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**.

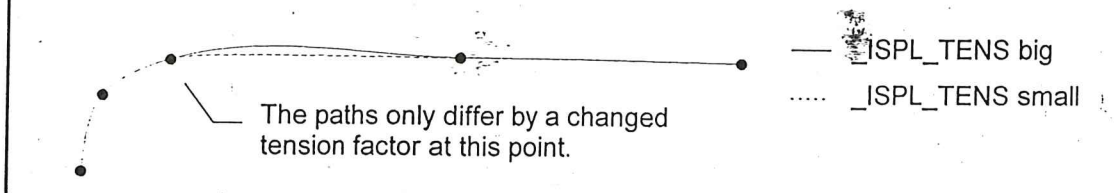
The adjustment of the user commands BAHN_GESCHW and BAHN_BESCHL is immediately clear and won't be mentioned any more in the following.

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%.



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.

TABLE OF CONTENTS

1 GENERAL DESCRIPTION OF CAPACITY 4
1.1 HARDWARE CONDITIONS 4
1.2 CAPACITY 4
2 OPERATING SURFACE/PROGRAMMING UNIT 7
2.1 PROGRAM GENERATION 7
2.1.1 Entering commands 7
2.1.2 Edition of commands 8
2.1.3 Deletion of commands 8
2.2 START PROGRAM 8
2.3 START-UP BEHAVIOR 8
2.4 ADJUSTMENT OF THE MONITORING TIME 9
2.5 TEST PROGRAM 10
3 COMMAND SET 11
3.1 OPERANDS 11
3.2 LOGIC OPERATIONS 12
3.2.1 AND-operation/ANDNOT-operation 12
3.2.2 OR-operation / ORNOT-operation 13
3.2.3 EXCLUSIVE-OR-operation / EXCLUSIVE-ORNOT-operation 14
3.3 SETTING COMMANDS AND RESETTING COMMANDS 15
3.4 LOADING AND TRANSFER COMMANDS 16
3.5 TIMERS 16
3.6 COUNTERS 17
3.7 COMPARISON COMMANDS 17
3.8 ACCUMULATOR COMMANDS 18
3.9 BRANCH COMMANDS 19
3.10 SUBPROGRAM TECHNIQUE 19
3.11 COMMENTS 19
4 COMMUNICATION WITH THE ROBOT CONTROL 20
4.1 SYSTEM VARIABLE _SPS[512] 20
4.2 SYSTEM MARKERS OF THE PLC 20
5 TREATMENT OF ERRORS 21
5.1 EDITING ERRORS 21
5.2 TRANSLATION ERRORS 22
5.3 RUNTIME ERRORS 23
5.4 USER-DEFINED CLEAR TEXT MESSAGES 24
6 SYSTEM VARIABLES AND MACHINE DATA 25
6.1 SYSTEM VARIABLES 25
6.2 MACHINE DATA 25

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.

- 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.

2.1.2 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)

2.5 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.

3.2 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.

3.2.3 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. The result is stored in ACCU1.		(32 bit-operation)

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.

3.4 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.

3.8 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

  K E
  K Z

  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.

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.

5.2 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.

5.4 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.

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:

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

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?)

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*

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

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".

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

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

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.

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.

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.

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

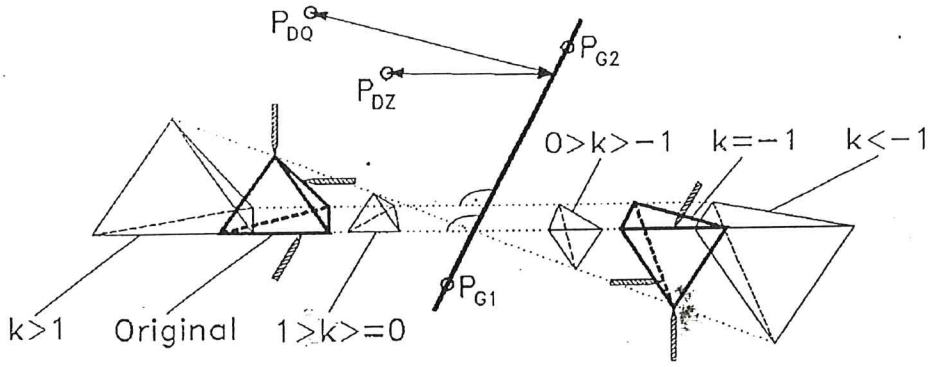
Structure of the reference programs is defined in chapter 3.2 .

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 **(III. 4).** The direction of the positions P_{DQ} and P_{DZ} is not taken into consideration when calculating k.



III. 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 .

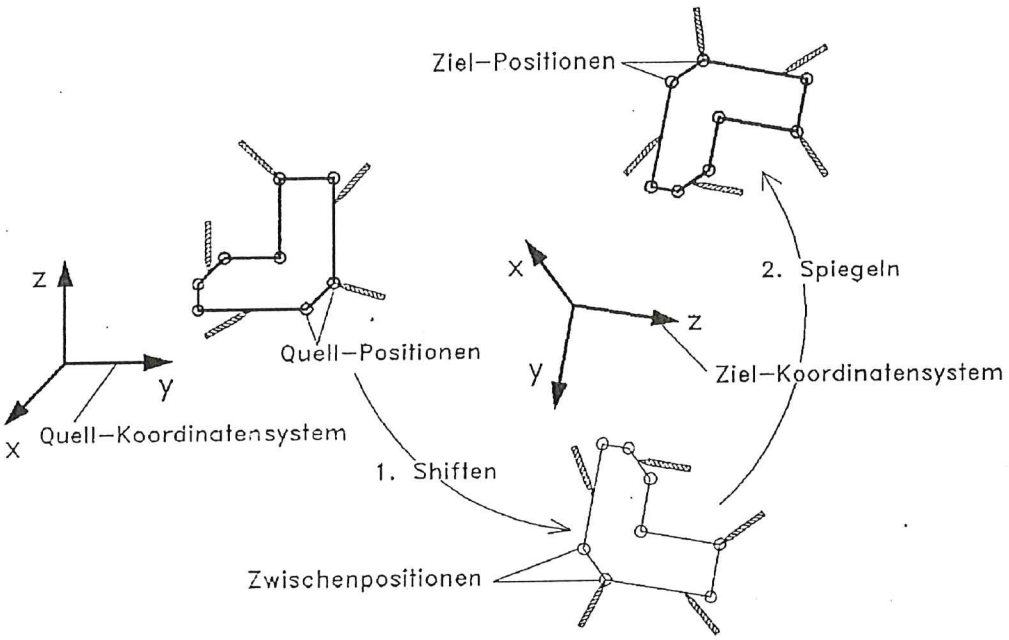
2.6 6D-transformation of user programs with simultaneous mirror inversion

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 **(III. 6).**

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



III. 6: 6D-transformation with mirror inversion

Structure of the reference programs is defined in chapter 3.5 .

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.

2.9 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

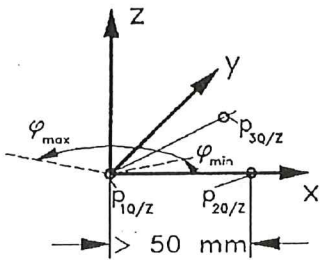
For all transformations executed with the TRAFO_6D command, tool data stored in the position steps of the source program are used.

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.



III. 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).

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).

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.

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).

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" **(III. 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
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
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

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

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

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
```

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**

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.

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.

Description of the palletizing pattern program

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.

Example:

```
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
```

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.

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
.
.
```

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
```

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
```

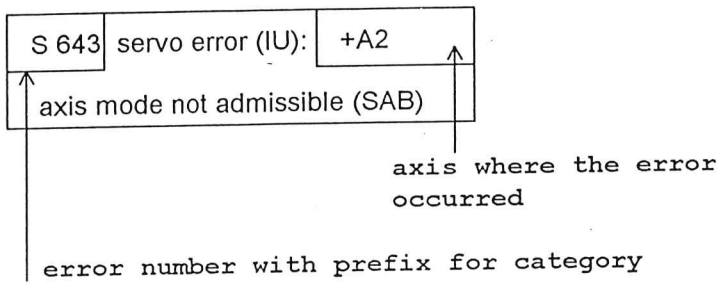
Example:

```
 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
```

1 Regulator error messages

1.1 STRUCTURE OF THE ERROR MESSAGE

Structure of the regulator error message on the teach pendant:



- 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.

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 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
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 recognized!	1. Control was stopped (RESET) 2. No sufficient grounding of the total installation 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 installation 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 installation 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 installation 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 program 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 connection to the motor

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

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.