-Y-Z-coordinates of the TCP are indicated in mm and the orientation angles A, B, C are indicated in degrees.

tXYZ mouse	Cartesian movement of the TCP with the adapted 6D-mouse. The TCP is not moved via the keyboard of the teach pendant (PHG), but via the connected 6D-mouse.
rABC mouse	Orientation change via the 6D-mouse
AXIS 7 - 12	When the robot is equipped with up to 6 additional axes, those can be moved into negative resp. positive direction via the movement keys "1-", "1+" to "6-", "6+" . The key pair "1" applies for axis 7, the key pair "2" for axis 8 etc. If the keys of several axes are simultaneously operated, the corresponding axes will also move at the same time.

Second menu

ACTI CAL1	activation of the mouse-calibration at the first adaptation point
ACTI CAL2	activation of the mouse-calibration at the second adaptation point
ACTI KPHG	activation of the mouse-calibration at the teach pendant (PHG)
RUN CAL1	calibrating the mouse at the first adaptation point
RUN CAL2	calibrating the mouse at the second adaptation point
RUN KPHG	calibrating the mouse at the teach pendant (PHG)

Operation in INFO - Mode Key "Info"

Operation in INFO-Mode

The INFO- Mode is activated with the white operating mode key "Info".
In INFO - Mode it is possible to call information about the robot, about variable contents and about the state of user inputs, user outputs and PLC markers .

An opened menu window can be closed with "Quit/Esc", or by selecting another menu item.

Selection of the possible operating functions is made with the function keys under the display. The keys on the left and right hand side of the function keys are used for change between the menu levels.

First menu

Achs Istw	Display of the axes' positions of the robot. For rotation axes indication is given in degrees, for linear axes in millimeters.
Kart Istw	Display of the cartesian actual values of the TCP. The current coordinates X, Y and Z are displayed in millimeters and the orientation angles A,B and C in degrees. The displayed data are calculated from the axes positions and the tool data activated last.
Inkr Istw	Indication of the axes' positions of the robot in increments. (resolution of the resolvers: 4096 increments per motor revolution)
Vari able	Indication of contents of global variables. It is also possible to indicate contents of local variables being defined in the current program (only from the EDIT - Mode). After selection of the menu item the name of the variable to be checked must be entered. For integer variables the display format can be selected (decimal; hexadecimal; binary).
Err Enab	Error messages being constantly given by the system and thus blocking the display and operation can be suppressed. Suppression is canceled when the drives are switched on.
Stat Ums	Switching off or on the status window

Second menu

Eing dig	Binary representation of the user inputs. Each input is represented by a bit indicating the current status (zero or one). Counting manner: First line beginning at the right h.s.: Byte 0 Bit 0 to 7; Byte 1 Bit 0 to 7; Second line beginning at the right h.s.: Byte 2 Bit 0 to 7; Byte 3 Bit 0 to 7 etc.
Ausg dig	Binary representation of the user outputs. Each output is represented by a bit indicating the current status (zero or one). Counting manner: First line beginning at the right h.s.: Byte 0 Bit 0 to 7; Byte 1 Bit 0 to 7; Second line beginning at the right h.s.: Byte 2 Bit 0 to 7; Byte 3 Bit 0 to 7 etc.
SPS Merk	Representation of the PLC markers in hexadecimal format (system variable _SPS[n]). In the first line there are the markers _SPS[1] and _SPS[2]; in the second line there are the markers _SPS[3] and _SPS[4] a.s.o.

Operation in FKT - Mode key "Funct"

Operation in FKT - Mode

The FKT - mode is activated with the white operating mode key "Funct".
The following operations are possible:

setting the system clock;
modification of the program protection;
interruption of archival storage and

Selection of the possible operating functions is made via the function keys under the display.

First menu

Dat/ time	Programming the system clock. One after the other the following inputs are expected, the numbers below 10 can be entered with 1 digit. After each input the key "ENTER" must be operated or a comma must be entered. day (1...31) month (1...12) year (98; 99; 00; 01...) hour (0...23) minute (0...59) second (0...59) The last input (second) must be finished with the key "ENTER" . The following message is displayed: real time clock is programmed
DNC rst	initialization of the DNC-interface after a transmission error

Prog prot

Modifying the protection of marked programs.
After selection of this function first the input of a code number is expected. Then, the various protection functions can be activated one after the other. With entering the number 1 the indicated operation is locked, by entering the digit 0 the indicated operation is released. The following operation possibilities can be locked:

loesch.	deletion of programs
ausf.	execution of programs
edit.	editing of program steps
strukt.	modification of the program structure
umben.	renaming of programs

After the last input the following message is displayed:

program protection modified.

If a wrong code number has been entered, or if no programs have been marked, the corresponding message will be given at the end of the input procedure.

Order rst

Abort of an archival storage. The current transmission of a file is finished. All files not yet transmitted are canceled in the order list and archival storage is finished.

Operation in CWIN - mode key "Windows"

Operation in CWIN-mode

The CWIN- mode is activated by the white operating mode key "Window".

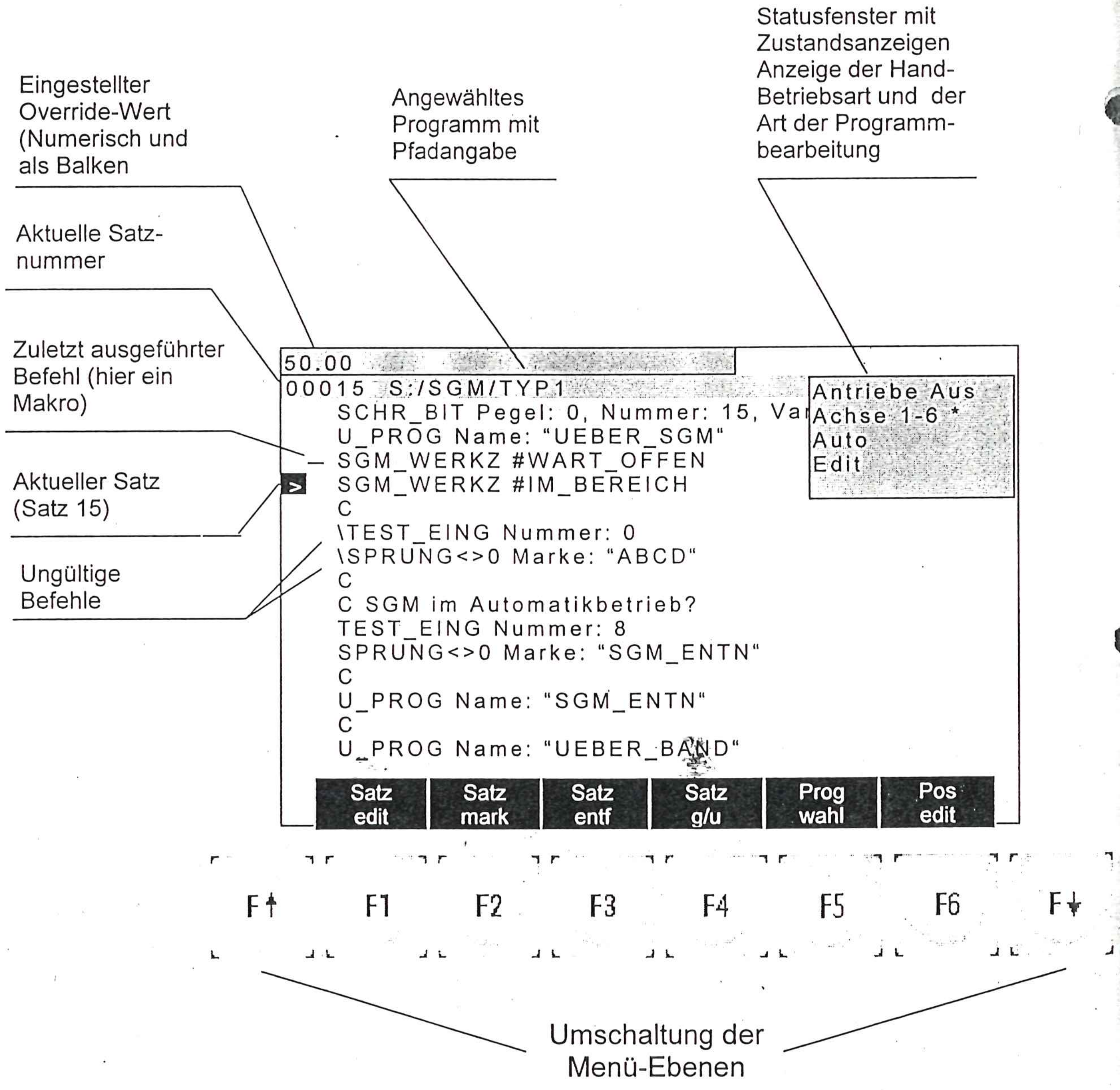
Horizontal or vertical division of the display in two windows is possible. After executed division and operation of the key "Quit/Esc" the cursor can be moved from one window to the other with the key "Switch Win" . While a program is running in automatic mode in one window, another program for instance can be edited in the second window.

First menu

Tlng Hztl	Horizontal division (divi hori) of the display. For further operation the key "Quit/Esc" must be operated.
Tlng Vrtk	Vertical division (divi vert) of the display. For further operation the key "Quit/Esc" must be operated.
Tlng aufh	Cancel existing division (divi undo). For further operation the key "Quit/Esc" must be operated.

Operation in EDIT - Mode program selected, no further key operated

Indication in the display of the PHG EDIT-Mode


Eingestellter Override-Wert (Numerisch und als Balken)
Angewähltes Programm mit Pfadangabe
Statusfenster mit Zustandsanzeigen Anzeige der Hand-Betriebsart und der Art der Programm-bearbeitung
Aktuelle Satz-nummer
Zuletzt ausgeführter Befehl (hier ein Makro)
Aktueller Satz (Satz 15)
Ungültige Befehle
Satz edit
Satz mark
Satz entf
Satz g/u
Prog wahl
Pos edit
F↑
F1
F2
F3
F4
F5
F6
F↓
Umschaltung der Menü-Ebenen

```
50.00
00015  S:/SGM/TYP1
    SCHR_BIT Pegel: 0, Nummer: 15, Va    Antriebe Aus
    U_PROG Name: "UEBER_SGM"             Achse 1-6 *
  — SGM_WERKZ #WART_OFFEN                Auto
>   SGM_WERKZ #IM_BEREICH                Edit
    C
    \TEST_EING Nummer: 0
    \SPRUNG<>0 Marke: "ABCD"
    C
    C SGM im Automatikbetrieb?
    TEST_EING Nummer: 8
    SPRUNG<>0 Marke: "SGM_ENTN"
    C
    U_PROG Name: "SGM_ENTN"
    C
    U_PROG Name: "UEBER_BAND"
```

Description of the indication in the display

The EDIT-mode is activated as soon as a program (no directory) has been selected. The display consists of 20 lines in total.
The first line indicates the set value of the override regulator. This value is indicated as number and as red beam the length of which varies with setting of the regulator.

The second line (blue) contains the number of the step where the cursor is located. Behind the step number the current directory and the name of the selected program are indicated the instructions of which are indicated in the lines 3 to 18 .

The lines 19 and 20 contain the selection menus for the function keys.

The first step of a program always contains a program type followed by the name, the last step is always named "END".

For instructions containing several parameters the display might run out of the window to the right hand side. In order to read the rest of a line, lateral displacement of the text in the window is possible with the arrow keys on the right and left hand side. Each operation of the key displaces the text by one character, if the key is kept pressed the text is continuously running into the selected direction.
If the key "SHIFT" was operated prior to pressing the arrow key, the text jumps to the right resp. left hand side by 20 characters. With key "ALT" and subsequent operation of the arrow key on the left the display jumps to the beginning of the lines. "ALT" and arrow key on the right places the text so far to the left hand side that the last character of the current line is at the right edge of the window (when the normal display consists of more than 37 characters).

For scrolling in the current program there are used the keys arrow up and arrow down. The cursor is moved line by line, if the key is kept pressed, the cursor is continuously running up or down.

The combination "SHIFT" and arrow key moves the cursor 15 lines up resp. down. The combination "ALT" arrow key upward sets the cursor on the first line (step 1), the combination "ALT" arrow key downward on the last line of the program ("END").

Direct jump on a step inside the program is possible via the key "Step" after input of a number and operation of the key "ENTER".

Invalid step

An invalid step is marked with the sign "\" (backslash) before the command word. Invalid means that this instruction is not executed in automatic and test mode.

Instruction executed last

The sign "-" stands on the step that was completely executed as last instruction in automatic or test mode before the program was stopped.

Operation in EDIT-Mode

Selection of the possible operation functions is ensured via the function keys under the display. The keys on the left and right hand side of the function keys are used for switch-over between the menu levels.

First menu

Step edit	Editing the current step in a program . The cursor can be moved in the current step with the arrow key on the right resp. left hand side. The combination "Shift -arrow key" moves the cursor by 20 characters to the right resp. left hand side. The combination "Alt -arrow right" puts the cursor to the end of the line. The combination "Alt -arrow left" puts the cursor to the beginning of the line. The command word and the describers cannot be edited.
Step mark	Marking the current step; various block operations can be executed with marked steps (see below)
Step del	Deletion of the current step from the program. After deletion the cursor jumps to the preceding step. The first and the last step of a program cannot be deleted. **There is no inquiry!** Deletion of a step at the time being cannot be reset!
Step v/i	Validate resp. invalidate current step. An invalid step is marked by the sign "\" (backslash) before the command word. Invalid means that this step is not executed in automatic or test mode.

Prog sel	Program selection by entering the name of the program or the directory When the program to be selected is not located in the current directory, the path must be indicated. Directories are indicated by "/" (slash). If only the slash is entered, direct change into the master directory is executed.
Pos edit	Overwriting of a programmed position by the actual position of the robot. **The position is overwritten <u>immediately</u> and without inquiry!!**

Second menu (block operations)

Blck imp	Insertion of a block from the "intermediate file" (The block stands in the program S:/$SYSTEM/$TEMP/$PASTE.MPR) The block is inserted into the current program at the cursor position
Blck exp	Copy marked block of the current program into the "intermediate file". (program S:/$SYSTEM/$TEMP/$PASTE.MPR is generated resp. overwritten) Only one block must be marked in the current program (one line or several continuous lines without blank).
Blck del	Deletion of all marked steps in the current program **There is no inquiry!** Deletion of a block at the time being cannot be reset!
Blck vali	All marked steps of the current program are validated.
Blck inv	All marked steps of the current program are invalidated . (S. step g/u)
Blck copy	Copy block in the current program. One continuous block must be marked only (one line or several lines without blank). The block is inserted at the cursor position.

Third menu

Find repl	Search of a character string in the current program. Character strings can be replaced by other ones, when they are not part of command words, describers or system equates.
Code word	Input of the code word for definition of the operator surface. Selection of 5 levels with various authorization for access can be selected. See table "authorization for access ROBOTstarV":
Edit mem	Not for general use! Only fur customer service!
Undo	Not yet implemented at the time being.

Access authorization ROBOTstarV

	Customer			REIS	
Level	1	2	3	4	5
access via	without code number	code word 1	code word 2	PIN	TAN
user	operator	setter	programmer	programmer (AEI,BK)	EE
offered menus	none	application-spec. operation macros	all application-spec. system instructions and macros	all system instructions	all open
operation	START / STOP home position with/without robot HNZE	execution of macros reprogramming editing archival storage program and step selection	programming of all user-specific system instructions programming of macros	all open programming of all system instructions programming of macros	all open

ROBOTstarV
Organization of the binary inputs and outputs

The communication between hardware and software takes place via two system variable arrays:

binary inputs:	**_IBIN_IN[10]**
binary outputs:	**_IBIN_OUT[10]**

Both arrays consist of 10 variables with 4 Bytes each.
Every Bit of these variables represents an input or an output respectively.

The inputs and outputs are addressed via Byte- and Bit-numbers. Consequently 40 Bytes can be addressed. The **Bytes** are numbered from **0 to 39.**
The **Bits** number **0 to 7** are available in each Byte.

The following commands serve for direct access to the inputs and outputs:

SCHR_BIT	group of commands Log
TESTE_BIT	group of commands Contr
WARTE_BIT	group of commands Contr

The first parameter to be entered is a system equate:

#AUSGANG	access to a user output
#EINGANG	access to a user input

Addressing is made in two of the following parameters:

Byte:	Byte number of the input or output (0 to 39)
Bit_Nr:	Bit number of the a. m. Byte (0 to 7)

Example: Setting an output:

SCHR_BIT #AUSGANG, level: 1, Byte: 5, Bit_Nr: 3

With this command the output would be set which is represented in the variable _IBIN_OUT[2] in Bit 11.

Organization of the binary inputs and outputs
in system variables
by the example of the user inputs

Variable	Byte - Number			
_IBIN_IN[1]	3	2	1	0
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[2]	7	6	5	4
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[3]	11	10	9	8
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[4]	15	14	13	12
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[5]	19	18	17	16
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[6]	23	22	21	20
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[7]	27	26	25	26
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[8]	31	30	29	28
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[9]	35	34	33	32
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210
_IBIN_IN[10]	39	38	37	36
	Bit 76543210	Bit 76543210	Bit 76543210	Bit 76543210

DOCUMENTATION

REIS GMBH & CO MASCHINENFABRIK OBERNBURG

GENERAL

Title: Path Control

Author:	Wetzel
Date:	01.07.98
Control version:	RSV
Software revision:	01_00

Date:	19.06.98
Release:	Som

TABLE OF CONTENTS

1 Parameters of the path control

1.1 INTRODUCTION

The path control of the ROBOTstarV allows configuration of traversing ways with many parameters. The central traversing parameters such as movement mode, path speed or activation of the approximation mode are adjusted by means of user commands. Other parameters being used less frequently are programmable in system variables.

As a matter of principle, all **operations on system variables and the user commands are self-locking,** thus being valid for all following programmed points or paths. **Exceptions** are just the command **BAHN_ZEIT** (path time) and programming of the orientation weight in the system variable **_ORI_WEIGHT**.

In an initialization file all **path parameters are pre-allocated with values. Working is already possible with this pre-allocation.**
Finally, for recognition of limit cases, e.g. paths which don't have to move any distance, but only a readjustment of orientation, and for the spline mathematics there are further parameters which are stored in the machine data and which must only be changed under instruction.

1.2 ADJUSTMENT OF PARAMETERS BY MEANS OF USER COMMANDS

1.2.1 The interpolation mode

Command:	**BEWEG_ART**	#PTP #LINEAR #SPLINE #ZIRK_MIN_ORI #ZIRK_BAHN_ORI

The type of interpolation is controlled with the command BEWEG_ART.

The movement mode #SPLINE connects programmed points by a path without corners.

The ZIRK movement modes are used for circular interpolation. The system constant #ZIRK_MIN_ORI (RSIV: ZIRK1) indicates that readjustment of orientation is made on the shortest way here. The constant #ZIRK_BAHN_ORI (RSIV: ZIRK2) activates the circular interpolation with orientation readjustment in a coordinate system moving along the circular contour.

Attention: In circular interpolation, the orientation in the auxiliary point is taken into consideration for the interpolation.

1.2.2 The path speed

Command: **BAHN_GESCHW** [mm/s]:<value>

Used with: **all interpolation modes except #PTP**

Definition of the path speed in the modes #LINEAR, #ZIRK_MIN_ORI, #ZIRK_BAHN_ORI and #SPLINE (RSIV: GESCH_CP).

1.2.3 The path acceleration

Command: **BAHN_BESCHL** [%]:<value>

Used with: **all interpolation modes except #PTP**

Definition of the path acceleration in the modes #LINEAR, #ZIRK_MIN_ORI, #ZIRK_BAHN_ORI and #SPLINE. The percentage refers to a maximum acceleration which can be accessed to via the system variable _RACC_MAX. This variable is allocated in the initialization file for the system variables (→chapter 1.5).

1.2.4 The PTP speed

Command: **PTP_GESCHW** [%]:<value>

Used with: **interpolation modes #PTP**

Definition of the speed in #PTP mode. The percentage refers to the max. axes speeds of the robot. (machine data RAXES_NOM_SPEED).

1.2.5 The PTP acceleration

Command: **PTP_BESCHL** [%]:<value>

Used with: **interpolation modes #PTP**

Definition of the acceleration in #PTP mode. The percentage refers to the max. axis acceleration of the robot. (machine data RAXES_NOM_ACCEL)

1.2.6 The traversing time on a CP-path

Command: **BAHN_ZEIT** [s]:<value>

Used with: **all interpolation modes except #PTP**

Definition of the traversing time along a CP-path (modes: #LINEAR, #ZIRK_MIN_ORI, #ZIRK_BAHN_ORI and #SPLINE).

This command is not self-locking and has priority towards a programmed path speed. It **can** be used instead of the path speed command BAHN_GESCHW and it **must** be used if a path has been programmed without distance, but with orientation readjustment; because in this case a path speed is without importance.

1.2.7 Activation of the approximation mode for PTP and CP

Command:	**UEBERSCHL**	#EIN #AUS

There is no distinction between activation of CP-/PTP-approximation. The parameters for configuration of approximation are only considered with activated approximation:

The following description parameters are used:

- BAHN_RADIUS and BAHN_DISTANZ
- CP-speed in the approximation point _IV_FLYBY_S (RSIV: UEBER_CP)
- Orientation speed in the approximation point _IV_FLYBY_O
- ...

With activated approximation, **always** a path approximation is executed. Pure speed approximation **doesn't** exist any more.

The movement mode SPLINE is a particularity. Since the path already doesn't have any corners in this case, a programmed BAHN_RADIUS and/or BAHN_DISTANZ won't be considered.

1.2.8 The path approximation radius

Command:	**BAHN_RADIUS**	[mm]:<value>

Used with: **ÜBERSCHL #EIN in all interpolation modes except #PTP**

Definition of the "path approximation sphere" on a CP-path. (RSIV:RAD_CP). In a distance of BAHN_RADIUS before reaching a programmed position, the robot leaves the programmed path and only enters into the next path in a distance of BAHN_RADIUS after the position. A connection path is moved inbetween.

1.2.9 The path approximation distance

Command: **BAHN_DISTANZ** [mm]:<value>

Used with: **ÜBERSCHL #EIN in all interpolation modes except #PTP**

This parameter defines the distance to the programmed corner point of a CP-path. A **distance programming has priority** towards the radius programming.

The command can be switched off by BAHN_DISTANZ = 0.0, which also corresponds to the pre-allocation (→ chapter 1.5). Then an approximation path is calculated by means of BAHN_RADIUS.

1.3 ADJUSTMENT OF PARAMETERS VIA SYSTEM VARIABLES

1.3.1 Speed approximation factor of the path

Action:	**KOPIERE** source:<value>,destination:**_IV_FLYBY_S**
Explication of name:	**V** stands for speed, **FLYBY** for approximation parameter and **S** for the path.
Used with:	**ÜBERSCHL #EIN in all interpolation modes except #PTP**
Range of values:	0[%] - 100[%]

The system variable _IV_FLYBY_S (RSIV: Command UEBER_CP) contains a percentage factor describing the passage speed in the approximation point. The approximation point is situated in the middle of the approximation path.

$$v_{flyby} = (v_1 + v_2) / 2 * \_IV\_FLYBY\_S / 100;$$

v_1 : path speed of the current path;
v_2 : path speed of the following path;
v_{flyby} : path speed in the approximation point.

1.3.2 Speed approximation factor of the orientation

Action:	**KOPIERE** source:<value>,destination:**_IV_FLYBY_O**
Explication of name:	**V** stands for speed, **FLYBY** for approximation parameters and **O** for the effect to the orientation.
Used with:	**ÜBERSCHL #EIN in all interpolation modes except #PTP**
Range of values:	0[%] - 100[%]

The a.m. copy action effects a speed approximation for the path orientation with approximation command being switched on. The system variable represents a percentage factor describing the orientation speed in the approximation point similar to the effect of _IV_FLYBY_S.

1.3.3 Speed factor of the circular auxiliary point orientation

Action:	**KOPIERE** source:<value>,destination:**_IV_CIRCHP_O**
Explication of name:	**V** stands for speed, **CIRCHP** for circular auxiliary point and **O** for orientation.
Used with:	**BEWEG_ART #ZIRK_MIN_ORI or #ZIRK_BAHN_ORI.**
Range of values:	0[%] - 100[%]

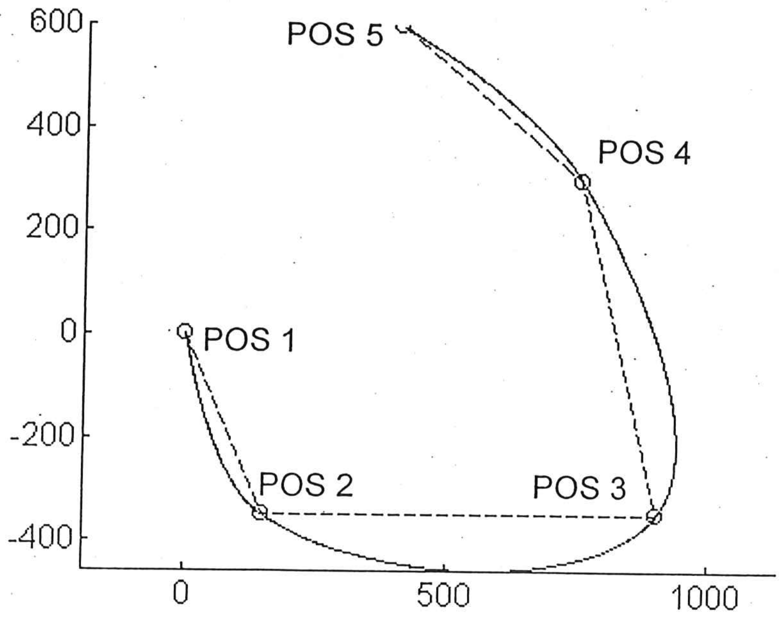
The a. m. copy action effects a speed approximation for the circular auxiliary point orientation. The system variable represents a percentage factor defining the orientation speed in the auxiliary point completely analog to IV_FLYBY_O.

1.3.4 Curvature parameters of the spline interpolation for the course of the position path

Action:	**KOPIERE** source:<value>,destination:**_ISPL_TENS**
Explication of name:	**SPL** stands for spline, **TENS** for tension
Used with:	**BEWEG_ART #SPLINE.**
Range of values:	0[%] - 100[%]

The variable is only used with programmed #SPLINE interpolation mode. With this factor the course of the path between two programmed points can be changed.

The name is explained by the comparison of the path with the curvature of an elastic material. The lower the tension is at a programmed position, the stronger the path approximates the linear path, the bigger will thus be the discontinuity at the position. The higher the tension is, the more the path strikes out, so that the discontinuity at the position becomes smaller (→ example plots).

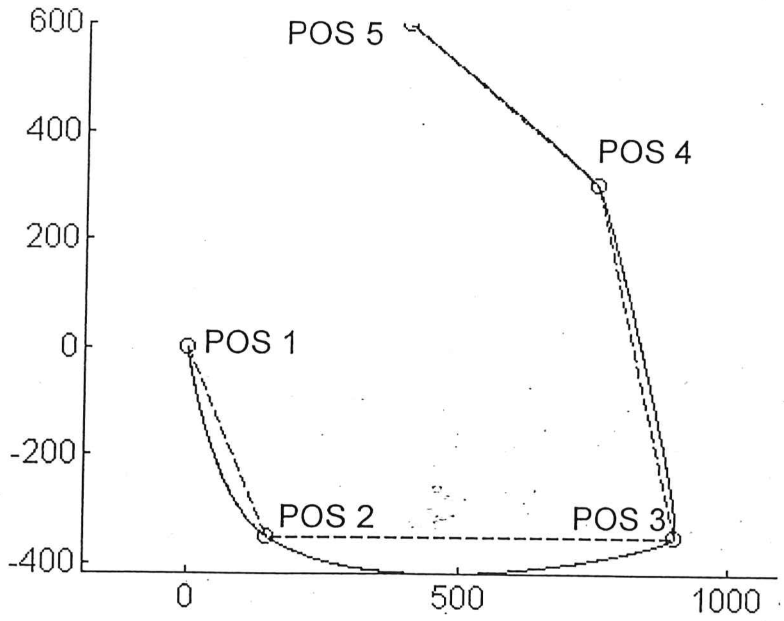


Teach Procedure:

1. POSITION 1

2. BEWEG_ART: #SPLINE

3. POSITION 2
4. POSITION 3
5. POSITION 4
6. POSITION 5



Teach Procedure:

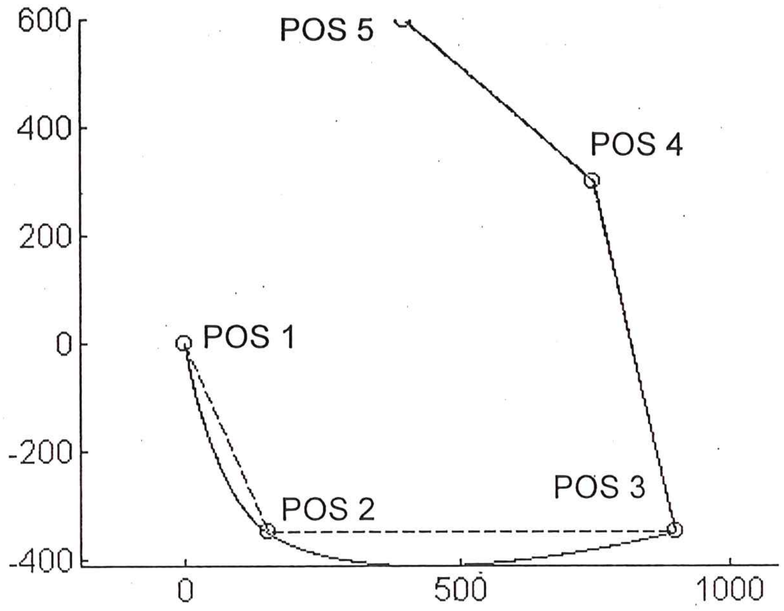
1. POSITION 1

2. BEWEG_ART: #SPLINE

3. POSITION 2

4. _ISPL_TENS = 20
5. POSITION 3
6. POSITION 4

7. _ISPL_TENS = 100
8. POSITION 5



Teach Procedure:

1. POSITION 1

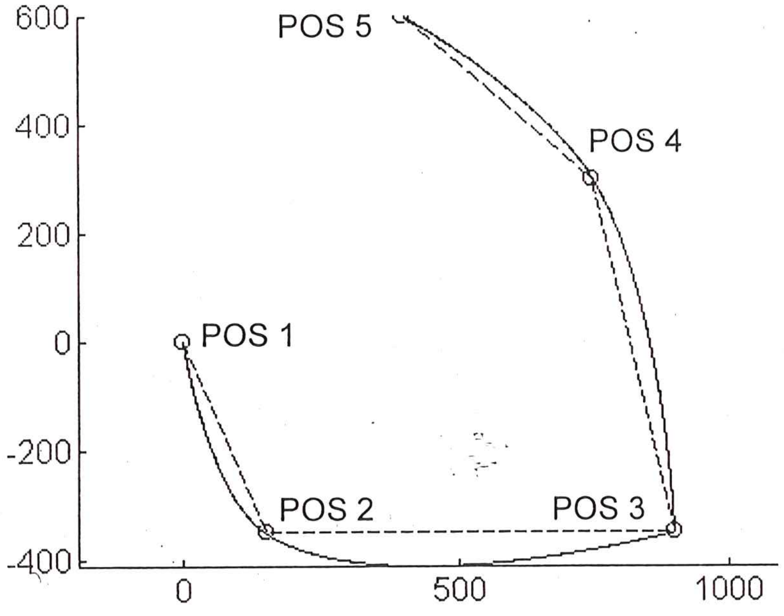
2. BEWEG_ART: #SPLINE

3. POSITION 2

4. _ISPL_TENS = 0

5. POSITION 3
6. POSITION 4

7. _ISPL_TENS = 100
8. POSITION 5



Teach Procedure:

1. POSITION 1

2. BEWEG_ART: #SPLINE

3. POSITION 2

4. _ISPL_TENS = 0
5. POSITION 3

6. _ISPL_TENS = 100
7. POSITION 4
8. POSITION 5

1.3.5 Curvature parameters of the SPLINE interpolation for the course of the orientation path

Action:	**KOPIERE** source:<value>,destination:**_IORI_TENS**
Explication of name:	**ORI** stands for orientation, **TENS** for tension
Used with:	**BEWEG_ART #SPLINE or UEBERSCHL #EIN in the interpolation mode #LINEAR, #ZIRK_MIN_ORI and #ZIRK_BAHN_ORI**
Range of values:	0[%] - 100[%]

The factor indicates the tension of the spline for the course of the orientation. The closer the factor is to zero, the stronger the orientation path approximates the interpolation on the shortest way, the bigger consequently the discontinuity will be at the programmed position.

Since with activated approximation the orientation is interpolated in the spline concept in all CP-interpolation modes, this variable is of great importance in these cases.

1.3.6 Weighting of the orientation

Action:	**KOPIERE** source:<value>,destination:**_IORI_WEIGHT**
Explication of name:	**ORI** stands for orientation, **WEIGHT** for weight.
Used with:	**all interpolation modes except #PTP independent from the approximation action**
Range of values:	0[%] - 100[%]

The factor indicates a measure in how far the orientation of a programmed point shall be considered for interpolation. Accordingly, 100% means that the programmed orientation is passed through.

The smaller the factor is adjusted, the less the programmed orientation will be considered at the corresponding point.

With 0% the programmed orientation won't be taken into account at all. In this case the path control calculates an average orientation by means of the neighbor orientations.

Due to this way of functioning it is **impossible** to allocate a weight _IORI_WEIGHT ≠ 100% to two successive positions. The command is independent from approximation actions and is used for shortening an orientation distance. It is **not self-locking**.

1.3.7 Weighting of the orientation of the circular auxiliary point

Action:	**KOPIERE** source:<value>,destination:**_IORI_CIRCHP**
Explication of name:	**ORI** stands for orientation, **CIRCHP** for circular auxiliary point.
Used with:	**interpolation modes #ZIRK_MIN_ORI and #ZIRK_BAHN_ORI**
Range of values:	0[%] - 100[%]

The factor indicates a measure in how far the orientation of a programmed circular auxiliary point shall be considered for interpolation. Accordingly, 100% means that the programmed orientation is passed through. A programming of 0% leaves the orientation of the auxiliary point disregarded for the circular interpolations.

Like with _IORI_WEIGHT, an average orientation is calculated by means of start point and end point of the circle.

Like most of the parameters, this one is also self-locking for all following circular auxiliary points.

1.3.8 The brake time for programming of a BAHN_ZEIT (path time)

Action:	**KOPIERE** source:<value>,destination:**_IACC_TIME**
Explication of name:	**ACC** stands for acceleration, **TIME** for a period of time.
Used with:	**paths without distance, paths with programmed BAHN_ZEIT (path time)**
Range of values:	0[%] - 50[%]

The percentage factor defines the brake time on a CP-path and refers to the programmed BAHN_ZEIT (path time).

For example, _IACC_TIME = 20% means that 20% of the BAHN_ZEIT (path time) each are used for acceleration and deceleration respectively, and the remaining 60% for the movement with constant speed. With maximum adjustment _IACC_TIME = 50% there is no constant movement phase (triangular profile).

1.4 ADJUSTMENT OF PARAMETERS IN THE MACHINE DATA

The constants for the "spline mathematics" are filed in the machine data. These machine data are pre-adjusted by the system and don't have to be changed by the user. For the sake of completeness they are listed in the following. The following system constants are needed:

1.4.1 RSPOS_APPROX_QUAL / RSORI_APPROX_QUAL

Significance:	positive approximation quality of the reparameterization for position [in mm] and orientation [in degree].
Range of values:	RS..._APPROX_QUAL > 0

Default:	RSPOS_APPROX_QUAL = 1e-6
	RSORI_APPROX_QUAL = 1e-5

1.4.2 RDELTA_OV_MAX

Significance:	max. admissible override change per interpolation cycle.
Range of values:	0 < RDELTA_OV_MAX <= 1.0
Default:	0.1

1.4.3 RAPPR_FACT_MAX

Significance:	restriction of the approximation area relative to the path length.
Range of values:	0 < RAPPR_FACT_MAX < 0.5
Default:	0.4

1.4.4 RSYNC_WEIGHT

Significance:	weight of the reference phase duration for the synchronization of speed profiles:
Range of values:	0.0 ≤ RSYNC_WEIGHT ≤ 1.0
Default:	1.0

1.4.5 RSMOOTH_WEIGHT

Significance:	dimension for the smoothness for reduction of the acceleration for synchronization of speed profiles.
Range of values:	0.0 ≤ RSMOOTH_WEIGHT ≤ 1.0
Default:	0.5

1.4.6 RSYNC_MIN_PHASE

Significance:	minimum phase duration for synchronization of speed profiles proportionate to the total duration.
Range of values:	0.0 ≤ RSYNC_MIN_PHASE ≤ 0.5
Default:	0.1

1.4.7 RTIME_EPS

Significance:	minimum path duration for the detection of path zero steps in sec.
Range of values:	RTIME_EPS > 0.0
Default:	0.001

1.4.8 RS_POS_EPS

Significance:	minimum distance of programmed points in mm for determination of paths without travel distance.
Range of values:	RS_POS_EPS > 0.0
Default:	0.01

1.4.9 RS_ORI_EPS

Significance:	minimum distance of programmed orientations in degrees for determination of paths without orientation way to be traveled.
Range of values:	RS_ORI_EPS > 0.0
Default:	0.1

1.4.10 RS_AX_EPS

Significance:	minimum distance of programmed additional axes positions in degrees for determination of path zero steps in PTP and for additional axis zero steps.
Range of values:	RS_AX_EPS > 0.0
Default:	0.001

1.5 DEFAULT OF THE PARAMETERS

For all path parameters, a pre-allocation of the path parameters is made when the system is started, and that in the initialization file of the system variables S:\$SYSTEM\$SYSMAK\SYVARINI.MPR. In the first instance, working is possible with this default.

Allocation of the system variables to the path parameters is made as follows:

default system variable	*appropriate path parameter*	*value*
_IDEF_MOVMOD	command: BEWEG_ART	0 → #PTP
_IDEF_FLYFLG	command: ÜBERSCHL	0 → #AUS
_RDEF_VEL_CP	command: BAHN_GESCHW	10.0
_IDEF_ACC_CP	command: BAHN_BESCHL	50
_IDEF_VELPTP	command: PTP_GESCHW	20
_IDEF_ACCPTP	command: PTP_BESCHL	50
_RDEF_PATH_R	command: BAHN_RADIUS	1.0
_RDEF_PATH_D	command: BAHN_DISTANZ	0.0
_RDEF_PATH_T	command: BAHN_ZEIT	0.0
_IDEF_VFLY_S	variable: _IV_FLYBY_S	100
_IDEF_VFLY_O	variable: _IV_FLYBY_O	0
_IDEF_VCRC_O	variable: _IV_CIRCHP_O	100
_IDEF_SPLTEN	variable: _ISPL_TENS	100
_IDEF_ORITEN	variable: _IORI_TENS	100
_IDEF_ORIWGT	variable: _IORI_WEIGHT	100
_IDEF_ORICRC	variable: _IORI_CIRCHP	100
_IDEF_ACCTIM	variable: _IACC_TIME	25

2 Guidelines for programming in SPLINE operation for processing of free form surfaces

2.1 INTRODUCTION

Besides the path control modes LINEAR, ZIRK_MIN_ORI and ZIRK_BAHN_ORI, another movement mode is at the programmer's disposal in the RSV-surface: SPLINE. This movement mode connects programmed points with a continuous path without corners. Thus, arbitrary curved path courses can be generated in the best possible manner with a minimum of programmed points.

In addition, the orientations are connected in a similar way without any discontinuity. All this happens without leaving or shortening the programmed positions and orientations.

The **SPLINE interpolation mode** is therefore ideal for the processing of free form surfaces **and should always be used for these application cases.**

Programming of such paths requires special strategies regarding the choice of programming positions. In addition, the path behavior can be adapted to the corresponding requirements by means of many parameters. The default adjustment of the path parameters creates a course of path suitable for many applications. Optimization of same, however, has to be effected by adaptation of these parameters.

The SPLINE movement mode creates at programmed points already smooth paths without discontinuity. Therefore no additional measures need to be taken regarding path approximation. Programming of the user commands **BAHN_RADIUS** and **BAHN_DISTANZ** is consequently **omitted.**

All movement parameters, their significance and default are described in chapter 1.

2.2 PROCEDURE FOR THE SPLINE PROGRAMMING

In the movement mode SPLINE there are many possibilities for modification of the course of the path. From this results a special strategy for generation of a robot program.

Based on the experience with programming of arbitrary curved screens which have to be processed with constant speed, the following guideline has proven to be promising:

1. **Extremely exact setting of positions, especially of orientations.**

2. **Switching on the approximation command.**

3. **Processing the movement in operating mode TEST1 or TEST4** for verification of the position choice, because the robot stops at each position here. Doing so, the path between the programmed positions is of no importance.
   → Adaptation of positions.

4. **Processing the movement in operating mode TEST1 or TEST4** for verification of the course of the path and the course of the orientation between the programmed positions.
   → Adaptation of the path parameters _ISPL_TENS, _IORI_TENS, _IORI_WEIGHT.

5. **Processing the movement in operating mode TEST2, TEST3 or AUTO** at low speed (via override) for verification of the path and orientation speed in the programmed points.
   → Adaptation of the path parameters _IV_FLYBY_S, _IV_FLYBY_O
   and perhaps the machine data RSYNC_WEIGHT.

6. Perhaps increase of the **filter variable _FILTER** for smoothing of nominal values for the axes.

In any case, the **filter** is the **last resort** for smoothing of movements!

The total of programming must be done with filter switched off or with the smallest filter, because otherwise the optimization of parameters is nearly impossible.

2.3 SETTING OF POSITIONS

As a matter of principle, programming should be started with as few positions as possible. For the choice of positions you should particularly take care of curvature changes of the contour to be processed.

Points **must** be set

- in each **curvature change,** e.g. at the outlet from a small radius to a path only slightly curved.
- in each **curvature peak,** e. g. in the middle of a small radius.
- in each orientation change

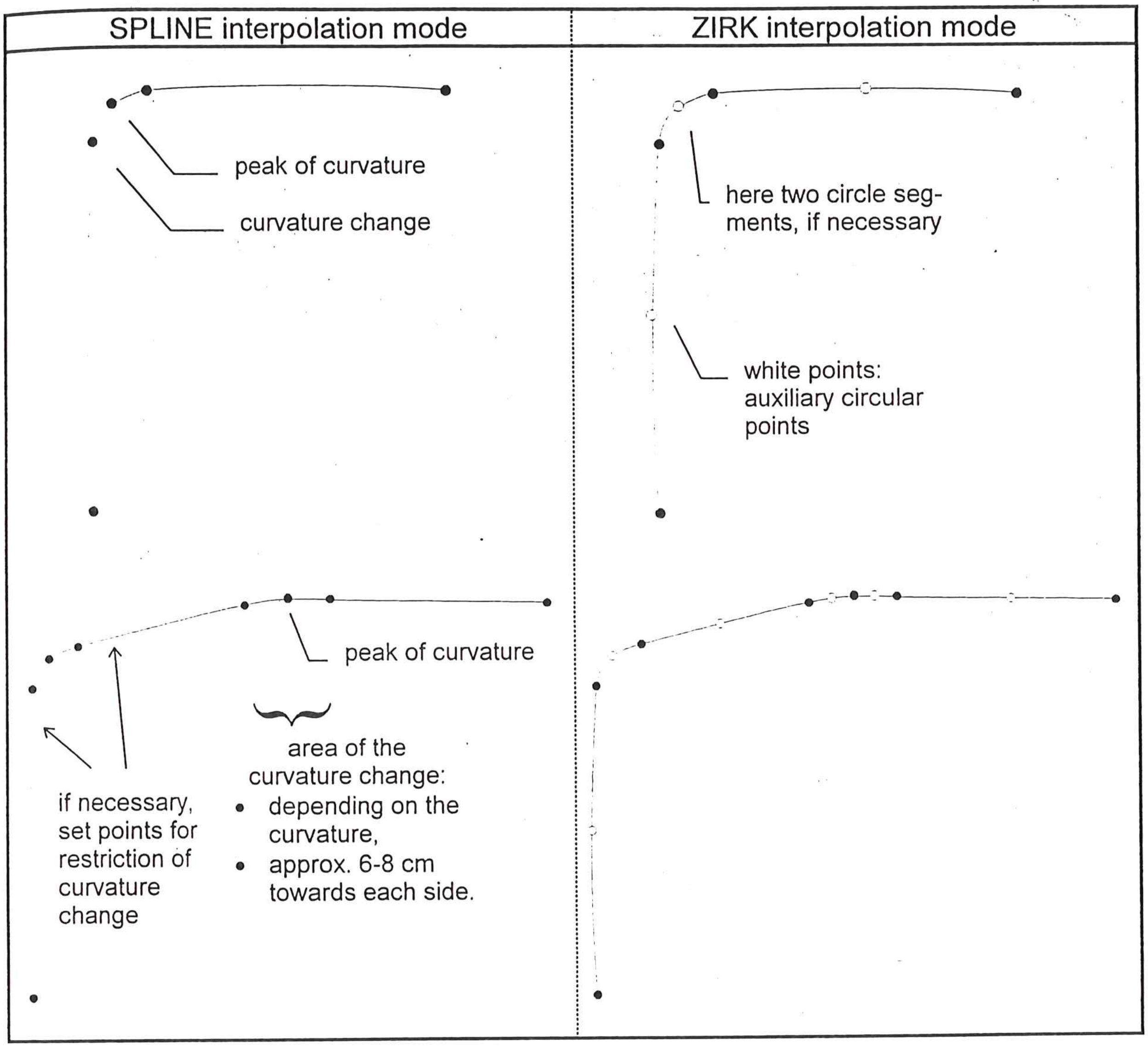
Points **should** be set

- in the **neighborhood of a curvature change,** in order to restrict the effects of the curvature change to a small area.

Since a constantly high speed has to be maintained also in small radii, it has to be observed for adjustment of the orientation that the contribution of the head axes to the robot movement is as small as possible.

The choice of positions is illustrated by characteristic two-dimensional examples. The general three-dimensional case doesn't raise any new problems, since curvature changes and peaks will occur also in this case. In each change of curvature direction a position has to be set here, too.

The conventional method of programming by means of circle segments is compared in order to see the differences. Here a circle end point must be programmed at each curvature.


SPLINE interpolation mode
ZIRK interpolation mode
peak of curvature
curvature change
here two circle segments, if necessary
white points: auxiliary circular points
peak of curvature
area of the curvature change:
• depending on the curvature,
• approx. 6-8 cm towards each side.
if necessary, set points for restriction of curvature change

2.4 OPTIMIZATION OF THE PATH PARAMETERS

In the movement mode SPLINE the **course for path and orientation** can be modified separately. **The speed profile for the course of path and orientation** can also be **modified separately**.

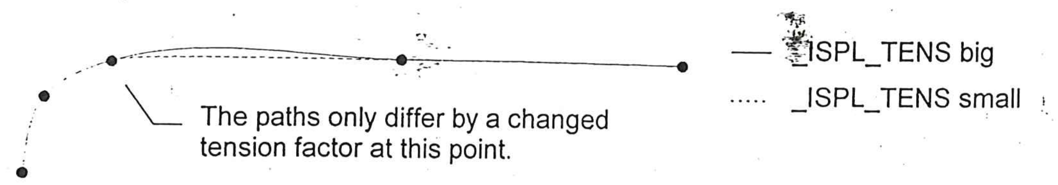
The adjustment of the user commands BAHN_GESCHW and BAHN_BESCHL is immediately clear and won't be mentioned any more in the following.

The **default of the path parameters** generates with **a modification** a course of path which is suitable for many applications: For simulation of the RSIV path behavior the system variable _IV_FLYBY_O was initialized with 0%, for processing of free form surfaces, however, stopping of the orientation in the programmed points is not desirable.

Therefore, either the initialization of _IV_FLYBY_O must be changed by the system variable _IDEF_VFLYO in the system file S:\$SYSTEM\$SYSMAK\SYVARINI or **_IV_FLYBY_O** has to be set **to 100%** at the beginning of the user program.

The following table gives possibilities for solution of some problems.

Problem	Solution
The path **strikes out too far** before/after a position before it swivels into the direction of the previous/following point.	The tension factor for the way is too high, the path must be torn nearer to the straight connection: → Reduce **_ISPL_TENS** before the positions(perhaps to 50%), then reset to 100%.


The paths only differ by a changed tension factor at this point.
— _ISPL_TENS big
····· _ISPL_TENS small

The **robot head strikes out far** between the positions. The orientation path thus moves too far between the positions.	The tension factor for the orientation is too high. → Reduce **_IORI_TENS** before the positions (perhaps to 20% or even 0%), then increase it again. Perhaps a global decrease of _IORI_TENS will be necessary, in this case, however, not to 0%.
The robot approaches orientations which are not really interesting: This problem occurs most frequently in case of **points set tight**. This may reveal itself by a **striking out of the robot head** or by great differences in the orientation speed of two adjacent paths.	Switching off orientations: → **_IORI_WEIGHT = 0** The command is not self-locking, it therefore only has an effect on the following position. Besides this, **two directly adjacent orientations must not be switched off**.
In TEST2, TEST3 or AUTO movement the robot **stops at each position, although approximation** is activated.	The robot shall pass through the programmed points without speed reduction: → **_IV_FLYBY_S = 100.** This corresponds to the default.
In TEST2, TEST3 or AUTO movement the **orientation** stops at each point, **although approximation** is active. In case of points set tight to each other this will result in jerks at those points.	→ Set **_IV_FLYBY_O = 100**.

Despite the a. m. modification, the robot **jerks** in TEST2, TEST3 or AUTO movement at positions where paths with small and big orientation ways follow one after the other.	The jerks result from **too high orientation accelerations.** Deceleration resp. acceleration of the orientation takes place within extremely short time. The length of this phase in the orientation path corresponds to the one of the path speed profile, because the machine data RSYNC_WEIGHT is set to 100%. → Reduce **RSYNC_WEIGHT** to 50 or (if the profiles may be completely independent from each other) to 0.

DOCUMENTATION

REIS GmbH & Co Obernburg

GENERAL

Title: Integrated PLC

Authors	: May, Elter
Date	: 19.11.98
Control version	: RSV
Software revision	: 02_01

RELEASE:

Date:	20.11.98
Signature:	*Som*

TABLE OF CONTENTS

1 General Description of Capacity

1.1 HARDWARE CONDITIONS

The integrated PLC is running on a CPU board together with the controller software. The calculating capacity of the CPU board is divided between the controller software and the integrated PLC. A preponderance for PLC and controller software can be allocated by presetting a power stage for the PLC.

The power stage defines the number of PLC-commands per IPO-cycle. The PLC-interpreter therefore only processes the defined number of commands per cycle.

Default:	20
Machine data:	IPLC_LOAD

A too high value may lead to a system error 1052.

1.2 CAPACITY

- The PLC programs are handled like robot programs, i.e. they are entered, tested and started via the teach pendant.
- Total of commands like Siemens-PLC S5/100 U, but:
  - Without the possibility to transfer parameters to subprograms
  - Different mnemonics, but same functionality
  - No data modules present (not necessary, because there are sufficient markers)
  - The total extent of languages is permitted in subprograms
  - Max. nesting depth for subprograms: 16
  - There are no start-up modules
    (for S5: organization modules OB20 .. OB22)
    Realization is possible via the initialization marker and a corresponding main program structure.
  - No distinction between PB (program module) and FB (function module).
  - Integrated PLC has functionality like FB.
  - Functions "AND" and "OR" have same priority (with S5 "OR" has higher priority).
  - Parentheses have to be set accordingly.
  - Commands for multiplication and division are included in the basic command set
  - No networks programmable (partially can be solved via subprograms)
  - Includes XOR-function
- Cycle time:

The capacity of the PLC depends on the configuration of the robot control, e.g. number of additional axes, CPU calculating capacity, position regulating cycle, duty of the control software (LIN, CIRC, PTP operation) and of the selected PLC power stage.

- The PLC cycle time may be monitored by a watchdog.
  The watchdog time is adjusted in the system variable _ITIME_PLC.
- Due to the increased cycle time, the integrated PLC shouldn't be used for time-critical controls, like for instance path switch-off.
- It has to be observed that the reaction time to state changes at the inputs or in the markers corresponds to the double cycle time.
- The PLC shares the user inputs and outputs with the robot controller.

- In the standard configuration PLC and robot controller can access to the outputs at the same time, the PLC having the higher priority.
- However, the outputs may also be allocated individually, either only to the robot control or only to the PLC.
- The number of inputs/outputs is defined by the number of connected I/O-modules, but max. 40 modules with 8 inputs each and 40 modules with 8 outputs each.
- The inputs/outputs can be addressed individually (bit-addressing)
  I/O 0.1 - I/O 39.7
- The inputs/outputs can be addressed in arrays (Byte-addressing)
  I/O 0 - I/O 39
- There are 2048 marker bytes with 8 bits each. The marker arrays 0 to 511 of those are freely configurable.
- The marker arrays 512 to 2047 (system markers) are reserved for the exchange of system states between robot controller and PLC.
- Access to the marker arrays possible via Byte address or double word address.

Information in the system markers:

- Axes actual values A1 .. A24 for range monitoring of axes
- Status of the system inputs and system outputs (PLC has only reading access)
- POWER-ON marker bit (initialization marker)
- PLC-error bit (division by zero, multiplication overflow)
- etc. (see marker list)

– The marker arrays can be read and written both by the robot controller and the PLC. They serve for quick and direct communication between the robot controller and the PLC.

– The markers can be addressed individually (M 0.0 - M 2047.7) (bit-addressing)
  - or as 8-bit word (M 0 - M 2047) (Byte-addressing)
  - or as 32-bit word (M 0 - M 2044) (double word addressing)

– The markers are maintained after switch-off of the control.

– The timer programming differs from S5; the function, however, is identical.
  Example timer 10 sec:

```
S5:    L   KT 100.1   (unit coded in the 2nd parameter)
       SE T1

Reis:  L   1000       (unit 0.01 sec)
       ST E,T1
```

– The timers are situated in an own data range

– 32 timers are programmable (32 bit-word per timer)

– The time basis for timers is 1/100 sec.

– After each PLC command, a comment can be annexed for better program documentation.

– There are two 32-bit accumulators switched in series.

– After switch-on of the controller the PLC is started automatically. The start program is freely selectable via the system variable _SSTARTPLC and the corresponding path via the system variable _SPLC_PATH.

– Naming of the programs is free (up to 128 characters)

– During program start, the PLC program is checked with regard to plausibility for the recognition of

- Recursive subprogram calls
- Exceeding of the nesting depth
- Incorrect parentheses
- Uncovered subprogram calls
- Uncovered branches

– Faulty programs won't be started.

– In case of inactive PLC-commands the additional information regarding status, VKE, accu1 and accu2 is represented in red color on the teach pendant display.

2 Operating Surface/Programming Unit

2.1 PROGRAM GENERATION

PLC programs can be handled like robot programs. For distinction they have the program identification 'PLC'. PLC programs can be selected, copied and renamed like robot programs.

After generation of a new PLC program, the step for beginning of the program and the end step are displayed:

```
PLC 'program name'
END
```

Attention: **When deleting PLC programs, just the source code of the program is deleted; if the deleted program is being treated in the PLC cycle at the time being, it won't be influenced by deletion!**

2.1.1 Entering commands

Program generation is supported by a menu system. The commands relevant for the PLC are filed in 7 menus with max. 6 commands which can be achieved by means of the PLC key.

Procedure:

Press the key PLC in a HAND sub-operating mode. The first menu with the PLC commands is displayed. The other menus are reached by pressing the cursor key(s).

The command is selected with the function key allocated to the menu item or with the Enter key. Then the parameters are entered with the alphanumeric keyboard of the portable teach pendant.

During the parameter input the syntax check is active. As long as a red bar is displayed in the input line, the syntax is not okay and the command cannot be programmed.

After pressing the key <ENTER> and provided that the input was correct with regard to the syntax, the command is inserted into the actual program after the cursor.

2.1.2 Edition of commands

Commands can be edited like with robot programs.

2.1.3 Deletion of commands

Commands can be deleted like with robot programs.

Since the PLC-interpreter is working with a copy of the PLC-programs, there are all the possibilities available to change the PLC-programs during the runtime. The changes, however, will only become effective after hand-over to the PLC-interpreter by "PLC start".

2.2 START PROGRAM

In order to start a PLC-program or to activate changes, the key RUN must be pressed and the second submenu has to be scrolled to. A menu is displayed where the command "Start PLC" will be selected. If no error occurs during compilation, the changes will be taken over into the cyclic treatment of the PLC.

Remarks on the directory structure in the RSV:
The RSV offers the possibility to store PLC-programs in an own subdirectory or e. g. to keep several versions of one application in own subdirectories.
The following applies as a matter of principle:
Search path + program name are formed of the contents of the system variables _SPLC_PATH and _SSTARTPLC. _SPLC_PATH must and _SSTARTPLC may contain a path indication.
The system variable _SPLC_PATH is predefined with "S:/PLC" and _SSTARTPLC is with blanks.

Examples:	_SPLC_PATH	_SSTARTPLC	started PLC-program
	S:/PLC	MAIN	S:/PLC/MAIN
	S:/PLC	PROJEKT1/MAIN	S:/PLC/PROJEKT1/MAIN
	S:/PLC	PROJEKT2/MAIN	S:/PLC/PROJEKT2/MAIN

A directory change is not possible with subprogram calls (command SU).

2.3 START-UP BEHAVIOR

When switching on the robot controller it is checked whether the name of a PLC-program is entered in the system variable _SSTARTPLC. If so, this program will be searched in the directory from _SPLC_PATH (see above) and if existing, it will be taken over into the cyclic program treatment. Practically this program should always exist in the system, all PLC_modules are called from this start program by corresponding subprogram commands.

_SPLC_PATH should contain an absolute path, because the control selects the main directory after switch-on and the PLC-programs are only started from the current directory if the path indication is missing.

When the control is powered-up the bit 4 in the system marker 1020 is set. With this bit it is now possible e.g. to execute initialization routines. Subsequently the bit must be reset via PLC-command (e. g. R M1020.4)

2.4 ADJUSTMENT OF THE MONITORING TIME

The monitoring time is defined in the system variable _ITIME_PLC. In order to adapt it, the corresponding step in the program SYVARINI must be changed.

The unit of the monitoring time is [1/100 sec.].

2.5 TEST PROGRAM

For testing the PLC programs, it is possible to fade in behind the PLC-command on the teach pendant display the operand status, the linkage result VKE and the contents of the two accumulators. For this purpose the source text of a just running PLC-program must be on the display and the test indication must be switched on. For switching on and off the TEST indication serves the menu item Test PLC from the second submenu under the PHG key Run.

marker M1021.14 reset:	TEST-indication is switched off
marker M1021.14 set:	TEST-indication is switched on

For active commands the additional information regarding status, VKE and accumulator are displayed in black color, for inactive commands in red.

Source text and just running PLC-program must coincide. The control deactivates the TEST-display automatically if there is no coincidence (e. g. after editing). Only after the coincidence has been established again by PLC start, the control will activate the TEST-indication again.

3 Command Set

3.1 OPERANDS

The operands are either bit operands or Byte operands (8 bits) or double word operands (32 bits).

The following operands are admissible for the Reis-PLC:

- inputs (can be addressed individually):	I 0.0 - I 39.7
- input Byte:	I 0 - I 39
- input double word:	I 0 - I 36
- outputs (can be addressed individually):	O 0.0 - O 39.7
- output Byte:	O 0 - O 39
- output double word:	O 0 - O 36

The number of max. admissible inputs and outputs of course also depends on the extension stage of the robot, i. e. on the number of connected CAN-I/O-modules:

- markers (can be addressed individually):	M 0.0 - M 2047.7
- marker Byte:	M 0 - M 2047
- marker double word:	M 0 - M 2044

Timers and counters are situated in a common data range. This data range consists of 32 long words (32 bits per long word). Each long word may be used either as timer or as counter. This requires a certain discipline in programming in order to avoid double assignments.

- timers	T 0 - T 31
- counters	Z 0 - Z 31

When switching off the robot, all marker, timer and counter contents are maintained.

Constants are entered either decimal or hexadecimal. Doing so, it has to be observed that an 'H' is annexed to hexadecimal numbers.

Example: correct: 12ACH or BA12CD33H
wrong: BA12CD33

3.2 LOGIC OPERATIONS

3.2.1 AND-operation/ANDNOT-operation

The following operations link the indicated parameter with the VKE.
The result is stored in the VKE.

command	significance		
U Ax.y	AND output	x=array no.	y=output no.
U Ex.y	AND input	x=array no.	y=input no.
U Mx.y	AND marker	x=marker no.	y=bit no.
U Tx	AND timer	x=marker no.	
U Zx	AND counter	x=marker no.	
UN Ax.y	ANDNOT output	x=array no.	y=output no.
UN Ex.y	ANDNOT input	x=array no.	y=input no.
UN Mx.y	ANDNOT marker	x=marker no.	y=bit no.
UN Tx	ANDNOT timer	x=marker no.	
UN Zx	ANDNOT counter	x=marker no.	
U ACCU	The content of ACCU1 and ACCU2 is linked with AND. (32 bit-operation). The result is stored in ACCU1.		

In order to summarize operations with AND, there is the command **"U ("**.
A closing parenthesis ')' belongs to this command.

3.2.2 OR-operation / ORNOT-operation

The following operations link the indicated parameter with the VKE. The result is stored in the VKE.

command	significance		
O Ax.y	OR output	x=array no.	y=output no.
O Ex.y	OR input	x=array no.	y=input no.
O Mx.y	OR marker	x=marker no.	y=bit no.
O Tx	OR timer	x=marker no.	
O Zx	OR counter	x=marker no.	
ON Ax.y	ORNOT output	x=array no.	y=output no.
ON Ex.y	ORNOT input	x=array no.	y=input no.
ON Mx.y	ORNOT marker	x=marker no.	y=bit no.
ON Tx	ORNOT timer	x=marker no.	
ON Zx	ORNOT counter	x=marker no.	
O ACCU	The content of ACCU1 and ACCU2 is linked with OR. (32 bit-operation) The result is stored in ACCU1.		

In order to summarize operations with OR, there is the command **"O("**.
A closing parenthesis ')' belongs to this command.

Attention: **The OR and the AND operation have the same priority. Therefore, parentheses have to be set in any case. There is no regulation that AND has priority over OR like for other PLC types!**

3.2.3 EXCLUSIVE-OR-operation / EXCLUSIVE-ORNOT-operation

The following operations link the indicated parameter with the VKE.
The result is stored in the VKE.

command	significance		
XO Ax.y	XOR output	x=array no.	y=output no.
XO Ex.y	XOR input	x=array no.	y=input no.
XO Mx.y	XOR marker	x=marker no.	y=bit no.
XO Tx	XOR timer	x=marker no.	
XO Zx	XOR counter	x=marker no.	
XON Ax.y	XORNOT output	x=array no.	y=output no.
XON Ex.y	XORNOT input	x=array no.	y=input no.
XON Mx.y	XORNOT marker	x=marker no.	y=bit no.
XON Tx	XORNOT timer	x=marker no.	
XON Zx	XORNOT counter	x=marker no.	
XO ACCU	The content of ACCU1 and ACCU2 is linked with EXCLUSIVE-OR. (32 bit-operation) The result is stored in ACCU1.		

In order to summarize operations with EXCLUSIVE-OR, there is the command "XO(".
A closing parenthesis ')' belongs to this command.

Due to implementation of parentheses, there is no more limitation with regard to parentheses nesting depth.

3.3 SETTING COMMANDS AND RESETTING COMMANDS

The following commands set the corresponding operand to "1", if the VKE is set.

command	significance		
S Ax.y	set output	x=array no.	y=output no.
S Ex.y	set input	x=array no.	y=input no.
S Mx.y	set marker	x=marker no.	y=bit no.
S Zx	Sets counter no. x to its initial value.		

The following commands set the corresponding operand to "0", if the VKE is set.

command	significance		
R Ax.y	reset output	x=array no.	y=output no.
R Ex.y	reset input	x=array no.	y=input no.
R Mx.y	rest marker	x=marker no.	y=bit no.
R Zx	Sets counter no. x to 0		
R Tx	Resets timer no. x.		

The following commands allocate the content of the VKE to the operand.

command	significance		
= Ax.y	output = VKE	x = array no.	y = output no.
= Ex.y	input = VKE	x = array no.	y = input no.
= Mx.y	marker = VKE	x = marker no.	y = bit no.

3.4 LOADING AND TRANSFER COMMANDS

The following commands load the operand into the ACCU1, the old content of ACCU1 is shifted into ACCU2. ACCU1 is loaded with a Byte and extended to 32 bits without sign.

command	significance	
L Ax	ACCU1 = output Byte	No.x
L Ex	ACCU1 = input Byte	No.x
L Mx	ACCU1 = marker Byte	No.x

ACCU1 is loaded with a double word

command	significance	
LD Ax	ACCU1 = output Byte	No.x to x+3
LD Ex	ACCU1 = input Byte	No.x to x+3
LD Mx	ACCU1 = marker Byte	No.x to x+3

The following commands copy the content of ACCU1 (bit 0 to bit 7) into the respective operand.

command	significance	
T Ax	output Byte	No.x = ACCU1
T Ex	input Byte	No.x = Accu1
T Mx	marker Byte	No.x = Accu1

The following commands copy the content of ACCU1 (bit 0 to bit 31) into the respective operand.

command	significance	
TD Ax	output Byte	No. x to x+3 = ACCU1
TD Ex	input Byte	No. x to x+3 = ACCU1
TD Mx	marker Byte	No. x to x+3 = ACCU1

3.5 TIMERS

The following timer types are implemented:

– pulse	: I
– extended pulse	: V
– with switch-on delay	: E
– with switch-off delay	: A

The timers are started with the start-timer command:

```
- ST <timertyp> , <timer>

e. g.: ST I , T 5
       ST A , T 31
```

The runtime of the timer has to be loaded into ACCU1 beforehand; the indicated number has the unit [1/100 sec.] as a matter of principle.

Functioning of the timers corresponds to the definitions of STEP 5.

3.6 COUNTERS

Also the counters are working according to STEP 5.

The commands

```
ZV <counter> , 1
ZR <counter> , 1
```

serve for counting up and down the counter.

As additional function, also the accumulator can be counted up and down with variable increment resp. decrement:

```
ZV ACCU , <incr>
ZR ACCU , <decr>
```

<incr> and <decr> have to be in the range from 0 to 255.

3.7 COMPARISON COMMANDS

The comparison commands compare ACCU2 with ACCU1 and set the VKE to 1 if the comparison is true. Otherwise the VKE is set to 0.

– equal	: =?
– not equal	: ><
– greater or equal	: >=
– greater	: >
– less or equal	: <=
– less	: <

3.8 ACCUMULATOR COMMANDS

- **Shifting of the accumulator**
  With the commands

  SH R , <no.>
  SH L , <no.>

  the bits in ACCU1 are shifted by <no.> digits to the right or to the left. The digits becoming free in doing so will be filled up with zeros. The parameter <no.> has to be in the range between 0 and 31.

- **Rotation of the accumulator**
  With the commands

  RO R , <no.>
  RO L , <no.>

  the bits in ACCU1 are rotated by <no.> digits to the right or to the left. In this case, <no.> also has to be in the range between 0 and 31.

- **Addition of the accumulators**
  With the command "+" the content of ACCU2 is added to the content of ACCU1; the result is in ACCU1.

- **Subtraction of the accumulators**
  With the command "-" the content of ACCU1 is subtracted from the content of ACCU2; the result is in ACCU1.

- **Multiplication of the accumulators**
  With the command "MU" the content of Accu2 is multiplied by the content of Accu1. The result is in Accu1. If there is an overflow, the marker 1020.5 is set.

- **Division of the accumulators**
  With the command "DI" the content of ACCU2 is divided by the content of ACCU1. The result is stored in ACCU1, the rest of the division is stored in ACCU2.

  In case of overflow or division by zero the marker 1020.5 is set.

- **Formation of complement**
  With the commands

  KE
  KZ

  the complement of one or the complement of two is formed in ACCU1. The result is in ACCU1.

- **Exchange of the accumulator contents**
  For exchange of the contents of ACCU1 and ACCU2 there is the command "TA".

- **Conversion HEX <-> BCD**
  Conversion from HEX to BCD : HD
  Conversion from BCD to HEX : DH

  The conversion takes place for Accu1 respectively.

3.9 BRANCH COMMANDS

In order to execute absolute or VKE-dependent branches in a program, the commands "SP" and "M" are provided as follows:

- SP <destination> : absolute branch to the label <destination>
- SP <destination> , 1 : branch to the label <destination>with VKE = 1
- SP <destination> , 0 : branch to the label <destination> with VKE = 0
- M <destination> : agreement of label <destination>

The names of the destination labels are freely selectable, max. 128 characters are allowed.

3.10 SUBPROGRAM TECHNIQUE

PLC programs can be nested in one another up to a depth of max. 16. Each PLC program may be called by any other PLC program and vice versa. There is no restriction of the extent of language. In order to call a PLC program as subprogram, there are the commands

- SU <prog> : absolute branch into the program <prog>
- SU <prog> , 1 : branch into the program <prog> with VKE = 1
- SU <prog> , 0 : branch into the program <prog> with VKE = 0

At the end of the called program, the calling program is returned to.

All subprograms called with SU must be in the same directory, a directory change is not possible with SU.

3.11 COMMENTS

Comments may be entered in two manners:

- as own command in a line, introduced by ";" or
- after all commands except "SP", "SU" and "M", separated from the command by ";".

4 Communication with the robot control

4.1 SYSTEM VARIABLE _SPS[512]

For the robot programs the system variable _SPS[512] has been defined as an array with 512 elements. This array is identical with the marker bytes 0 to 2047 of the PLC-programs. Thus, it is possible to access to the PLC-markers from the robot program with the commands KOPIERE, SCHR_BIT, WARTE_BIT, TESTE and TESTE_BIT.

Example: KOPIERE xxx,_SPS[54] writes the value xxx into the PLC-marker bytes 212 to 215.

Attention: **In robot programs the array index is counted from 1 to max. In the PLC program counting starts with 0.**
In robot programs all integer variables have a length of 4 bytes (double word). This also applies for array elements and thus, for the system variable _SPS[...].
Therefore _SPS[54] corresponds to the marker bytes M212 to M215.
This conversion is omitted if the commands SCHR_BIT, TESTE_BIT, WARTE_BIT, SUCH_BIT and TESTE are used with the parameter #MERKER. In this case the marker Byte is indicated directly.
Example: SCHR_BIT #MERKER, Pegel:1, Byte-Nr:0, Bit-Nr:5

4.2 SYSTEM MARKERS OF THE PLC

The marker bytes 512 to 2047 are used as system markers to which also the robot controller accesses. These markers must not be arbitrarily used for own programs!

The significance of the individual marker bytes is explained in chapter 7. Summary of the marker allocation.

5 Treatment of Errors

5.1 EDITING ERRORS

Errors occurring during editing are marked by a red bar on the teach pendant display. Such errors are caused e. g. by exceeding the value ranges (Byte number or bit number too high/too low, wrong timer number etc.) or by wrong operand types or by incomplete parameter input.

5.2 TRANSLATION ERRORS

This group of errors occurs after having started the PLC; the running program isn't impaired by this. After elimination of the cause of the error the command for starting the PLC must be executed again.

Messages:

- **"Start program doesn't exist"**
  The system variable _SSTARTPLC doesn't contain a name or the entered program doesn't exist or the entered name is wrong. The system variable _SPLC_PATH contains a wrong path.

- **"Program memory is full"**
  The program memory is so full that compilation can't take place any more.

  Measure: Remove programs from the memory which are no longer needed.

- **"No PLC-command"**
  A robot command has been programmed in the PLC-program, therefore remove this command.

- **"Nesting depth too big"**
  The max. nesting depth for subprograms has been exceeded or there are recursive calls.

- **"Subprogram not found"**
  A subprogram addressed with "SU" doesn't exist under the addressed name.

- **"Incorrect parentheses"**
  Parentheses are missing or parentheses are logically incorrect.

- **"Branch destination not defined"**
  The corresponding M-command to a branch destination in a SP-command cannot be found.

- **"Multiple definition of label"**
  A used branch label is not clear, i. e. it has been used at least twice.

5.3 RUNTIME ERRORS

Runtime errors always cause the PLC to go into the stop status, the drives to be switched off and the outputs to be reset.

All runtime errors have the form: PLC error 700x.

The numbers have the following significance:

- **7000: Time error**
  Time monitoring has responded. Either an endless loop has been programmed or the adjusted monitoring time is too short for the size of the program.

- **7001: Ram overflow**
  After compilation it is tried to reserve protected memory for the PLC. If this isn't successful, the PLC has to be stopped.
  Measure: Remove programs from the memory which are no longer needed.

- **7003: Incorrect function code**
  The PLC interpreter finds a function code (command), which is not defined.
  Measure: Start program anew.

- **7004: Incorrect start identification**
  The next command to be executed by the PLC is destroyed.
  Measure: Start program anew.

5.4 USER-DEFINED CLEAR TEXT MESSAGES

The program PLCMSG_CNF in the directory S:/PLC serves for the output of messages and errors in clear text. This program must contain the texts as well as the bit-coded reaction of the RSV-controller. The following reactions are possible:

Bit number	Reaction
0 - 3 = 0	Message output, the robot is not stopped
0 - 3 = 3	Error message, the robot is stopped
0 - 3 = 7	Error message with switch-off of the drives
8 = 0	This message is always displayed
8 = 1	This message is only displayed with drives being switched on

If bit 0 - 3 = 0 the message is displayed in a green window (operator guidance). The activity of the robot thus isn't influenced. If bit 0 - 3 <> 0 the message is displayed in a red window (error message) and the program treatment will be stopped with or without switch-off of the drives.

The texts may contain max. four parameters and may consist of max. 256 characters per text line. The first parameter is taken from the marker double word M952, the second one from M956, the third one from M960 and the fourth parameter is taken from M964.

Possible control characters: \n for return/linefeed
\d for display of a parameter as decimal number.
\h for display of a parameter as hexadecimal number.
\b for display of a parameter as binary number.

Format of a text line: Mnemonic text number reaction "text with max. 256 characters"

The program PLCMSG_CNF is already prepared in such a way that only the texts and the reaction of the control need to be entered.

The communication between integrated PLC and RSV control for output of freely defined clear text messages is running over the marker double word M948. Via the highest value Bit #31 in this marker a distinction is made whether a freely defined clear text message (Bit #31 set) or a standard message PLC error <parameter> (Bit #31 reset) shall be displayed.

For output of a clear text message the Bits 0 ... 30 must contain the text number of the corresponding I-step from the program PLCMSG_CNF and Bit 31 must be set. For output of the standard message the Bits 0 ... 30 must contain the parameter and Bit 31 must be reset.

Since the PLC markers may also be described by the robot program, it is now possible to initiate error messages also from the user program.

Attention:
At each PLC cycle end (END step in the PLC main program reached) the PLC interpreter checks the content of the marker double word M948 and triggers the message output, if the content of M948 is not equal to zero. Then the PLC interpreter deletes the marker M948 in order to prevent a cyclic output; otherwise the message cannot be acknowledged.

Nevertheless, the programmer must also pay attention that the marker M948 won't be described cyclically in the PLC program, otherwise the message cannot be acknowledged, either.

The marker M1021.1 indicates to the PLC that the message had been acknowledged.

6 System variables and machine data

6.1 SYSTEM VARIABLES

_SSTARTPLC	default:	16 blanks

Contains the name of the PLC program which shall be taken over into the cyclic program treatment automatically after switch-on of the control or with the command Start PLC (see chapter 2.2 and 2.3).

The program name in _SSTARTPLC is invalid if the first character is a blank. After the program name there must be a blank (string end identification).
A path indication prefixed to the program name is allowed.

_SPLC_PATH	default:	S:/PLC

Contains the path for the start program. The complete path with program name results from _SPLC_PATH + _SSTARTPLC.

_ITIME_PLC	default:	30 (0,3 s)

Contains the max. permitted cycle time in the unit 1/100 s for the active PLC program (watchdog).

6.2 MACHINE DATA

IPLC_LOAD	format:	4 Bytes integer
	default:	20
	range of values:	0 to xx

Contains the number of steps being treated by the PLC-interpreter per IPO-cycle.
Example: runtime for a PLC-program with 100 commands with an IPO-cycle = 10ms and
IPLC_LOAD = 20 : 50ms

At JSPS_TIME = 0 the PLC is deactivated, i. e. the robot control contains the total calculating time.

This value is adjusted only once during start-up of the robot control and must only be changed after consultation with Reis company.

7 Appendix: Summary of the System Marker Allocation

The system markers are in the range M 512 to M 2047. The following markers are occupied at the time being:

MARKER	Function	described by:
512 - 899	reserved	
900 - 903	oscillation frequency in Hz/100	PLC / robot
904 - 907	oscillation amplitude in mm/100	PLC / robot
908 - 911	waiting time until arc on in 0,01 sec	robot
912 - 915	waiting time after arc on (preheating time in 0,01 sec.)	robot
916 - 919	error response time during welding in 0,01 sec.	robot
920 - 923	waiting time until arc off in 0,01 sec	robot
924 - 939	reserved	
940 - 943	vision system data	
944 - 947	error number of the ROBOTstar V	robot
948 - 951	Error code for the output of the user defined PLC-errors. Bit #31 reserved for distinction between freely defined clear text message or standard message "PLC-error".	PLC
952 - 955	parameter 1 for clear text message	PLC
956 - 959	parameter 2 for clear text message	PLC
960 - 963	parameter 3 for clear text message	PLC
964 - 967	parameter 4 for clear text message	PLC
968.0	request for a program change to the program from the system variable _SNMIPRG	PLC
968.1	Auto Start if status changes from 0 to 1 Active only in AUTO	PLC
968.2	Auto Stop if status is 1. Active only in AUTO	PLC
968.3	activate lamp test with marker = 1	PLC
968.4 ... 968.7	reserved	
969.0	welding on = 1 / off = 0	robot
969.1	behavior with welding off 0 = default, 1 = stop path until source off	robot
969.2	source working, arc on = 1	PLC
969.3	preliminary sensor correction active	robot
969.4.... 971.7	reserved	

972 - 987	reserved	
988 - 991	system date	robot
992 - 995	system time	robot
996 - 999	input (Bit 0 to 15), level (Bit 16 to 23 and status (Bit 31) if an input inquiry is active in the robot program.	robot
1000 - 1003	system inputs (_ISYS_IN[2])	robot
1004 - 1007	system outputs (ISYS_OUT[2])	robot
1008 - 1011	system inputs (PCIN/_ISYS_IN[1])	robot
1012 - 1015	system outputs (PCOUT/_ISYS_OUT[1])	robot
1020.3	stops the robot movement as long as Bit = 1, no interpreter abortion! Not implemented	PLC
1020.4	Power-On Bit	robot
1020.5	error marker for the commands MU and DI	PLC
1020.6	robot-synchronous marker	robot
1020.7	stops robot movement as long as Bit = 1 (not Auto Stop)	PLC
1021.0	control in error status marker 944 - 947 contains error code	robot
1021.1	The error code entered in the marker double word 944 and also 948 was displayed and acknowledged	robot
1021.2	System initializations after Power-On are ready.	robot
1021.3	plus key pressed	robot
1021.4	minus key pressed	robot
1021.5	robot moves	PLC
1021.6.	PLC-test display is switched on	PLC / robot
1022 - 1023	operating mode	robot
1024 - 1119	actual values of axes 1 - 24	robot
1120.0	content of markers 1124...1135 is valid	robot
1122 - 1123	station number of the current station with table coordinate system	robot
1124 -	cartesian TCP-coordinates X, Y, Z in the current	robot

1135	coordinate system in 0.01 mm	
1136 - 1147	orientation angles A, B, C of the cartesian position in 0.01 degrees	robot
1148 - 1243	preset of the nominal position with movement of axes by the markers 1440...1447 preset in increments not implemented	PLC
1244.- 1339	preset of the acceleration with movement of axes by the markers 1440...1447 preset in % (1..100) of the nominal acceleration not implemented	PLC
1340 - 1435	Override for robot axes with external hand movement. The movement speed is multiplied again by this override value. Unit: % Range of values: +1.0 to +100.0 in DW16 not implemented	PLC
1436 - 1439	Override of 0% ... 100% by which the system override is multiplied. Only in AUTO, TEST2 and TEST4 operation active. Thus, a sequence program can be run with reduced speed, controlled by the internal PLC.	PLC
1440 - 1447	Markers for control of externally movable axes Marker 1440 ...1443 positive marker 1444...1447 negative not implemented	PLC
1448 - 1451	Markers for switching-over "speed or position control" (bit-coded for each axis) with movement through marker 1440...1447 0 -> speed control 1 -> position control not implemented	PLC
1452 - 1455	Bit-coded "axis is in position" with positioning control through marker 1440...1447 not implemented	robot
1456 - 1459	Station switch-over for the multi axes' transformation 1456.0 -> Station 1 1456.1 -> Station 2 etc. 1 means active 0 means inactive The control via PLC must be activated in JGRUP-TAB not implemented	PLC
1460 - 1463	Station switch-over for freeing of station axes. Significance like 1456.x The control via PLC must be activated in JSTMASKE not implemented	PLC
1464 - 1467	length of the current path until now in 1/100 mm, copy of _RPATH_DIST[1]	robot
1468 -	residual length of the current path in 1/100 mm,	robot

1471	copy of _RPATH_DIST[2]	
1471 - 1475	duration of the current path until now in 1/100 sec, copy of _RTIME_DIST[1].	robot
1476 - 1479	residual duration of the current path in 1/100 sec, copy of _RTIME_DIST[2].	robot
1480 - 1871	reserved	
1871.7	Toggle-Bit for the handshake with an external PLC	PLC
1872 - 1919	Markers containing the analog input values. marker allocation M1872...M1873 = analog input 1 M1874...M1875 = analog input 2 : : M1918...M1919 = analog input 24 Format: 16-Bit-A/D-converter data with sign bit. Conversion: marker value * 10.0[Volt] / (2**15 - 1)	PLC / robot
1920 - 1967	Markers containing the analog output values. marker allocation: M1920...M1921 = analog output 1 M1922...M1923 = analog output 2 : : M1966...M1967 = analog output 24 Format: 16-Bit-A/D-converter data with sign bit. Conversion: marker value * 10.0[Volt] / (2**15 - 1)	PLC / robot

1968 - 2007	Markers containing the binary inputs
	
	M1968 = input 0... 7
	M1969 = input 8... 15
	M1970 = input 16... 23
	M1971 = input 24... 31
	M1972 = input 32... 39
	:
	:
	M2007 = input 312...319
		robot

2008 - 2047	Markers containing the binary outputs
	
	M2008 = output 0... 7
	M2009 = output 8... 15
	M2010 = output 16... 23
	M2011 = output 24... 31
	M2012 = output 32... 39
	:
	:
	M2047 = output 312...319
		robot (PLC?)

The following operating modes are possible:

Main operating mode	Sub-operating mode	Code
HAND	HAND/AXIS	0101
	HAND/CART	0102
	HAND/ORI	0103
TEST	TEST_1	0201
	TEST_2	0202
	TEST_3	0203
	TEST_4	0204
AUTO	AUTO	0300
SPEC MODE	HOME_POS	0801

DOCUMENTATION

REIS GMBH & CO MASCHINENFABRIK OBERNBURG

Operating instructions

Title: 6D-Mouse: Calibration and traversing

Author:	May
Date:	13.11.1998
Control version:	RSV
Software revision:	1.6

Date: 01.12.98
Release: *Elter*

1 Task description

With the 6D-mouse an intuitive movement of the robot should be possible. With the movement of the robot with the keyboard of the teach pendant (PHG) you can only drive or rotate along the axes of the coordinate system just adjusted. In this case you need to know the directions of the three axes in the corresponding system of coordinates. It makes it more difficult, that by changing of the site of the server a permanent "rethinking" concerning the axes of the coordinates is necessary.

When using the method with the 6D-mouse this restriction does not exist any more. Now the robot can be "pressed" or "pulled" in any direction you like. So the approach of positions by hand is more facilitated.

The 6D-mouse, however, only offers an alternative to moving of the robot by PHG-keys. **Nothing changed when entering space points in a user program, therefore contours are not stored.**

2 Operating instructions

2.1 Conditions

The functionality for calibration of the 6D-mouse and the movement with the 6D-mouse is already stored in the RSV-system software.
The adaptions only refer to the fixing of the 6D-mouse fixture on the tool and on the PHG. Also the control PHG must be fitted with a 6D-mouse interface which is on the PHG reverse in the form of a socket with 5 poles.

2.2 Calibration of the 6D-Mouse

2.2.1 System programs for calibration

For calibration of the 6D-Mouse there already exist the three programs „MOUSE_KAL1“, „MOUSE_KAL2“ and „MOUSE_PHG“. These programs are in the list S:/$SYSTEM/$DATA and can be chosen with the PHG-Key „Coord“. The menu points „Run Kal1“, „Run Kal2“ and „Run K_PHG“ which are in the second menu are for this purpose.

Additionally the subdirectory S:/$SYSTEM/$DATA contains the program „MOUSE_DAT“ with the three global variables „V_KAL1[6]“, „V_KAL2[6]“ and „V_KALPHG[6]. These variables contain the 6D-mouse-data read in of the calibration carried out last with the following allocation:

V_KAL...[1]	X-Robot	V_KAL...[1].x contains coordinate 1 of the 6D-mouse V_KAL...[1].y contains coordinate 2 of the 6D-mouse V_KAL...[1].z contains coordinate 3 of the 6D-mouse
V_KAL...[2]	Y-Robot	V_KAL...[2].x contains coordinate 1 of the 6D-mouse V_KAL...[2].y contains coordinate 2 of the 6D-mouse V_KAL...[2].z contains coordinate 3 of the 6D-mouse
V_KAL...[3]	Z-Robot	V_KAL...[3].x contains coordinate 1 of the 6D-mouse V_KAL...[3].y contains coordinate 2 of the 6D-mouse V_KAL...[3].z contains coordinate 3 of the 6D-mouse
V_KAL...[4]	Rotation around hand-X	V_KAL...[4].a contains coordinate 4 of the 6D-mouse V_KAL...[4].b contains coordinate 5 of the 6D-mouse V_KAL...[4].c contains coordinate 6 of the 6D-mouse
V_KAL...[5]	Rotation around hand-Y	V_KAL...[5].a contains coordinate 4 of the 6D-mouse V_KAL...[5].b contains coordinate 5 of the 6D-mouse V_KAL...[5].c contains coordinate 6 of the 6D-mouse
V_KAL...[6]	Rotation	V_KAL...[6].a contains coordinate 4 of the 6D-mouse

around	V_KAL...[6].b contains coordinate 5 of the 6D-mouse
hand-Z	V_KAL...[6].c contains coordinate 6 of the 6D-mouse

2.2.2 Calibration at the tool

For calibration of the 6D-mouse on tools there are two programs „MOUSE _KAL1" and „MOUSE_KAL2". The program „MOUSE_KAL2" has the same structure as „MAUS_KAL1" and is only then necessary, if a second 6D-mouse-fixture is fixed on the tool.

„MOUSE_KAL1" describes the variable „V_KAL1[6]" and „MOUSE_KAL2" describes the variable „V_KAL2[6]". Thus two different tool calibrations can be stored and activated if required (fixing of the mouse to fixture 1 or fixture 2).

The calibration programs are structured in this way that starting from the actual robot position one after the other a movement to hand-X-direction, to hand-Y-direction, to hand-Z-direction and a rotation around the hand-X-axis, around the hand-Y-axis and around the hand-Z-axis is carried out. The TCP covers with the linear movement a way of -15mm and with the rotations an angle change of -5 degree is passed.

Sequence:

- Adjust at the PHG with the key selector switch the operating mode AUTO-TEST.
- Select the calibrating program. (PHG-key „Coord" and selection of the corresponding calibration with „Run Kal1" or „Run Kal2" in the 2nd function key menu)
- Preset the operating mode TEST2 (PHG-key „Run").
- Fix the 6D-mouse on the tool.
- Adjust the override to about 30%.
- Switch on the drives with the permission key.
- Start the calibration program (press PHG-key START).
- Detain the 6D-mouse so long until the first movement is over. At the 6D-mouse an excursion must arise opposite to the movement direction. Attention: The data of the 6D-mouse are read in only after the end of the movement, therefore don´t let go the 6D-mouse immediately.
- Bring back the 6D-mouse to zero position (let go shortly and detain again at once).
- Repeat the last two points for each direction of the movement.

2.2.3 Calibration at the PHG

The program "MAUS_PHG" serves for the calibration of the 6D-mouse at the PHG. This program describes the variable „V_KALPHG[6]" and is structured in this way that starting from the actual robot position one after the other a movement to world-X-direction, to world-Y-direction, to world-Z-direction and a rotation around the hand-X-axis, around the hand-Y-axis and around the hand-Z-axis is carried out. The TCP covers with linear movements a way of -50 mm and with the rotations a modification of angle of -10 degree is given.

Sequence:

- Adjust with the key selector switch the operating mode AUTO-TEST on the PHG. If you consider the safety prescriptions concerning the protection of persons the operating mode AUTO is also possible.
- Select the calibration program. (PHG-key „Coord“ and selection of the corresponding calibration with „Run K_PHG“ in the 2nd function key menu)
- Preset the operation mode TEST2 or AUTO (see first point/PHG-key „Run“).
- Fix the 6D-mouse on the tool.
- Adjust the override to about 30%.
- Switch on the drives with the permission key or with the key „Antriebe ein“ (drives on).
- Start the calibration program (press PHG-key START).
- No longer twist the PHG and follow-up with the 6D-mouse the movement of the TCP, that means turn the 6D-mouse in that direction in which the TCP moves. Attention: The data of the 6D-mouse are read in only after the end of the movement, therefore do not let go the 6D-mouse at once.
- Bring back the 6D-mouse into zero position while the robot drives back into initial position (let go shortly and detain again).
- Repeat the last two points for each direction of the movement.

2.3 Activation of the calibration

As already said, the results of the calibrations are in the variables „V_KAL1[6]“ „V_KAL2[6]“ and „V_KALPHG[6]“. After having started a calibration program the corresponding calibration is activated automatically. As long as the contents of the vector variables are not destroyed it is enough to recall simply the calibration data once found out with the help of the PHG-key „Coord“ and one of the menu items „Aktiv Kal1“, „Aktiv Kal2“ or „Aktiv K_PHG“ in the second function key menu.

To each of these menu items belongs a macro which is carried out after having operated the corresponding key.

Menu item	Macro
Aktiv Kal1	S:/$SYSTEM/$SYSMAK/AKTI_KAL1
Aktiv Kal2	S:/$SYSTEM/$SYSMAK/AKTI_KAL2
Aktiv K_PHG	S:/$SYSTEM/$SYSMAK/AKTI_KPHG

In these programs the system command „MAUS_KAL IB“ is carried out and the Bit 1 in the system variable „_VERFAHR[3]“ is set or is deleted.

The system command „MAUS_KALIB V...“ calculates from the data of the vector variable V... a transformation matrix for a translationary and for a rotary movement and stored these data in the system variables „_RMOUSE_T“ and „_RMOUSE_R“.

The Bit 1 in „_VERFAHR[3]“ determines, whether with the procedure the 6D-mouse-data should be additionally transformed into the hand system of coordinates. This is necessary only, when the 6D-mouse is fixed on the tool and the system of coordinates must be followed up.

Bit 1 = 0: Transformation of 6D-mouse to world
Bit 1 = 1: Transformation of 6D-mouse to hand

2.4 Data back-up

As long as nothing is changed on the 6D-mouse fixtures and as long as the content of the user program memory is not disturbed, a single calibration is enough for each available 6D-mouse-fixture. By the buffering of the user program memory the calibration data are not lost even if you switch off the control.

Nevertheless it is recommended to store the program "S:/$SYSTEM/$DATA/MAUS_DAT“ on disk. After a data loss it is possible in a simple way to restore the first calibration stand by reading in this program.

2.5 Movement with the 6D-mouse

The robot is to be handled with the 6D-mouse only in the movement modes „T_XYZ“ (translationary) and „R_ABC“ (rotatory).

An unwillingly change of the orientation angles or the TCP-coordinates, while movement is made with the 6D-mouse, can be prevented by blocking of the corresponding movement mode (translationary or rotatory). The two menu points „T_XYZ mouse“ and „R_ABC mouse“ underneath the PHG-key „Coord“ are meant for this.

Otherwise the same conditions are valid like for the procedure by PHG-keys:
- The robot must be synchronous.
- The drives must be switched on.
- The movement mode „T_XYZ“ and/or „R_ABC“ must be chosen.

The robot is moved by simple excursion of the 6D-mouse. With the right calibration the robot moves in this direction in which it is guided. Hereby the speed of the robot is proportional to the amplitude and reaches with maximal excursion of the 6D-mouse and with an override of 100% the maximal possible hand movement speed of _RVEL_KART and _RVEL_ORI.

3 Error messages

„Calibration not possible“

This error message shows that for at least one movement direction no data are read in by the 6D-mouse. At least one array element of the vector variables mentioned in the command „MAUS_KALIB“ contains only zero-values.

Possible causes:

- The 6D-mouse interface is defective
- The 6D-mouse was not guided during the calibration
- The 6D-mouse was let go too early during the calibration

4 Machine Data

IMOUSE_MIN

Threshold value for the reading in of the 6D-mouse-data. Data which are smaller than IMOUSE_MIN (amount) are replaced by zero.

Value = 0 if -IMOUSE_MIN < 6D-mouse-coordinate < +IMOUSE_MIN

Default: IMOUSE_MIN = 30

IMOUSE_MAX[1] to IMOUSE_MAX[6]

Maximal values for the standardization of the 6D-mouse-data to the maximal hand movement speed. The resulting speed is limited in every case to the value of _RVEL_CART or _RVEL_ORI.

Default:	IMOUSE_MAX[1] = 520
	IMOUSE_MAX[2] = 400
	IMOUSE_MAX[3] = 440
	IMOUSE_MAX[4] = 520
	IMOUSE_MAX[5] = 400
	IMOUSE_MAX[6] = 450

DOCUMENTATION

REIS GMBH & CO MASCHINENFABRIK

TRAVERSING MODULES

Title: Transformation of user programs (Transformation function)

Authors:	May, Dr. Dresselhaus
Date:	November 1998
Control:	RSV
Software:	2.0

Date:	1.12.98
Release:	Elter

TABLE OF CONTENTS
=================

1. Description of task

The transformation function of the robot control enables the user to transform a generated and fixed programmed user program (AWP = UP) in a suitable manner.

Depending on the defined transformation parameters the following tasks can be solved with the transformation function:

1.) User programs can be taken for shifted and/or turned workpieces without having to correct teach-in of the position steps.

2.) User programs can be used on several installations with almost identical arrangement of the work envelope without any correction of teaching.

3.) User programs can also be used for workpieces being mirror-inverted to the original workpiece without correction teaching of positions.

4.) User programs can be used for similar workpieces having a defined ratio of size to the original workpiece.

5.) User programs for workpieces with plane, straight line or point symmetries can be also used for the mirror-inverted workpieces.

2. Definition of the user command TRAFO_6D

Transformation of user programs is executed with the user command TRAFO_6D. With this command a source program is transferrred into a destination program. The position steps of the source program are transfomed depending on the transformation parameters; all other steps are copied into the destination program without any modifications.

It is possible to maintain the hand/tool orientation of the source program or to transform it.

For all transformations it has to be observed that the transformed positions are within the robot work envelope so that approach with robot is possible. With extreme shiftings or twistings of the coordinate systems there might occur transformation errors after transformation.

The user command TRAFO_6D executes the transformation of the position steps of the source program depending on the 5 command parameters.

The command is defined as follows:

TRAFO_6D	Ref_source	ref_q,
	Ref_destination	ref_z,
	Source	quelle,
	Destination	ziel,
	Control word	kontr_var

The significance of the command parameters is indicated in chapter 2.1. Chapters 2.2 to 2.6 contain the descriptions for the transformation procedures.

2.1 Definition of the command parameters

a) Program name of the program "reference-source"

This program contains the definitions for the coordinate system of the source program. Its structure depends on the mode of transformation and is described for the individual transformation modes.

b) Program name of the program "reference-destination"

This program contains the definitions for the coordinate system of the destination program. The structure depends on the transformation mode and is described for the individual transformation modes.

c) Program name of the source program

The source program contains the program to be transformed. The source program is arbitrarily structured. All position steps of the source program are transformed. Main and sub-programs can be transformed with the command TRAFO_6D.

d) Program name of the destination program

The destination program is generated by the command TRAFO_6D. If overwriting of the destination program is allowed, an already existing program with the same name will be deleted prior to this. If overwriting is not allowed, this will cause an error message in this case. The program type of the source program (main or sub-program) will be maintained.

If the source program contains global variable definitions, they are copied into the destination program and will be invalidated there, in order to avoid double definitions. If the names of the source and the destination program are identical, i.e. the source program itself is modified, the variable definitions remain valid.

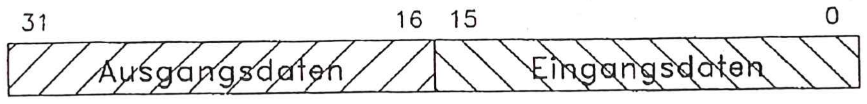
In the normal case the program names are entered during programming of the command as fixed character strings. Besides this, automatic generation of the names is possible. In this case, the program name is indicated by the name of the string variable (S...) instead of "prg_name".

e) Control variable

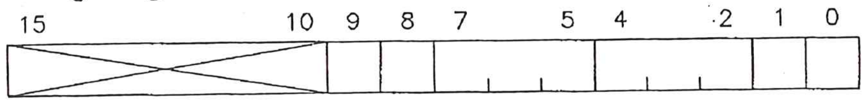
The control variable has to be of the integer type and consists of 4 bytes - 2 bytes each for data input and output. The structure of the control variable is shown in **ill. 1**.

The input data are entered in the variable prior to calling the command and will be kept even after processing of the command. The output data are entered in the output array of the variable with the transformation command. The error numbers entered in the output data array and their significance are summarized in the table of chapter 4.

Kontroll-Variable



Eingangsdaten



Bit 0: 0 evtl. vorhandenes Zielprogramm nicht überschreiben
1 evtl. vorhandenes Zielprogramm überschreiben

Bit 1: 0 Orientierung wird transformiert
1 Orientierung wird nicht transformiert

Bit 2–4: Transformationsart
000 Shiften
001 Strecken/Stauchen/Spiegeln
010 Shiften mit Spiegeln

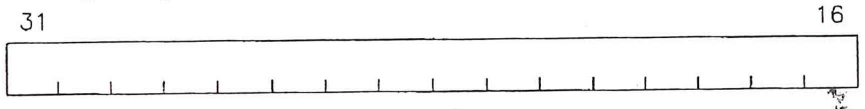
Bit 5–7: Transformationsparameter
000 Offset in kartes. Koordinaen } bei Transformationsart
001 Offset in Zylinderkoordinaten } Shiften

000 Bezugssystem Ebene } bei Transformationsart
001 Bezugssystem Gerade } Strecken/Stauchen/Spiegeln
010 Bezugssystem Punkt }

Bit 8: 0 keine Fehlerausgabe auf PHG
1 Fehlerausgabe auf PHG

Bit 9: 0 Auswertung aller Fehler der Vorzeichenberechnung
1 nur fatale Fehler der Vorzeichenberechnung werden ausgewertet

Ausgangsdaten



Bit 16–31: Fehlernummer

Ill. 1: Structure of the control variable

2.2 General 6D-transformation

Due to the general 6D-transformation it is possible to shift a user program taught with regard to a certain coordinate system into all 6 degrees of freedom (3 translatory and 3 rotatory ones).

Doing so, you assume for instance that a machining program has been taught for a workpiece which has a fixed position regarding a workpiece carrier. If the workpiece carrier was displaced within the robot work envelope or was transferred to another machine, usually all positions of the processing program must be taught again. This correction of teaching is avoided with the command TRAFO_6D.

2.2.1 Preset of shift by taught positions

For use of the 6D-transformation it is necessary to select three significant points of the workpiece carrier in the original position, to approach them with a defined tool (tip or similar) and to file them as positions in the program "reference-source". The same three points are also taught on the displaced carrier and stored in the program "reference-destination".

The original program (source program) is processed with the TRAFO_6D command and transferred into a destination program. By means of the destination program the workpiece is now machined in the shifted position **(ill. 2).**

When storing the positions in the reference programs, it has to be observed that all positions are defined in the same coordinate system. Either all positions have to be within the basic coordinate system (frame number = 0) or in a freely defined coordinate system (frame number >0).

2.2.2 Numerical preset of shift

Besides the source and destination coordinate system given by 3 reference points each, the shifting vector can also be given numerically.

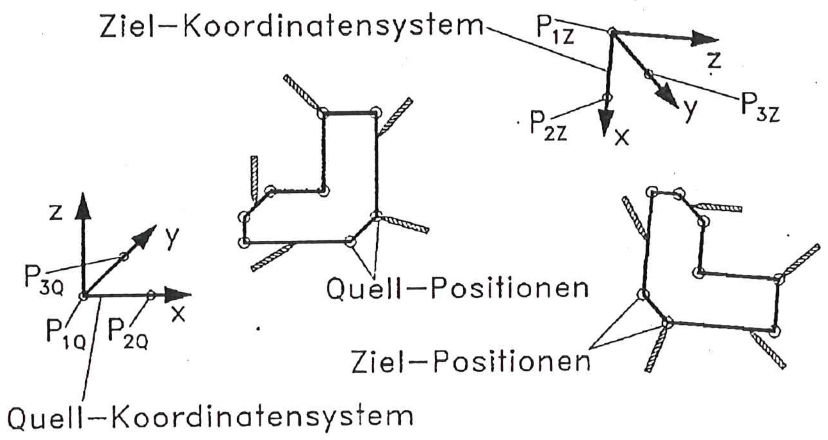
Depending on the transformation parameter being entered in the control variable (see ill.1), the offsets of the shift are stored in the components of a real array in the following manner.

The following values must be entered in the real array:

	cartesian coordinates	cylinder coordinates
1	delta x	delta rho
2	delty y	delta phi
3	delta z	delta z
4	delta A	delta A
5	delta B	delta B
6	delta C	delta C

For definition of the individual parameters please see ill. 8.
If a robot is equipped with additional axes, another array element has to be reserved for each additional axis; the offsets for adjustment of additional axes will be entered there (see chapter 2.7).

Structure of the reference programs for general 6D-transformation is defined in chapter. 3.1.



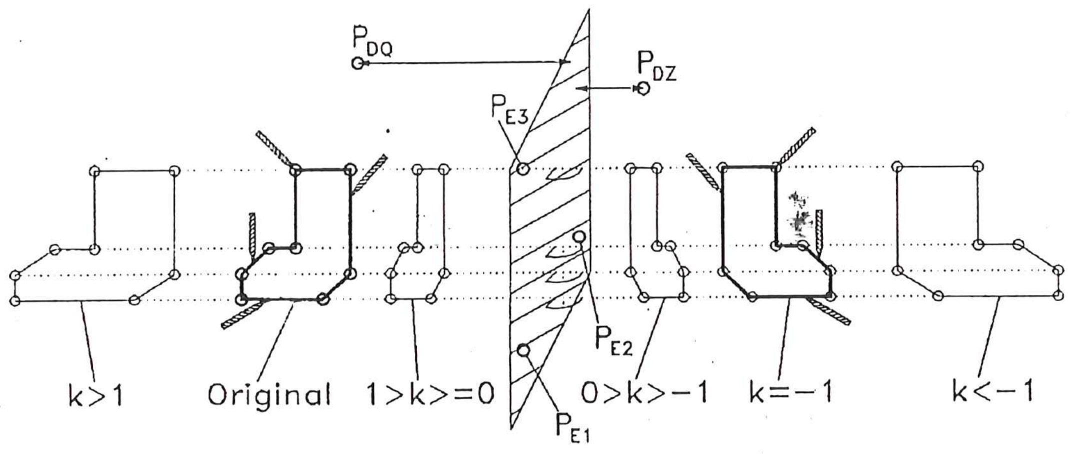
Ill. 2: General 6D-transformation

2.3 *Expansion, shrinking and mirror inversion of user programs on a plane*

By means of this transformation mode the user is enabled to reduce the programming efforts for workpieces with symmetries with regard to one plane. Furthermore, by means of the transformed user programs, without any correction of teaching it is possible to machine similar workpieces having a fixed ratio of size with regard to one plane or mirror-inverted workpieces having a plane symmetry.

The reference plane for the transformation (mirror plane) is given by three positions which are stored in the program "reference-source".

The size factor can be given in two ways: by numeric indication in a real variable or constant or by indicating two positions having a certain distance from the reference plane. The ratio of distance between the two positions indicates the size factor for the transformation **(Ill. 3).**



Ill. 3: Transformations regarding a plane

With the size factor k all positions of a source program with regard to the plane indicated in "reference-source" are expanded, shrinked and also mirror-inverted in case the k-factor has a negative sign.

Depending on the k factor, the following is possible:

$k > 1$	expansion
$1 > k \geq 0$	shrinking
$0 > k > -1$	shrinking and mirror inversion
$k = -1$	mirror inversion
$k < -1$	expansion and mirror inversion

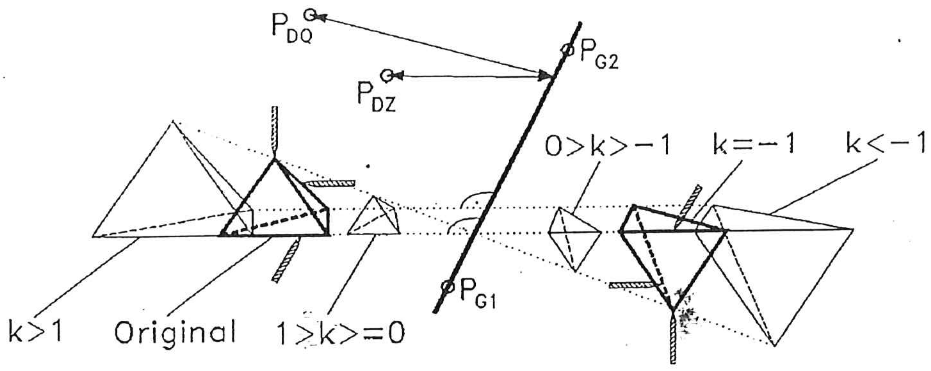
Structure of the reference programs is defined in chapter 3.2 .

2.4 Expansion, shrinking and mirror inversion of user programs on a straight line

This transformation mode can be used for workpieces having a symmetry with regard to a straight line (rotation symmetry).

Analogue to the plane symmetry there are the same possibilities for transformation of programs.

The reference straight line is given by two positions which are stored in the "reference-source" program. For indication of the size factor k also in this case numeric indication in a real variable or constant is possible, as well as the indication by two positions with defined distances to the reference straight line. The ratio of distance results in the size factor k **(Ill. 4).** The direction of the positions P_{DQ} and P_{DZ} is not taken into consideration when calculating k.



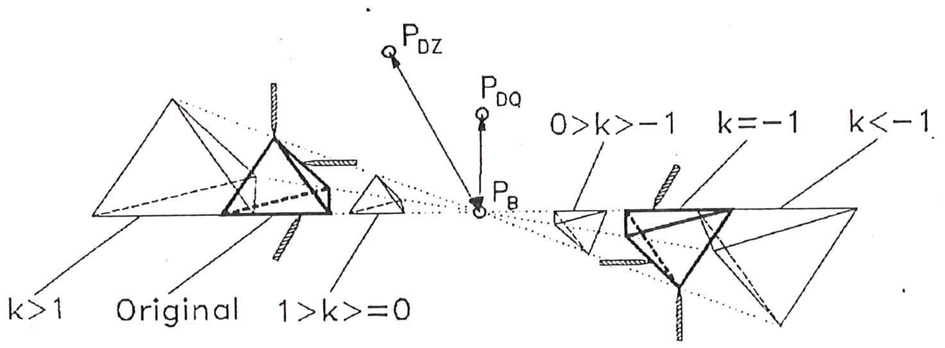
Ill. 4: Transformations regarding a straight line

Structure of reference programs for transformation of user programs regarding a straight line is defined in chapter 3.3 .

2.5 Expansion, shrinking and mirror inversion of user programs in a point

This transformation mode is provided for point-symmetric workpieces. User programs for machining of point-symmetric workpieces having a certain ratio of size or for mirror-inverted workpieces are transferred with this transformation mode **(Ill. 5).**

The reference point of the transformation is stored as position in "reference-source". The size factor k is given as real variable or constant analogue to the plane and straight line transformation or is calculated from the ratio of distance of two positions from the reference point. The direction of the positions PDQ and PDZ is not taken into consideration when calculating k.



Ill. 5: Transformations regarding a point

Structure of the reference programs for transformation of user programs regarding a point is defined in chapter 3.4 .

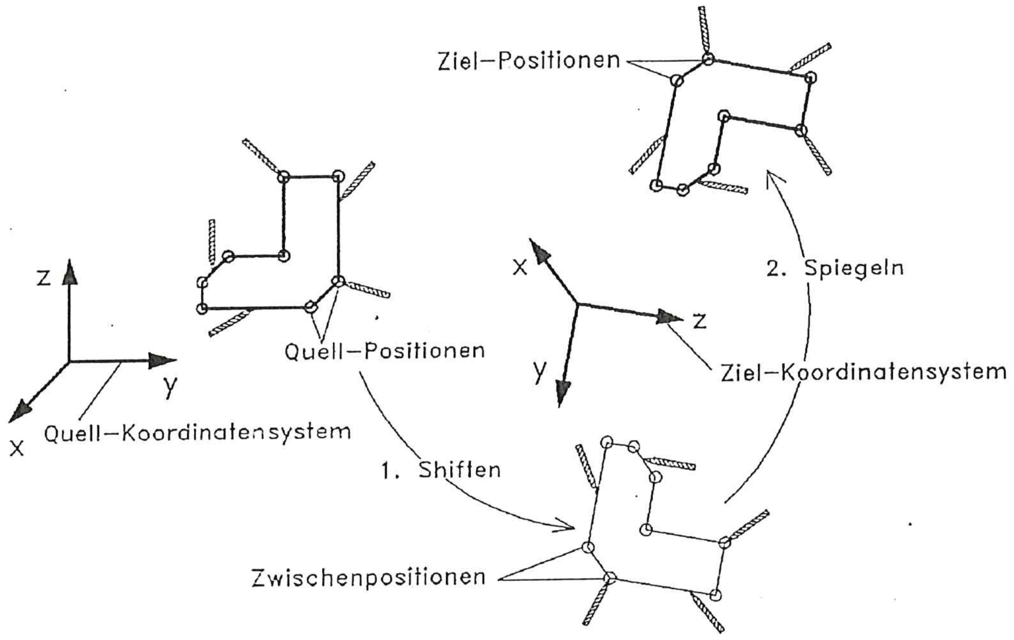
2.6 *6D-transformation of user programs with simultaneous mirror inversion*

The variant described here is especially suited for applications which require processing of mirror-inverted workpieces of same size and where exact determination of the mirror plane is difficult.

For preparation of the transformation only three significant points have to be selected at the original workpiece and are to be filed as positions in the program "reference-source". Furthermore, the corresponding points at the mirror-inverted workpiece must be stored as positions in the same order in the program "reference-destination".

Due to this transformation, the source program is shifted in all 6 dimensions with one step and then is mirror-inverted on the x-z plane of the destination coordinate system **(Ill. 6).**

Indication of a size factor k is not possible for this transformation mode.



Ill. 6: 6D-transformation with mirror inversion

Structure of the reference programs is defined in chapter 3.5 .

2.7 Treatment of additional axes

Besides transformation of positions in the main axes system of the robot, also the additional axes of the robot are transformed with the command TRAFO_6D.

For robots being equipped with additional axes it has to be observed that during programming of the positions in the reference programs (position steps or position variables) the additional axes are exactly adjusted to each other in "reference-source" or "reference-destination".

For treatment of additional axes, there are two cases depending on the transformation mode.

In the normal case (without mirror inversion) the additional axes are adjusted by means of the presettings in the reference programs. In cases where source programs are mirror-inverted, the additional axes are mirror-inverted, too.

2.7.1 Transformation of additional axes without mirror inversion

For transformation of additional axes without mirror inversion, the differences of the increments of all additional axes are calculated.

Presetting for the adjustment is made in two manners:

a) If **two different reference programs** "reference-source" and "reference-destination" exist:

   The differences of the increments are calculated from the positions in "reference-source" and "reference-destination". The calculated differences are added to the incremental values of the corresponding additional axes of the positions of the source program.

b) If only **one reference program** exists:

   "reference-source" = "reference-destination"

   Adjustment of additional axes is made by numerical presettings; for rotary axes they are indicated in degrees, for linear axes they are indicated in millimeters.

   The additional axes are adjusted by the indicated angles (for rotary axes) or distances (for linear axes).

If the programs "reference-destination" and "reference-source" had been programmed at different stations, further measures will be required for transformation of the additional axes.

In this case, the kinematic order of the additional axes at both stations is checked (entries in the group table). If the table shows that different additional axes are active, the increments of the corresponding axes are calculated.

If for instance at station 1 (reference-source) axes 7, 8 and 9 are active and at station 2 (reference-destination) axes 7, 10 and 11, the increments of axes 8 and 10 as well as the increments of axes 9 and 11 are calculated with each other. The increments of axis 7 are calculated as described above.

2.7.2 Transformation of additional axes with mirror inversion

The transformation of additional axes with mirror inversion is executed if the following conditions are met:

- The reference programs "reference-source" and "reference-destination" are different,
- The frame numbers in "reference-source" and "reference-destination" are equal,
- The size factor "k" in transformation mode expansion/shrinking/mirror inversion is negative.

Treatment of additional axes results from the position steps of the reference programs.

1.) Additional axes having identical positions in "reference-source" and "reference-destination will not be transformed.

2.) Additional axes having different positions in "reference-source" and "reference-destination" will be mirror-inverted with regard to their reference position. This reference position results from the average value of the increments in the two reference programs.

2.8 Transformation of positions with different frame number

With the transformation function positions taught in the basic coordinate system (frame number = 0) as well as positions taught in a freely defined coordinate system (frame number > 0) can be transformed.

2.8.1 Conditions for transformation of different position types

1.) The reference programs "reference-source" and "reference-destination" must only contain positions with the same frame number.

2.) Frame numbers > 0 may vary between "reference-source" and "reference-destination". They have to coincide within the reference programs.

3.) The increments of the additional axes may vary between "reference-source" and "reference-destination". They have to coincide exactly within the reference programs.

2.8.2 Reference programs contain positions with frame number = 0

If the reference programs contain positions in the basic coordinate system, the source program may contain positions with random frame number. The frame number of the transformed positions is adopted from the corresponding position from the source program.

2.8.3 Reference programs contain positions with frame number > 0

If the reference programs contain positions in a freely defined coordinate system the source program must only contain positions with frame number = 0 or with same frame number like in the reference program.

The transformation allows:

- transformation of positions with frame number = 0 into positions with frame number = 0.

- transformation of positions with frame number > 0 into positions with that frame number the positions have in "reference-destination".

2.9 Conditions for transfer of programs to other machines

Cartesian coordinates are filed in user programs. The cartesian coordinate system for each robot is defined by measurement of the robot.

Due to wrong measurement of a robot the cartesian coordinate system differs from a real rectangular cartesian coordinate system. Thus, all programs taught with this robot contain faulty cartesian values. Since the taught (faulty) positions are always approached again in the same manner for the same robot, this error doesn't have an effect on programs which are not transferred to other machines.

If a program is transferred from one robot to another, the program will run faulty there, since now the different coordinate systems cause differences in positions.

In order to avoid such errors, already prior to programming of the original program the correct measurement of both robots has to be checked.

Correct measurement can be checked by a simple test. At the first machine, at least three points are taught at a random workpiece which should be as large as possible.

The workpiece and the program are transferred to the destination machine. The workpiece is placed in such way that the taught positions of the program coincide with those of the workpiece.

If this is not possible, both robots have a different measurement and the transformations are not possible in a correct way.

2.10 Remark concerning the used tool data

For all transformations executed with the TRAFO_6D command, tool data stored in the position steps of the source program are used.

If a position step doesn't contain any valid tool data, the current valid tool data are used for the transformation being active in the user program where the command TRAFO_6D is processed.

Positions without valid tool data are, for instance, positions generated by offline programming.

If a transformation mode is selected where positions are mirror-inverted, correct orientation of the tool in direction of the tool angle must be observed when mounting the tool.

This can be tested e.g. by moving in hand operation in the mode HAND/CARTESIAN. If the robot then won't move towards the tool direction during movement in U-direction (movement key +A/-A), this causes faulty orientations for mirror-inverted positions.

3. Structure of the reference programs

The reference programs of the command TRAFO_6D serve for definition of coordinate or reference systems for the different transformation modes. The format of the reference programs has to be exactly kept in accordance with the definitions given in chapters 3.1 to 3.5.

The reference programs "reference-source" and "reference-destination" can be selected identical or not identical. The structure of the programs is different for both variants.

3.1 General 6D-transformation

a) Reference programs not identical

The transformation parameters are given by 6 positions which determine the source and the destination coordinate system.

- program "reference-source"

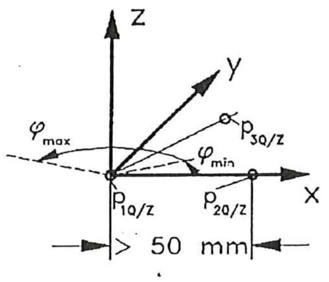
```
MP/SP       REFQ        or        MP/SP       REFQ
TOOL        T                     TOOL        T
POSITION    P1Q                   VAR_POS     P1Q
POSITION    P2Q                   VAR_POS     P2Q
POSITION    P3Q                   VAR_POS     P3Q
END                               END
```

- program "reference-destination"

```
MP/SP       REFZ        or        MP/SP       REFZ
TOOL        T                     TOOL        T
POSITION    P1Z                   VAR_POS     P1Z
POSITION    P2Z                   VAR_POS     P2Z
POSITION    P3Z                   VAR_POS     P3Z
END                               END
```

The three positions form a coordinate system each. P1Q or P1Z indicate the coordinate origin. P1Q and P2Q or P1Z and P2Z form the x-axis of the coordinate system. P3Q and P3Z are in the x-y-plane of the coordinate system **(Ill. 7)**.

To increase the accuracy a minimum distance of 50 mm is required for the positions; the angle has to be between 30 degrees and 150 degrees. If these conditions are disregarded, an error message will be displayed.



Ill. 7: Definition of the coordinate systems

b) reference programs identical

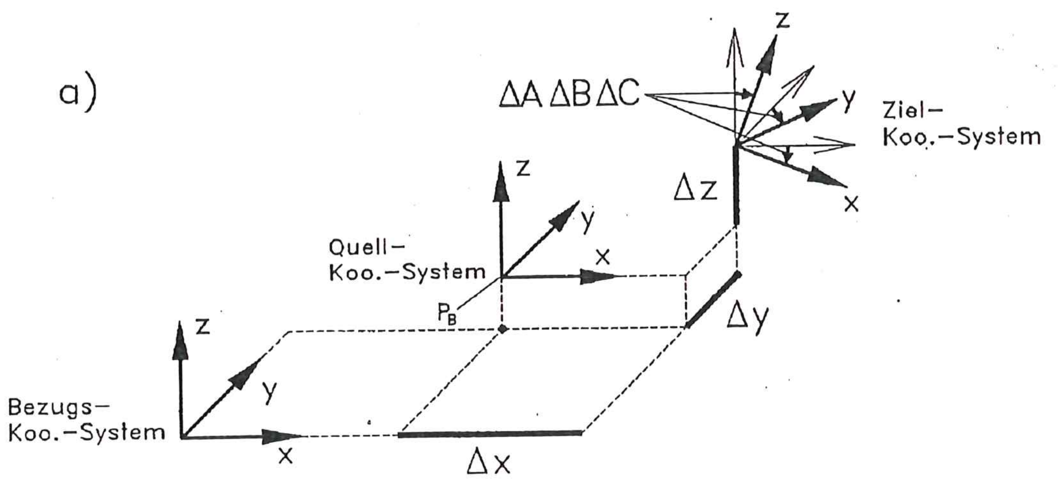
The transformation parameters are given by a position and a real type array.

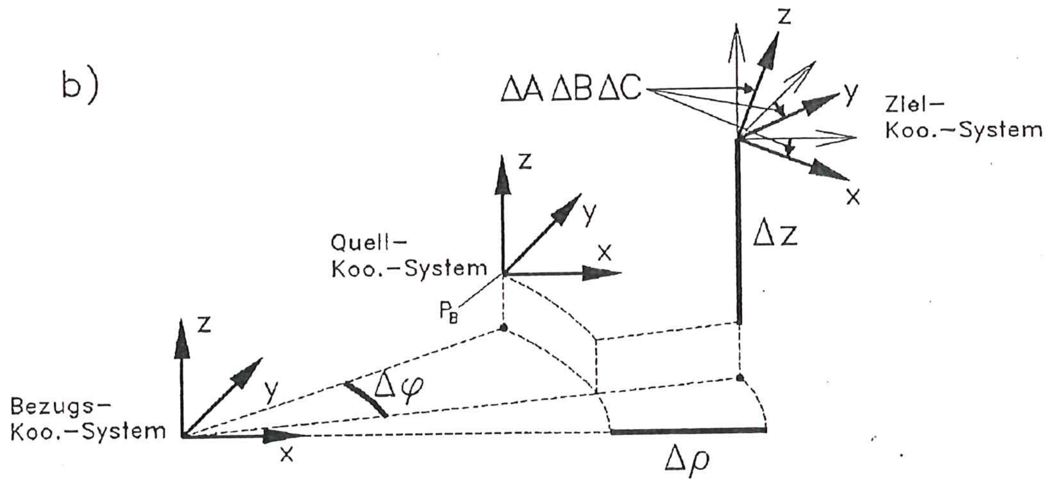
- program "reference-source" = "reference-destination"

```
MP/SP       REFQ        or          MP/SP       REFQ
VAR         RARR[6]                 VAR         RARR[6]
TOOL        T                       TOOL        T
POSITION    PB                      VAR_POS     PB
END                                 END
```

The position PB is the reference point for the transformation.
In the real array RARR, the shifts and torsions are entered (see ill. 8). If shifting in cartesian coordinates is selected, delta x, delty y and delta z will be stored in the first three elements; with shifting in cylinder coordinates, delta rho, delta phi and delta z will be stored in the first three elements.

If there are additional axes, another array element has to be reserved for each additional axis. For an installation with 3 additional axes, consequently VAR RARR[9] has to be defined. The elements 7 to 9 in the above example contain the change of the additional axes 7 to 9 in degrees (for rotary axes) or mm (for linear axes).





Ill. 8: 6D-transformation with reference point and offset vector
a) offset indicated in cartesian coordinates
b) offset indicated in cylinder coordinates

3.2 Expansion, shrinking and mirror inversion on a plane

a) reference programs not identical

The size factor k is given by two positions.

- program "reference-source"

```
MP/SP         REFQ       or       MP/SP       REFQ
TOOL          T                   TOOL        T
POSITION      PDQ                 VAR_POS     PDQ
POSITION      PE1                 VAR_POS     PE1
POSITION      PE2                 VAR_POS     PE2
POSITION      PE3                 VAR_POS     PE3
END                               END
```

- program "reference-destination"

```
MP/SP         REFZ       or       MP/SP       REFZ
TOOL          T                   TOOL        T
POSITION      PDZ                 VAR_POS     PDZ
END                               END
```

The size factor k is calculated with the ratio of the distances of the positions PDQ and PDZ from the reference plane.

k = distance (PDZ) / distance (PDQ)

If PDQ and PDZ are situated on the same side of the plane, k is set positive, if they are situated on different sides of the plane, k is set negative (mirror inversion!). The distances of PDQ and PDZ from the reference plane have to be 50 mm at least.

The reference plane is built by the three positions PE1, PE2 and PE3.

To increase the accuracy, just like for the general 6D-transformation, there is required a distance of 50 mm between the positions and an angle between 30 and 150 degrees.

If also additional axes shall be transformed during mirror inversion ($k < 0$), this is only possible within one station.

Presetting for the additional axes' transformation results from the position steps in the reference programs (see chapter 2.7.2).

b) reference programs identical

The size factor k is given numerically in a real variable or real constant.

- program "reference-source" = "reference-destination"

```
MP/SP          REFQ      or      MP/SP          REFQ
CONST/VAR      RVAR              CONST/VAR      RVAR
TOOL           T                 TOOL           T
POSITION       PE1               VAR_POS        PE1
POSITION       PE2               VAR_POS        PE2
POSITION       PE3               VAR_POS        PE3
END                              END
```

The size factor k is indicated by RVAR. The reference plane is built by the three positions PE1, PE2 and PE3.

To increase the accuracy, just like for the general 6D-transformation, there is required a distance of 50 mm between the positions and an angle between 30 and 150 degrees.

If also additional axes shall be adjusted during transformation, another definition of the real variable is required.

Instead of the definition CONST/VAR RVAR, the definition must now be VAR RARR[n+1], "n" being the number of additional axes. Accordingly, for three additional axes the definition must be VAR RARR [4]. In the elements 2 to 4 in the above example, the additional axes' adjustments of axes 7 to 9 have to be entered. For rotary axes indication is made in degrees, for linear axes in mm.

3.3 Expansion, shrinking and mirror inversion on a straight line

a) reference programs not identical

The size factor k is given by two positions.

- program "reference-source"

```
MP/SP       REFQ    or    MP/SP       REFQ
TOOL        T             TOOL        T
POSITION    PDQ           VAR_POS     PDQ
POSITION    PG1           VAR_POS     PG1
POSITION    PG2           VAR_POS     PG2
END                       END
```

- program "reference-destination"

```
MP/SP       REFZ    or    MP/SP       REFZ
TOOL        T             TOOL        T
POSITION    PDZ           VAR_POS     PDZ
END                       END
```

The size factor k is calculated with the ratio of the distances of the positions PDQ and PDZ from the reference straight line.

k = distance (PDZ) / distance (PDQ)

Factor k is always positive for variant a); mirror inversion of programs is not possible. The distances of PDQ and PDZ from the reference straight line have to be at least 50 mm.

The reference straight line runs through the positions PG1 and PG2. To increase the accuracy, PG1 and PG2 must have a minimum distance of 50 mm.

Mirror inversion of additional axes is not possible.

b) reference programs identical

The size factor k is given numerically in a real variable or real constant.

- program "reference-source" = "reference-destination"

```
MP/SP        REFQ      or      MP/SP        REFQ
VAR/CONST    RVAR              VAR/CONST    RVAR
TOOL         T                 TOOL         T
POSITION     PG1               VAR_POS      PG1
POSITION     PG2               VAR_POS      PG2
END                            END
```

The size factor k is indicated by RVAR. The reference plane is built by the three positions PG1 and PG2. In order to increase accuracy, a minimum distance of 50 mm between PG1 and PG 2 is required.

If additional axes are to be adjusted the reference program must be modified (see last paragraph in chapter 3.2, section b).

3.4 Expansion, shrinking and mirror inversion in a point

a) reference programs not identical

The size factor k is given by two positions.

- program "reference-source"

```
MP/SP       REFQ     or      MP/SP       REFQ
TOOL        T                TOOL        T
POSITION    PDQ              VAR_POS     PDQ
POSITION    PB               VAR_POS     PB
END                          END
```

- program "reference-destination"

```
MP/SP       REFZ     or      MP/SP       REFZ
TOOL        T                TOOL        T
POSITION    PDZ              VAR_POS     PDZ
END                          END
```

The size factor k is calculated with the ratio of the distances of the positions PDQ and PDZ from the reference point.

k = distance (PDZ) / distance (PDQ)

Factor k is always positive for variant a); mirror inversion of programs is not possible. The distance from PDQ and PDZ from the reference point has to be at least 50 mm. The reference point for transformation is given by the position PB.

Mirror inversion of additional axes is not possible.

b) reference programs identical

The size factor k is given numerically in a real variable or real constant.

- program "reference-source" = "reference-destination"

```
MP/SP       REFQ     or      MP/SP       REFQ
VAR/CONST   RVAR             VAR/CONST   RVAR
TOOL        T                TOOL        T
POSITION    PB               VAR_POS     PB
END                          END
```

RVAR indicates the size factor k. The reference point for the transformation is defined by the position PB.

If additional axes shall be adjusted, the reference program has to be changed (see last paragraph in chapter 3.2, section b).

3.5 6D-transformation with simultaneous mirror inversion

The structure of the reference programs for this transformation mode is identical to the one valid for general 6D-transformation. Only variant a) (reference programs not identical) is allowed.

- program "reference-source"

```
MP/SP          REFQ     or      MP/SP      REFQ
TOOL           T                TOOL       T
POSITION       P1Q              VAR_POS    P1Q
POSITION       P2Q              VAR_POS    P2Q
POSITION       P3Q              VAR_POS    P3Q
END                             END
```

- program "reference-destination"

```
MP/SP          REFZ     or      MP/SP      REFZ
TOOL           T                TOOL       T
POSITION       P1Z              VAR_POS    P1Z
POSITION       P2Z              VAR_POS    P2Z
POSITION       P3Z              VAR_POS    P3Z
END                             END
```

The structure of the source coordinate system is analog to the procedure described in 3.1 under a).

For building the destination coordinate system the position P3Z first is mirror-inverted in the x-z-plane of the destination coordinate system and then the destination coordinate system is formed by the positions P1Z, P2Z and the mirror-inverted position P3Z.

If additional axes shall also be transformed during mirror inversion, this is only possible within one station.

Presetting for additional axes' transformation results from the position steps in the reference programs (see chapter 2.7.2).

4. Table of error messages

When processing the command TRAFO_6D, two different modes of error messages are generated: errors given by the interpreter during processing of the command (programming errors) and errors being sent by the transformation function (transformation error).

a) programming error

description	remedy
"program in the stack"	check parameter of TRAFO_6D; rename destination program

b) Transformation error

Transformation errors are entered into the output array of the control register and can be evaluated after processing of the command TRAFO_6D.

Besides this, it is possible to indicate the error number on the PHG (teach pendant). For this purpose, bit 8 of the control word has to be set to "1" **(Ill. 1)**. In this case the running program is aborted and the error number is indicated on the PHG.

The emitted error message is as follows:

TRAFO_6D-error: nnn

For the errors marked with (*) in the following table, after acknowledgement of the error message, the step content indication in addition is set to the currently processed step of the source program.

The significance of the error numbers is shown in the following table:

error no.	description	remedy
1	The program "reference-source" was not found	Generate "reference-source" program; check the parameters of the TRAFO-6D command
2	"reference-source" contains invalid step	Validate all steps in "reference-source"
3	Number of positions in "reference-source" is wrong	Check syntax of the "reference-source" program; check in the control variable, whether correct transformation mode was selected
4	Positions in "reference-source" don't build a coordinate system	Check positions in reference-source" (distances, angles)
5	Increments of the additional axes in "reference-source" are not identical	Identical adjustment of increments of the additional axes
6	"reference-source" has wrong format	Check syntax of the "reference-source" depending on the transformation mode
7	"reference-source" contains wrong step	Check syntax of the "refeference-source" depending on the transformation mode
8	Definition of a variable/constant in "reference-source" not found	Program variable/constant definition
9	Variable in "reference-source" not initialized	Describe the variable with a valid value
10	Distance of the positions in "reference-source" is too small	Check positions; adjust distance > 50 mm
11	Angle between positions in "reference-source" is too small	Check positions; adjust angle > 30 degrees

error no.	description	remedy
12	Number of additional axes in the machine data (IAXES_DESCR) doesn't coincide with the one in the position step in "reference-source"	Modify machine data; teach positions again
13	"reference-source" contains positions with wrong frame	Teach all positions in "reference-source" and "reference-destination" in the same coordinate system
14	position in "reference-source" contains wrong frame number	Teach all positions in "reference-source" with frame number > 0 at the same station
15	Error with rotary table-world-transformation in "reference-source"	Check positions in "reference-source"; check calibration of the additional axes; check station number
50	real variable in "reference-source" not defined as array	Correct the definition of the real variable
51	real array in "reference-source" has wrong number of elements	Correct the definition of the real array. Adjust number of elements acc. to definition depending on the transformation mode.
101	The program "reference-destination" was not found	Generate program "reference-destination"; check parameters of the command TRAFO_6D
102	"reference-destination" contains invalid step	Validate all steps in "reference-destination"
103	Number of positions in "reference-destination" is wrong	Check syntax of the program "reference-destination"; check in control variable whether correct transformation mode was selected
104	Positions in "reference-destination" don't build a coordinate system	Check positions in "reference-destination" (distances, angles)

error no.	description	remedy
105	Increments of the additional axes in "reference-destination" are not identical	Identical adjustment of the increments of the additional axes
106	"reference-destination" has wrong format	Check the syntax of "reference-destination" depending on the transformation mode
107	"reference-destination" contains wrong step	Check the syntax of "reference-destination" depending on the transformation mode
108	Definition of a variable/constant in "reference-destination" not found	Program variable/ constant definition
109	variable in "reference-destination" not initialized	Describe the variable with a valid value
110	Distance of the positions in "reference-destination" is too small	Check positions; adjust distance > 50 mm
111	Angle between positions in "reference-destination" is too small	Check positions; adjust angle to > 30 degrees
112	Number of additional axes in the machine data (IAXES_DESCR) doesn't coincide with that in the position step in "reference-destination"	Modify machine data; teach positions again
113	"reference-destination" contains positions with wrong frame	Teach all positions in "reference-source" and "reference-destination" in the same coordinate system
114	position in "reference-destination" contains wrong frame number	Teach all positions in "reference-destination" with frame number > 0 at the same station

error no.	description	remedy
115	Error with the rotary table-world-transformation in "reference-destination"	Check positions in "reference-destination"; check calibration of the additional axes; check station number
150	"reference-destination" has the same name as "reference-source"	Check name of "reference-destination"; check transformation mode
151	"reference-destination" has not the same name as "reference-source"	Check name of "reference-destination"; check transformation mode/parameters
201	Source program not found	Generate source program; Check parameter of the TRAFO_6D command
202 (*)	Number of additional axes in the machine data (IAXES_DESCR) and of a position in the source program don't coincide	Modify machine data; teach position again
203 (*)	Error with rotary table-world-transformation in the source program	Check positions in the source program; check calibration of additional axes; check station number
204 (*)	Error with world-rotary table-transformation in the source program	Check positions in the source program; check calibration of additional axes; check station number
205 (*)	Position in the source program contains wrong frame number	Check positions in the source program; check station number of the reference programs
206 (*)	Source program contains VAR_POS step	Generate source program in such a way that no VAR_POS steps are contained.
301	Destination program exists already	Modify control-variable of the command TRAFO_6D (permit overwriting); Change name of the destination program
302	No memory for destination program	Check user program memory; back-up programs and delete if required

401	Control variable contains invalid value	Check value of the control variable and enter valid value acc. to definition

error no.	description	remedy
402	Control variable contains invalid transformation mode	Check value of the control variable and enter valid value acc. to definition
403	Control variable contains invalid transformation parameter	Check value of the control variable and enter valid value acc. to definition
501	Internal error during copying of the source program	Back-up current user memory and data; Contact Reis
502	Internal error during transformation	Back-up current user memory and data; Contact Reis
503	Internal error when reading the position data	Back-up current user memory and data; Contact Reis
504 (*)	Internal error during determination of the axes' signs in the source program	Check position steps of the source program
505 (*)	Internal error during determination of the axes' signs in the destination program	Check position steps in the source program; check transformation mode and parameters; check source and destination coordinate systems
506 (*)	Internal error during determination of the axes' signs in the destination program	Check position steps in the source program; check transformation mode and parameters; check source and destination coordinate systems
507	Internal error concerning axes' factors of the additional axes	Check machine data; correct adjustment of axes' factors of the additional axes

reis

DOCUMENTATION

REIS GMBH & CO MASCHINENFABRIK OBERNBURG

PROGRAMMING

Title: **COPY-command**

Author:	Wenzel
Date:	04.12.98
Control version:	RSV
Software version:	2.3

Date:	07.01.99
Release:	Elter

1. Description of task

The COPY command is used for inscription of variables.

2. Operating manual

2.1 COPYING VARIABLES

The command belongs to the command group **Var** and can be found in the first menu.

Command syntax:

```
COPY        Source:                 ,   Variable:
            <name of a variable>    ,   <name of a variable>
            <   constant    >       ,   <name of a variable>
```

All variable types, even arrays, are allowed for source and destination. For variables with several components - those are the variables of the type **T**ool, **V**ector, **F**rame and **P**osition - source and destination must be of the same type. For the variables of the **I**nteger or **R**eal type a type conversion is executed. This also applies for system variables if they are also of the type integer or real.

The data of the source must be valid, the data of the destination will be validated after copying. For arrays only the array size addressed with the index must be valid, resp. only this array size will be validated.

Examples for correctly programmed COPY-commands:

```
COPY        2.5             ,   R-Var
COPY        8               ,   R-Var[Index]   (type conversion)
COPY        I-Var           ,   I-Var
COPY        I-Var           ,   R-Var[Index]   (type conversion)
COPY        R-Var[Index]    ,   I-Var[Index]   (type conversion)
COPY        T-Var           ,   T-Var
COPY        V-Var           ,   V-Var[Index]
COPY        P-Var[Index]    ,   P-Var[Index]
COPY        _I-VAR          ,   I-Var          (system variable)
COPY        _I-VAR          ,   R-Var[Index]   (type conversion)
COPY        _I-VAR          ,   _I-VAR         (system variable)
                                ,
```

Example for incorrectly programmed COPY-commands:

```
COPY        2.5             ,   V-Var
COPY        P-Var           ,   T-Var
```

2.2 COPYING STRUCTURAL ELEMENTS

In the ROBOTstar V it is possible to copy variables with different components (structures) component by component. In the ROBOTstar V the variable types Tool, Vector and Position are released for structure access. The same syntax is valid as described under item 2.1.

As separator for the structure access there was defined the „ . „ .
The structure element names in the ROBOTstar V are predefined and exact description of those is given below.
There is no distinction made between capitalization.
Copying of individual components with arrays of structure variables is **not yet possible** in the version RSV 2.3 .

Access to a structure element therefore must **always** be as follows:

<name of the variable>.<name of the structure element>

$\uparrow$

separator

2.2.1 STRUCTURE ACCESS WITH VECTORS

A variable type **Vector** is defined as follows in the ROBOTstar V:

1.	frame number
2.	X-shifting
3.	Y-shifting
4.	Z-shifting
5.	A-orientation offset
6.	B-orientation offset
7.	C-orientation offset
8. ... 31.	articulation angle offsets for axis 1 to axis 24

From this there are derived the following structure element names for copying in components:

As an example there are copied digital constants into a **Vector variable**:

```
COPY        Source:0     ,     Variable:V-Var.Frame
COPY        Source:0     ,     Variable:V-Var.X
COPY        Source:0     ,     Variable:V-Var.Y
COPY        Source:0     ,     Variable:V-Var.Z
COPY        Source:0     ,     Variable:V-Var.A
COPY        Source:0     ,     Variable:V-Var.B
COPY        Source:0     ,     Variable:V-Var.C
COPY        Source:0     ,     Variable:V-Var.A1
COPY        Source:0     ,     Variable:V-Var.A2
...
COPY        Source:0     ,     Variable:V-Var.A24
```

The following structure copying actions are executed with vectors in the ROBOTstar V:

COPY	Source:**<real-number>**	,	Variable:**V**-Var.**<structure element>**
COPY	Source:**<integer-number>**	,	Variable:**V**-Var.**<structure element>**
COPY	Source:**R**-Var	,	Variable:**V**-Var.**<structure element>**
COPY	Source:**I**-Var	,	Variable:**V**-Var.**<structure element>**
COPY	Source:**V**-Var.**<structure element>**	,	Variable:**I**-VAR
COPY	Source:**V**-Var.**<structure element>**	,	Variable:**R**-Var
COPY	Source:**V**-Var.**<structure element>**	,	Variable:**V**-Var.**<structure element>**

2.2.2 STRUCTURE ACCESS WITH POSITIONS

A variable type **Position** is defined as follows in the ROBOTstar V:

1.	**number of the main axes**
2.	**number of the additional axes**
3.	**position type**
4.	**frame number**
5.	**X-shifting**
6.	**Y-shifting**
7.	**Z-shifting**
8.	**A-orientation offset**
9.	**B-orientation offset**
10.	**C-orientation offset**
11. ... 34.	**articulation angle offsets for axis 1 to axis 24**

The following structure element names are derived from this for copying in components:

As an example number constants are copied into a **Position variable**:

```
COPY          Source:0    ,    Variable:P-Var.AA       /* Additional Axes */
COPY          Source:0    ,    Variable:P-Var.MA       /* Main Axes */
COPY          Source:0    ,    Variable:P-Var.PT       /* PosTyp */
COPY          Source:0    ,    Variable:P-Var.Frame
COPY          Source:0    ,    Variable:P-Var.X
COPY          Source:0    ,    Variable:P-Var.Y
COPY          Source:0    ,    Variable:P-Var.Z
COPY          Source:0    ,    Variable:P-Var.A
COPY          Source:0    ,    Variable:P-Var.B
COPY          Source:0    ,    Variable:P-Var.C
COPY          Source:0    ,    Variable:P-Var.A1
COPY          Source:0    ,    Variable:P-Var.A2
...
COPY          Source:0    ,    Variable:P-Var.A24
```

The following structure copying actions with positions are executed in the ROBOTstar V:

COPY	Source:**<real-number>**	,	Variable:**P**-Var.**<structure element>**
COPY	Source:**<integer-number>**	,	Variable:**P**-Var.**<structure element>**
COPY	Source:**R**-Var	,	Variable:**P**-Var.**<structure element>**
COPY	Source:**I**-Var	,	Variable:**P**-Var.**<structure element>**
COPY	Source:**P**-Var.**<structure element>**	,	Variable:**I**-Var
COPY	Source:**P**-Var.**<structure element>**	,	Variable:**R**-Var
COPY	Source:**P**-Var.**<structure element>**	,	Variable:**P**-Var.**<structure element>**

2.2.3 STRUCTURE ACCESS WITH TOOLS

A variable type **Tool** is defined in the ROBOTstar V as follows:

1.	**X-shifting**
2.	**Y-shifting**
3.	**Z-shifting**
4.	**rotation around the x-axis of the tool frame**
5.	**rotation around the y-axis of the tool frame**
6.	**rotation around the z-axis of the tool frame**

The following structure element names are derived from this for copying in components:

As an example number constants are copied into a **Tool variable**:

COPY	Source:0	,	Variable:**T**-Var.**X**
COPY	Source:0	,	Variable:**T**-Var.**Y**
COPY	Source:0	,	Variable:**T**-Var.**Z**
COPY	Source:0	,	Variable:**T**-Var.**RX**
COPY	Source:0	,	Variable:**T**-Var.**RY**
COPY	Source:0	,	Variable:**T**-Var.**RZ**

The following structure copying actions with tools are executed in the ROBOTstar V:

COPY	Source:**<real-number>**	,	Variable:**T**-Var.**<structure element>**
COPY	Source:**<integer-number>**	,	Variable:**T**-Var.**<structure element>**
COPY	Source:**R**-Var	,	Variable:**T**-Var.**<structure element>**
COPY	Source:**I**-Var	,	Variable:**T**-Var.**<structure element>**
COPY	Source:**T**-Var.**<structure element>**	,	Variable:**I**-Var
COPY	Source:**T**-Var.**<structure element>**	,	Variable:**R**-Var
COPY	Source:**T**-Var.**<structure element>**	,	Variable:**T**-Var.**<structure element>**

2.2.4 STRUCTURE ACCESS BETWEEN VECTORS AND POSITION VARIABLES

ROBOTstar V proved that it might be useful to exchange data between vector variables and position variables.

The following structure copying actions between vector variables and position variables are supported in the ROBOTstar V:

COPY	Source:**V**-Var.**<structure element>**	,	Variable:**P**-Var.**<structure element>**
COPY	Source:**P**-Var.**<structure element>**	,	Variable:**V**-Var.**<structure element>**

However, copying is only possible among identical "element groups". For this purpose the following element groups have to be observed:

element group „translation“:	structure elements **X, Y and Z**
element group „orientation“:	structure elements **RX, RY and RZ**
element group „art. angle“:	structure elements **A1 to A24**
element group „frame“:	structure element **frame**

It is only possible to exchange elements of **one** element group (e.g. translation) between position variables and vector variables.

Examples:

Possible combinations:

COPY	Source:**P**-Var.**x**	,	Variable:**V**-Var.**z**
COPY	Source:**P**-Var.**Frame**	,	Variable:**V**-Var.**Frame**
COPY	Source:**V**-Var.**A7**	,	Variable:**P**-Var.**A9**

Combinations which are **not** supported:

COPY	Source:**V**-Var.**Frame**	,	Variable:**P**-Var.**A9**
COPY	Source:**V**-Var.**z**	,	Variable:**P**-Var.**rx**

3. Error messages

The following error messages may occur in conjunction with the COPY-command:

<u>Definition not found!</u>

The variable indicated in the COPY-command was not yet defined or the definition of a LOC_CONST resp. LOC_VAR could not be found during nested search of local definitions.

<u>Definition is an array!</u>

This case occurs if one of the operands is a variable without array indication, the variable definition itself, however, is an array.

```
VAR     I_ARRAY[10]
COPY  5  , I_ARRAY
```

<u>Syntax error in the index!</u>

The array indication is incomplete, e.g. one of the parenthesis is missing or exists twice.

<u>Nesting not allowed!</u>

Nested array indications, e.g.

```
COPY 5  ,  I_ARRAY[I_VAR[5]]
```

are not allowed.

<u>Index variable not found!</u>

This error case might occur, if the index of an array is a variable and this has not yet been defined or the definition of a LOC_CONST resp. LOC_VAR could not be found during nested search of local definitions.

<u>Index variable not initialized!</u>

This error case might occur, if the index of an array is a variable and this has not yet been inscribed.

Index too big or too small!

The index must be greater zero and must not exceed the value indicated in the variable definition.

Definition is no array!

This case occurs, if one of the operands is a variable with array indication, and if the variable definition itself, however, is no array.

```
VAR     I_ARRAY
COPY 5  ,  I_ARRAY[3]
```

The operand is no variable!

The destination must only be a variable definition (no constant definition).

Wrong operand types!

The types of the Source and the destination do not match, a type conversion is not possible.

Variable not initialized!

The variable definition of the source was not yet inscribed and therefore must not be read.

Reading not allowed!

The system variable must not be read, not even, if it was already inscribed.

Writing not allowed!

The system variable must not be described. In the normal case then only reading is allowed.

Step not defined!

During execution of the command the system software recognized an error in the structure of the user programs.

DOCUMENTATION

REIS GmbH & Co Obernburg

Title: **General Information for Palletizing**

Authors	: Wenzel/Elter/May
Date	: 26. November 1998
Control	: RSV
Software	: 2.3

Release: **1.12.98/Elter**

Description of application task

The palletizing function to a great extent facilitates generation of programs for palletizing tasks.

The RSV offers the possibility to fulfill palletizing tasks via the so called palletizing pattern programs. The number of the palletizing patterns is only limited by the size of the user program memory since they are also stored there.

<u>Description of the palletizing pattern program</u>

A palletizing pattern program has a fixed format containing the initial identification, four definitions for global or local integer variables, three definitions for local vector variables, three definitions for local integer constants, a tool step and four positions.

The contents of the four global variables indicate the current loading state of the pallet.

The local constants determine the partscounts between the central point and the three pallet corner points.

For the partscounts instead of the local constants (step 15 to 17 in the example) also local or global variable definitions may be used. Thus, the pallet pattern can be changed in automatic mode. **The corresponding pallet must be initialized after each manipulation of these partscounts so that the delta vectors contain the correct data.**

Position variables may be used for the central point and the corner points.

Comment lines are allowed in the palletizing pattern program. The variable names are freely selectable.

<u>Example:</u>

```
 1    PAL          <pallet pattern name>
 2    C            GLOBAL VARIABLES
 3    C            =================
 4    VAR          ITOTAL
 5    VAR          ACTVALUE1
 6    VAR          ACTVALUE2
 7    VAR          ACTVALUE3
 8    C            LOCAL VECTOR VARIABLES
 9    C            ======================
10    LOC_VAR      V_ZP_EP1
11    LOC_VAR      V_ZP_EP2
12    LOC_VAR      V_ZP_EP3
13    C            LOCAL CONSTANTS
14    C            =================
15    LOC_CONST    I_PARTS1,<xxx>
16    LOC_CONST    I_PARTS2,<xxx>
17    LOC_CONST    I_PARTS3,<xxx>
18    C            PALLET POSITIONS
19    C            ==================
20    TOOL         T
21    POSITION     <central point>
22    POSITION     <corner point 1>
23    POSITION     <corner point 2>
24    POSITION     <corner point 3>
25    END
```

A pallet is defined by 7 parameters:

1.) Central point:
1st position in the palletizing pattern program
Position of the first part deposited on the pallet

2.) Corner point 1:
2nd position in the palletizing pattern program
With corner point 1 the first row of the palletizing pattern is defined. The parts are arranged into the direction of the corner point 1 beginning at the central point.

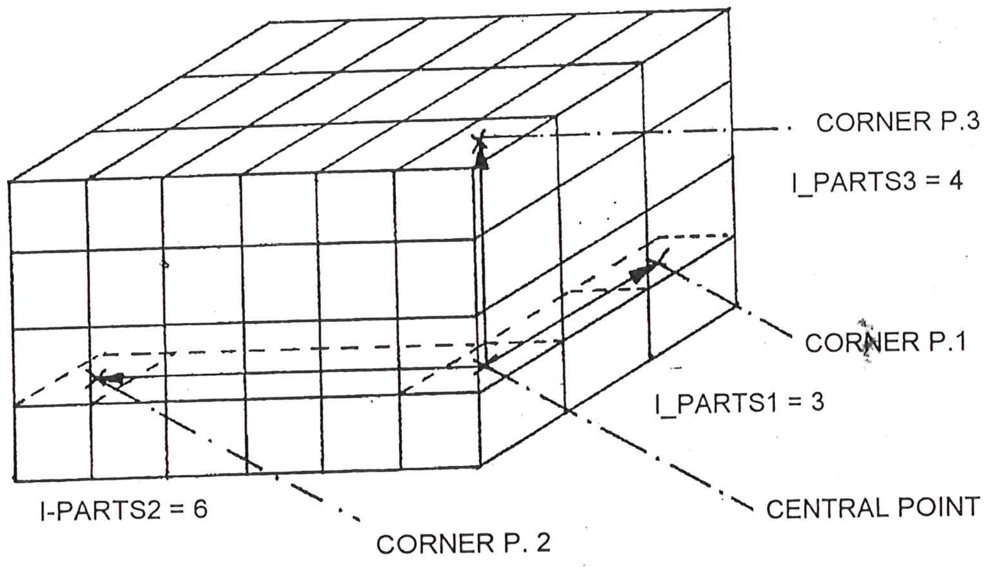
3.) Corner point 2:
3rd position in the palletizing pattern program
With corner point 2 the first layer of the pallet is defined. Further rows are arranged into the direction of the corner point 2 in parallel to the first row.

4.) Corner point 3:
4th position in the palletizing pattern program
This point defines the direction into which the different layers of the pallet are arranged.

5.) Local constant I_PARTS1:
Partscount per row into direction
Central point ---> corner point 1

6.) Local constant I_PARTS2:
Partscount per row into direction
Central point ---> corner point 2

7.) Local constant I_PARTS3:
Partscount per row into direction
Central point ---> corner point 3



Pallet patterns are defined in main programs and therefore can be treated like the remaining main programs.

The command PALLET
The command PALLET is located in the second submenu under the teach pendant key Spec. It has two parameters, the remark concerning the palletizing pattern program name and the command type. The system equates #ON, #OFF and #INIT describe the command type.

With each PALLET command it is checked whether the palletizing pattern program exists and whether it corresponds to the syntax agreed upon.

PALLET <Pal-program>, #INIT
This command initializes the corresponding variables in the palletizing pattern program. The user must ensure that the partscount per palletizing direction (local constants 1...3) and the pallet directions (positions 1...4) will be correctly programmed.

The command **PALLET <Pal-program>, #INIT** calculates the total partscount from the individual partscounts and stores it under the global variable ITOTAL. The global variables ACTVALUE1 ... ACTVALUE3 are allocated with zero and incremented according to the running time.

Delta vectors per palletizing direction (= displacement vector between two parts) are calculated from the corresponding position information and the partscounts and stored in the local vector variables.

The command **PALLET <Pal-program>, #INIT** must be processed at least once before using the command **PALLET <Pal-program>,#ON** since otherwise the variables are invalidated.

PALLET <Pal-program>, #ON
This command calculates from the actual global variables ACTVALUE1...ACTVALUE3 and the local vector variables V_ZP_EP1... V_ZP_EP3 the corresponding palletizing offset vector to the running time.

This palletizing displacement is added to the running time to the following positions. At the same time, the total parts counter ITOTAL is decremented by one. The displacement vector is kept so long until a new command **PALLET <Pal-program>,#ON** will be processed or until it will be deleted again by the command **PALLET <Pal-program>,#OFF**.

PALLET <Pal-program>,#OFF
The pallet displacement is reset again by this command. After #OFF resp. before the next #ON the total parts counter must be tested with the TEST command and with pallet being completely processed a corresponding reaction (pallet change) has to be introduced.

Example programs:

Three-dimensional palletizing:

```
 1 MP                    "PALTEST"
 2 TOOL                  variable:T
 3 MOVE_MODE             #PTP
 4 PTP_SPEED[%]:         80
 5 PTP_ACCEL [%]:        80
 6 PATH_SPEED [mm/s]:    200.0000
 7 PATH_ACCEL [%]:       100
 8 PALLET                Prog_name:"PATTERN1",#INIT
 9 POSITION              <start position>
10 MOVE_MODE             #LINEAR
11 LABEL                 "CYCLE"
12 PALLET                Prog_name:"PATTERN1",#ON
13 POSITION              <over gripping position (central point)>
14 POSITION              <gripping position (central point)>
15 POSITION              <over gripping position (central point)>
16 PALLET                Prog_name:"PATTERN1",#OFF
17 TEST                  #VARIABLE,Op_1:ITOTAL,#>,Op_2:0,label:"CYCLE"
18 STOP
19 END
```

Corresponding palletizing pattern program:

```
 1 MP              "PATTERN1"
 2 C
 3 C Global variables
 4 C ==============
 5 VAR             name:ITOTAL
 6 VAR             name:ACTVALUE1
 7 VAR             name:ACTVALUE2
 8 VAR             name:ACTVALUE3
 9 C
10 C Local vector variables
11 C ==================
12 LOC_VAR         name:V_ZP_EP1
13 LOC_VAR         name:V_ZP_EP2
14 LOC_VAR         name:V_ZP_EP3
15 C
16 C Local constants
17 C ==============
18 LOC_CONST       name:I_PARTS1,value:3
19 LOC_CONST       name:I_PARTS2,value:3
20 LOC_CONST       name:I_PARTS3,value:2
21 C
22 C Pallet positions
23 C ==============
24 TOOL            variable:T
25 POSITION        <central point>
26 POSITION        <corner point1>
27 POSITION        <corner point 2>
28 POSITION        <corner point 3>
29 END
```

Two-dimensional palletizing (single layers):

same as three-dimensional palletizing, but:

in the palletizing pattern program PATTERN1:

```
      .
      .
20   LOC_CONST        I_PARTS3,0
      .
```

One-dimensional palletizing (rows):

same as three-dimensional palletizing, but:

in the palletizing pattern program PATTERN1:

```
      .
      .
19   LOC_CONST        I_PARTS2,0
20   LOC_CONST        I_PARTS3,0
      .
      .
```

Miscellaneous:

- Irregular palletizing patterns like:

```
X   X   X   X   X
  X   X   X   X
 X   X   X   X   X
  X   X   X   X
 X   X   X   X   X
```

require programming of 2 palletizing patterns like:

```
X   X   X   X   X

X   X   X   X   X

X   X   X   X   X
```

and like:

```
X   X   X   X

X   X   X   X
```

- Omitting individual rows:

```
           7   X    X    X    X    X
           6
           5   X    X    X    X    X
ACTVALUE2        4  X    X    X    X    X
  ^        3   X    X    X    X    X
  |        2   X    X    X    X    X
  |        1   X    X    X    X    X
  |        0   X    X    X    X    X
  |
  |            0    1    2    3    4
  |   ----------------------> ACTVALUE1
```

By manipulating the parts' counters actual to the running time it is possible to omit entire rows resp. individual parts' positions.

Corresponding main program:

```
 1 MP                     "PALTEST"
 2 TOOL                   variable:T
 3 MOVE_MODE              #PTP
 4 PTP_SPEED[%]:          80
 5 PTP_ACCEL [%]:         80
 6 PATH_SPEED [mm/s]:     200.0000
 7 PATH_ACCEL [%]:        100
 8 PALLET                 Prog_name:"PATTERN1",#INIT
 9 POSITION               <Start position>
10 MOVE_MODE              #LINEAR
11 LABEL                  "CYCLE"
12 PALLET                 Prog_name:"PATTERN1",#ON
13 POSITION               <over gripping position (central point)>
14 POSITION               <gripping position (central point)>
15 POSITION               <over gripping position (central point)>
16 PALLET                 Prog_name:"PATTERN1",#OFF
17 TEST                   #VARIABLE,Op_1:ACTVALUE2,#<>,Op_2:6,label:"CYCLE"
18 ADD                    Op_1:1,Ziel_Var:ACTVALUE2
19 SUB                    Op_1:5,Ziel_Var:ITOTAL
20 TEST                   #VARIABLE,Op_1:ITOTAL,#>,Op_2:0,label:"CYCLE"
21 STOP
22 END
```

Corresponding palletizing pattern program:

```
 1 MP   "PATTERN1"
 2 C
 3 C Global variables
 4 C ==============
 5 VAR               name:ITOTAL
 6 VAR               name:ACTVALUE1
 7 VAR               name:ACTVALUE2
 8 VAR               name:ACTVALUE3
 9 C
10 C Local vector-variables
11 C ==================
12 LOC_VAR           name:V_ZP_EP1
13 LOC_VAR           name:V_ZP_EP2
14 LOC_VAR           name:V_ZP_EP3
15 C
16 C Local constants
17 C ==============
18 LOC_CONST         name:I_PARTS1,value:5
19 LOC_CONST         name:I_PARTS2,value:8
20 LOC_CONST         name:I_PARTS3,value:0
21 C
22 C Pallet positions
23 C ==============
24 TOOL              variable:T
25 POSITION          <central point>
26 POSITION          <corner point1>
27 POSITION          <corner point 2>
28 POSITION          <corner point 3>
29 END
```

ATTENTION:

If the actual parts' counters are manipulated, also the total parts' counter must be adapted.

If two pallets are simultaneously required in a user program with the same palletizing pattern (e.g. removal from pallet, treatment and deposit onto pallet), this palletizing pattern must be programmed twice. This is required, because the internal delta vectors have different directions.

EXTENSION OF THE PALLETIZING PATTERN

The palletizing pattern program can contain four further constants. These constants are optional, i.e. they need not be programmed. In this case an integer constant and three real constants are concerned.

If programmed, the integer constant is taken as a control constant. The functionalities are entered in bit code.

Bit 0 = 1 ->	The direction vectors defined in the pallet program by the positions are rectangularly set to each other with each command PAL #INIT. In this case, the positions of the corner points 2 and 3 of the pallet program are modified.
Bit 1 = 1 ->	Three real constants follow after the integer constant. These three real constants set the distances between the individual parts' positions into the three dimensions (ZP-EP1, ZP-EP2 and ZP-EP3). Thus, the programmed corner points of the pallet only set the shifting direction.

Remark:
The pallet pattern is only programmed in the correct manner, if either none of these optional constants or if only the integer constant or if all four constants are programmed.

Applications:
If the user knows that his pattern to be palletized is of rectangular shape, programming errors of the corner points can be compensated by setting the bit 0 in the integer variable. The value of the integer variable then must be '1'.

If the user knows the parts' distance, he can enter it into the three real constants. In this case he must program the integer constant with the value '2' and the three real constants with the corresponding parts' distance in mm. If the integer constant is allocated with the value '3', the pallet points are additionally set to each other in a rectangular way.

If unexpected irregularities occur with these measures, calibration of the robot is not exact enough.

Example for a pallet program with extension:

```
 1 MP               "PATTERN1"
 2 C
 3 C Global variables
 4 C ==============
 5 VAR              name:ITOTAL
 6 VAR              name:ACTVALUE1
 7 VAR              name:ACTVALUE2
 8 VAR              name:ACTVALUE3
 9 C
10 C Local vector variables
11 C ==================
12 LOC_VAR          name:V_ZP_EP1
13 LOC_VAR          name:V_ZP_EP2
14 LOC_VAR          name:V_ZP_EP3
15 C
16 C Local constants
17 C ==============
18 LOC_CONST        name:I_PARTS1,value:<number>
19 LOC_CONST        name:I_PARTS2,value:<number>
20 LOC_CONST        name:I_PARTS3,value:<number>
21 C
22 C Pallet positions
23 C ==============
24 TOOL             variable:<T-variable>
25 POSITION         <central point>
26 POSITION         <corner point1>
27 POSITION         <corner point 2>
28 POSITION         <corner point 3>
29 C
30 C Optional constants
31 C =================
32 LOC_CONST        name:I_STEUER,value:<control bits>
33 LOC_CONST        name:R_ABST_ZP_EP1,value:<distance>
34 LOC_CONST        name:R_ABST_ZP_EP2,value:<distance>
35 LOC_CONST        name:R_ABST_ZP_EP3,value:<distance>
36 END
```

Structure of the palletizing pattern with position variables

As already mentioned, pallet programs may also contain position variables instead of the programmed positions. The corresponding variable definitions must be in the same pallet program between the local constant definitions and the tool step.

The number of the variable definitions may either only be = 0 (no position variables used) or = 4. Any other number of definitions causes a syntax error.

In a pallet program with position variables each combination from programmed positions and position variables is allowed for the central point and the corner points.

The four definitions of the position variables have the following significance:

Definition 1: central point
Definition 2: corner point 1
Definition 3: corner point 2
Definition 4: corner point 3

The variable names are freely selectable.

When using global definitions, the programmed pallet patterns can be arbitrarily modified to running time.

The position variables can be inscribed with the command ACT_POS.

Example:

```
 1 MP                  "PATTERN1"
 2 C
 3 C Global variables
 4 C ==============
 5 VAR                 name:ITOTAL
 6 VAR                 name:ACTVALUE1
 7 VAR                 name:ACTVALUE2
 8 VAR                 name:ACTVALUE3
 9 C
10 C Local vector variables
11 C ==================
12 LOC_VAR             name:V_ZP_EP1
13 LOC_VAR             name:V_ZP_EP2
14 LOC_VAR             name:V_ZP_EP3
15 C
16 C Local constants
17 C ==============
18 LOC_CONST           name:I_PARTS1,value:<number>
19 LOC_CONST           name:I_PARTS2,value:<number>
20 LOC_CONST           name:I_PARTS3,value:<number>
21 C
22 C Position variables
23 C ==============
24 VAR                 name:P_ZP
25 VAR                 name:P_EP1
26 VAR                 name:P_EP2
27 VAR                 name:P_EP3
28 C
29 C Pallet positions
30 C ==============
31 TOOL                variable:<T-variable>
32 VAR_POS             variable:P_ZP
33 POSITION            variable:P_EP1
34 POSITION            variable:P_EP2
35 POSITION            variable:P_EP3
36 END
```

DOCUMENTATION

REIS GMBH & CO MASCHINENFABRIK OBERNBURG

General

Title: Errors and messages of the RSV

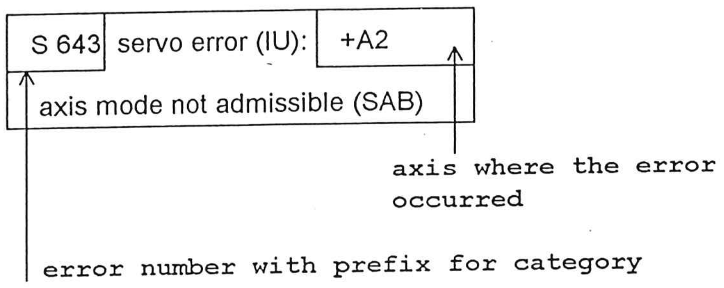
Authors:	Laue / Wetzel
Date:	10.11.98
Control version:	RSV
Software:	02_01

Date:	04.12.98
Release:	Som

1 Regulator error messages

1.1 STRUCTURE OF THE ERROR MESSAGE

Structure of the regulator error message on the teach pendant:


S 643
servo error (IU):
+A2
axis mode not admissible (SAB)
axis where the error occurred
error number with prefix for category

- Servoregulator error (IU): error of the REIS-Interface-Unit
- Servoregulator error (CU): error of the IRT Control-Unit

The error range 641-700 includes the servoregulator errors, i.e., errors recognized on the axis regulators.
The former message regulator error xx thus is omitted from RSV-software version 02-01 on.

1.2 MEANING OF THE ERROR NUMBERS

Written in italics = message text indicated on the teach pendant

error number	meaning	possible cause
643	*axis mode not admissible (SAB)*	An axis mode not admissible for this axis was requested, e.g. passive switching for an axis which must not be released. What axis modes are the admissible ones results from the machine data IAXES_DESCR.
644	*axis mode not admissible (TMS)*	see error number 643
645	*data change not admitted with drives being active*	Machine data was modified with drives being active the modification of which is only allowed with drives being off.
646	*axis speed too high (FIPO)*	The control gives a speed being too high to this axis. The max. speed increase can be adjusted as factor RSPEED_OVERL in MD_PROG . Too high axis speed in CP_mode
647	*tracking distance too long*	The axis could not follow the given nominal value.
648	*monitoring of standstill*	The axis left the given standstill position.

649	*maximum axis speed reached (FIPO)*	The control gives a speed to this axis that cannot be processed by the servoamplifier.
650	*machine data exceeds value range*	A value was entered in MD_PROG that exceeds the admissible range.
651	*disturbed communication to the ADSP*	There is a communication error to the Control-Unit (CU).
652	*new setup-data (node number/ baudrate) are adjusted*	Setup data of the servo were changed.
653	*over- resp. undervoltage:* over- or undervoltage recognized! occurs in conjunction with error 654	see error 654
654	*error of the power pack:* servoamplifier power pack indicates an error!	1. failure of the voltage supply 2. hardware failure
656	*excess temperature power electronics:* the temperature of the servoamplifier is higher than 80 °C!	1. servoamplifier is overloaded 2. fan defective 3. ambient temperature at the control cabinet is too high
657	*I^2*t monitoring ! motor overloaded in the cycle*	1. overload of the axis 2. sluggish mechanics 3. current vibration, unfavorable regulator parameters 4. current limit RMS too low
658	*excess temperature motor*	1. overload of the axis (See error 657) 2. cable break 3. current vibration, unfavorable regulator parameters
659	*ADSP has recognized failure of the Reis processor !* (with regard to the software)!	1. no sufficient grounding of the total installation 2. extremely disturbed environment 3. hardware failure of the servoamplifier
660	*ADSP-Firmware wrong:* servoamplifier software is faulty!	data loss in the FLASH of the CU
661	*ADSP-Parameter faulty:* parameters of the servoamplifier are faulty!	data loss in the FLASH of the CU
662	*Resolver signals faulty:* a safe position actual value generation is not possible!	1. resolver defective 2. problem in the resolver cabling
663	*I_max not adjustable:* the max. motor current adjusted in the machine data RCURRENT_MAX cannot be made available by the servoamplifier.	1. machine data was edited 2. machine data was read in from disk - check 3. servoamplifier was exchanged - check performance data of the servoamplifier

664	*I_max <= 0:* Imax of the servoamplifier is <= 0 ! The max. motor current adjusted in the machine data RCURRENT_MAX is negative and thus invalid.	check machine data
665	*I_max not adjustable:* The motor nominal current adjusted in the machine data RCURRENT_RMS cannot be made available by the ser-voamplifier.	1. machine data was edited 2. Machine data was read in from disk - check 3. servoamplifier was exchanged -Check performance data of the servoamplifier
666	*I_rms <= 0:* Irms of the servoamplifier <= 0! The motor nominal current adjusted in the machine data RCURRENT_RMS is negative and thus invalid.	check machine data
667	*CAN-Watchdog:* A failure of the CAN-bus was recog-nized!	1. Control was stopped (RESET) 2. No sufficient grounding of the total instal-lation 3. Extremely disturbed environment 4. Wiring problem on the CAN-line 5. CAN-bus not correctly terminated (120 ohm at the beginning and end of the bus)
668	*TMS-Watchdog:* A failure of the Reis signal processor was recognized!	1. No sufficient grounding of the total instal-lation 2. Extremely disturbed environment 3. Hardware failure of the servoamplifier
669	*SAB-Watchdog:* A failure of the micro controller was recognized!	1. No sufficient grounding of the total instal-lation 2. Extremely disturbed environment 3. Hardware failure of the servoamplifier
670	*ADSP-Watchdog:* A failure of the IRT signal processor was recognized!	1. No sufficient grounding of the total instal-lation 2. Extremely disturbed environment 3. Hardware failure of the servoamplifier
671	*ADSP Overspeed* *unfavorable regulator adjustment:* 120% of the max. admissible motor speed were reached!	Unfavorable adjustment of the current and speed regulator - reduce reinforcements !
672	*Nominal value filter not adjustable:* Nominal value filter is not adjustable and is limited to maximum or minimum !	1. Machine data IFILTER or IMAN_FILTER was edited 2. Machine data was read in from disk 3. Wrong filter adjustment in the user pro-gram 4. Interpolation cycle was reduced
673	*Cable break in the motor phase:* One or several of the three phases of the motor supply line is faulty!	1. cable break 2. defective contactors do not provide a con-nection to the motor

674	*Nominal position left or not reached:*	1. The given nominal position was not kept
675	*error in the filtered nominal value* : The sum of the filtered position nominal value features a deviation from the unfiltered position nominal value!	1. No sufficient grounding of the total installation 2. Extremely disturbed environment hardware failure of the servoamplifier
676	*error in the cycle relation IPO to the position regulator.* Wrong cycle relation interpolation to position regulator (no rest!)!	1. Machine data IIPO_FACT or IPOS_CNTRL_FACT was edited 2. Machine data was read in from disk - check
677	*reference point angle was modified*: the machine data RHOME_POS was modified! Since reference position is changed by this, the system is set asynchronous!	1. machine data was edited 2. data were read in from disk 3. new servoamplifier was installed
678	*Offset angle was modified:* The machine data IENC_OFFS_HOME was modified! Since reference position was changed by this, the system is set asynchronous!	1. machine data was edited 2. data were read in from disk 3. new servoamplifier was installed
679	*Traversing direction was changed:* The machine data IAXES_DIR was modified! Thus, the traversing direction of the corresponding axis is turned. In order to ensure a safe state, the system is set asynchronous!	1. machine data was edited 2. data were read in from disk 3. new servoamplifier was installed
680	*Checksum error back-up data (AC-Fail):* - system is set asynchronous! The data for position actual value generation are backed up in the switched-off state in the flash of the servoamplifier. During new start of the servoamplifier it is checked whether this data is OK.	1. hardware failure in external power pack of the servoamplifiers 2. wiring problem of the AC-fail line (ext. power pack - servoamplifier) 3. hardware failure on servoamplifier
681	*ADSP-revolution counter not zero after reset:* Revolution counter of the IRT position actual values is not zero - system is set asynchronous! After run-up of the system all revolution counters must be set to zero.	1. No sufficient grounding of the total installation 2. Extremely disturbed environment 3. Hardware failure of the servoamplifier
682	*Resolver position out of tolerance:* The system is set asynchronous! The resolver position changed in switched off condition.	1. Brake of the motor defective - axis slips 2. a big static load moves the axis 3. faulty resolver signals - perhaps in conjunction with error 5
683	*Drives were not switched off:* Drives were still active at the time of switch-off (main switch!) - system is set asynchronous!	1. The drives must be switched off when the installation is switched off 2. Power failure during operation

684	*CU-firmware must be up-dated!*	The servo software has recognized an old IU software for which an up-date cannot be made automatically. -> execute IU-update, read in loader version >= 30 and IU-software >= 2104

2 List of the „path errors“ with remedy measures

Here there are mentioned and commented selected error messages „PATH ERRORS ???“. If other messages occur in conjunction with path errors, the manufacturer must be informed accordingly.

No.	Meaning	Reaction
6	**Programming error:** Path execution has no data from the preparation.	Reduce speed or increase distance of the positions.
102/105	**Programming error:** BAHN_RADIUS = BAHN_DISTANZ 0 mm with activated approximation.	Program BAHN_RADIUS unequal 0 .
504	**Error:** algorithm for position reparameterization divergent.	Acknowledge and check for reproducibility. If error is reproducible, slightly change the speed (via override or command BAHN_GESCHW). In any case **inform the manufacturer.**
505	**Error:** algorithm for orientation reparameterization divergent.	Acknowledge and check for reproducibility. If the error occurs with reversion of orienta-tion, set _IORI_TENS = 0. If this is not the case, **inform the manufac-turer.**
611	**Error** with calculation of the speed profile for TCP-path.	Avoid a possibly programmed BAHN_ZEIT-command. In this case perhaps the machine data RSYNC_WEIGHT, RSMOOTH_WEIGHT, RSYNC_MIN_PHASE were set in the wrong manner. **Inform the manufacturer.**
615 - 619	**Error** in conjunction with start/stop operation	Acknowledge, move robot away and start anew. Check for repeatability. **Inform manufacturer.**

No.	Meaning	Reaction
620	**Programming error:** Approximation together with a path without distance to be traversed and without orientation readjustment (path zero step) is not admitted.	Switch off approximation at the beginning and end of path zero steps or delete point that was programmed twice.
621	**Programming error:** Approximation together with a path without distance to be traversed is not admitted.	Switch off approximation or avoid points with same TCP.
622	**Programming error:** Programming of a BAHN_ZEIT for current path where only orientation must be traversed is missing.	Program BAHN_ZEIT or avoid points with same TCP.
623	**Programming error:** Approximation to the following path where neither distance must be traversed nor orientation must be readjusted, is not admitted.	Switch off approximation at the beginning and end of this path or delete point that was programmed twice.
624	**Programming error:** Approximation to the following path where only orientation must be readjusted, is not admitted.	Switch off approximation at the beginning and end of the following path.
625	**Programming error:** Programming of a BAHN_ZEIT for following path where only orientation must be traversed, is missing.	Program BAHN_ZEIT or avoid points with the same TCP.
706	**Programming error:** In the time given by the path, additional axes cannot reach their end positions without being overloaded.	Increase duration of movement by a smaller path speed (command BAHN_GESCHW) or by higher BAHN_ZEIT, if it is programmed.
810/811	**Error:** Data of the start-up point do not coincide with the data of the path preparation.	Faulty START/STOP cases: move robot out of dangerous area and force step preselection. **Inform manufacturer.**

903	**Programming error** Weighted orientation for two successive points is not allowed.	COPY command in the destination variable _IORI_WEIGHT or remove _IORI_CIRCHP.