Kawasaki Robotics

ASTORINO

AS language manual

PREAMBLE

This manual describes the principles and functions of the AS language used in the "ASTORINO" robot and the related "astorino" software, which is part of the scope of delivery.

ASTORINO is a training robot that has been developed specifically for educational institutions. Students can use ASTORINO to try robot-assisted industrial process automation in practice.

This manual is valid from the robot firmware version 3.8.1 and 3.8.1B

If you have further questions, please contact Kawasaki Robotics support.

Contact:

Kawasaki Robotics GmbH

tech-support@kawasakirobot.de

+49 (0) 2131 – 3426 – 1310

TABLE OF CONTENTS

1 AS LANGUAGE

1.1 INTRODUCTION

This manual describes the AS* language used in ASTORINO robot controllers. The purpose of this manual is to provide detailed information about the entire AS language, its basic uses, data types, robot trajectory control, and all commands and instructions for effective use of the language. This manual does not contain robot operating procedures, which are included in the User Manual. Please read this manual and the other instructions listed below.

The use of the robot is only allowed after careful reading and understanding of the instructions.

This manual assumes that the robot has been installed and connected in accordance with the requirements listed in the owner's manual. If you have any questions or concerns about the operation of the robot, please contact Kawasaki EMEA.

The version of the AS language used in ASTORINO robots is a simplified version of the full AS language used in Kawasaki Robotics robots from Kawasaki Heavy Industries Ltd.

The main differences are:

- Most optional expressions in functions are missing,
- System is single threaded.

Note* AS must be pronounced [az].

---

1. This manual cannot be considered as a guarantee for systems in which robots are used. Kawasaki shall also not be liable for any accidents, damages and/or copyright infringements arising from the use of such systems.

2. It is recommended that all employees responsible for starting, programming, servicing or controlling the robot are trained in advance in the courses offered by Kawasaki.

3. Kawasaki reserves the right to make changes, corrections and updates to this manual without prior notice.

4. Keep the instructions in an easily accessible place. In the event of a change of workplace of the robot, transfer to another branch or sales, it is mandatory to attach this instruction to it. If you lose or damage this manual, please contact Kawasaki.

2 CHARACTERISTICS OF THE AS LANGUAGE

In the AS system, the robot is controlled and works according to the program. The program is prepared in advance and describes the tasks to be performed. (A method of reproducing a learned program.)

Instructions available in AS can be divided into two groups: console commands and program instructions.

Console commands: They are entered in the terminal in the astorino or astorinoIDE software after the character (>) and executed immediately.

Some of these commands are also used in programs as program instructions.

Program instructions: used in programs to define robot movements, monitor and control external signals, etc. A program is a set of program instructions.

In this manual, the commands you enter on the screen are called commands, and the program instructions are called instructions.

The simplified AS language has a number of unique features:

1. Two coordinate systems: the global coordinate system based on the robot and the coordinate system of the tool associated with the tool mounted at the end of the arm.

The robot can be programmed in any of these coordinate systems.

2. In teach-in or playback mode, you can move the robot along a linear path. In teach, the tool orientation can be maintained at the same time.

3. Program names can be arbitrary, and their number is limited only by the available robot memory.

4. Each movement sequence can be defined as a program.

5. The robot can be programmed using a personal computer running astorino or astorinoIDE software.

3 AS EXPRESSIONS

This chapter describes the data types and variables used in the simplified AS language.

3.1 NOTATION AND CONVENTIONS

1. Lowercase and uppercase characters

In order to facilitate the comprehension of the text, this manual adopts the following rules for the use of lowercase and uppercase letters. All AS keywords (commands, instructions, etc.) are capitalized. Variables and other data entered are lowercase. Nevertheless, when entering from the AS terminal, the size of letters is not important.

2. Spacebar tab stop

There must be at least one space or tab* between the parameter (or statement) and the parameter. You should also insert a space or tab stop between parameters not separated by a comma or other bounding character. Redundant spaces or tabs are ignored by the system.

Note* A parameter is data required to properly execute a command or other function. For example, for the SPEED command, specify a parameter that specifies the speed of the robot. If a command or function uses multiple parameters, they must be separated by a comma or space.

Example

SPEED 50

3. ENTER key

Most program commands or instructions are executed when you press ENTER. In this statement, the ENTER key is indicated by the symbol ↵.

Numeric values

Values are in decimal unless otherwise noted. Mathematical expressions are used to determine parameter values in on-screen commands and AS instructions. Nevertheless, it is important to keep in mind the limitations on these values. The following are rules for interpreting values in different contexts.

- Distance

It is used to define the length of the robot's movement between two points. The unit of distance is a millimeter (mm); unit is omitted at the time of entry. Both negative and positive values are allowed.

- Angles

They describe the orientation of the tool axis and the angular position using 3 Euler rotation angles (O – A – T) respectively. Both negative and positive values are allowed, and the maximum permissible angle value is 180

- Axis number

The axis number is the total value from 1 to the number of available axes (a standard robot has 6 axes). The axes are numbered sequentially, starting from the base axis. (They are usually referred to as JT1, JT2,.....)

- Signal numbers

Signal numbers are used to identify binary signals (ON/OFF). These are integer values. The acceptable ranges are given in the table below.

	Standard scope
External output signals (TCP MODBUS) Outputs on the robot arm*	1-8 (9-56) 57-58
External input signals (TCP MODBUS) Robot arm inputs*	1001-1008 (1009-1056) 1057-1058
Internal signals	2001-2016

*version B of the astorino robot

A negative signal number indicates the OFF status.

- Keywords

In general, you can use any variable name in AS. Nevertheless, certain words are reserved, such as the names of commands, instructions, etc. and cannot be used to name position data, as variable names, etc.

4 POSITION DATA, NUMERIC AND TEXT DATA

Three types of data are available in the simplified AS system: Position data (point), numeric data (real) and text data (string).

4.1 POSITION DATA

Position data is used to determine the position and orientation of the robot in the workspace. Robot position and orientation refer to the position of the tool center point (TCP) and tool orientation (coordinates), unless otherwise stated. The position and orientation of the robot together determine the position of the robot.

The position determines the place where the robot is located and how it is directed, so when giving motion instructions, both data are determined.

1. The robot moves the tool center point (TCP) to the specified position.

2. The coordinate system of the robot tool is rotated to a specific orientation.

Position data is determined by specifying displacement values in axes or transformation coordinates:

1. Axis displacement values

The position of the robot is determined by specifying the linear or angular displacement in each of the coordinate systems of the robot axis. Values for angular axes are given in degrees, and values for linear axes are given in millimeters. After performing the specified movements in the axes, the position and orientation of the center point of the tool are clearly defined.

Example

The axes numbers are designated as JT1,..., JT6, and the displacement values are given under the axle numbers.

	JT1[°]	JT2[°]	JT3[°]	JT4[°]	JT5[°]	JT6[°]
#pose=	0.00,	33.00,	-15.00,	0.00,	-40.00,	30

2. Transformation values (X,Y,Z,O,A,T)

These values describe coordinate transformations relative to the reference coordinate system. Unless otherwise stated, the transformation values of the tool's coordinate system relative to the robot's underlying coordinate system are given. The position is determined by XYZ values relative to the base coordinate system, and orientation by the Euler angles (O-A-T) of the tool relative to the base coordinate system

Example

X Y Z O A T

	X[mm]	Y[mm]	Z[mm]	O[°]	A[°]	T[°]
pose=	0.00,	1434.00,	300.00,	0.00,	90.00,	70.00

If the robot has more than six axes, the position of the additional axis is given together with the transformation values.

Example

X Y Z O A T JT7

	X[mm]	Y[mm]	Z[mm]	O[°]	A[°]	T[°]	JT7[mm]
pose=	0.00,	1434.00,	300.00,	0.00,	90.00,	70.00,	1000.0

Using axis displacement values and transformation coordinates has some advantages and disadvantages. Choose one of these methods, as appropriate.

	Axis displacement values	Transformation coordinates
Advantages	• High precision of reproduction and no ambiguity of the robot configuration in a given position	• The center of the tool's coordinate system in playback mode does not change even after the tool is replaced. (Zero tool coordinate system offset). • Ability to use relative coordinates (e.g. coordinates in the object system). • Convenience of processing, as the data is given as XYZOAT values.
Disadvantages	• The tool center point (TCP) changes when the tool changes (the zero tool base remains the same). • Unable to use relative coordinates (e.g. item coordinates)	• The coordinates change according to the transformation coordinates of the baseline or tool layout, so full traceability is required to assess the safety impact of any change
Recommended uses	• Defining the starting position in the program • Setting the robot configuration just before or in the position described by the transformation coordinates • Use for other frequently used items	• Describing relative coordinates, such as object coordinates • Describe the position to change using numeric values and the SHIFT function • Describe the position to be changed based on the information provided by the sensor

For the articulation offset variable, define a variable with a name beginning with #.

For a transform value variable, define a variable without the # character.

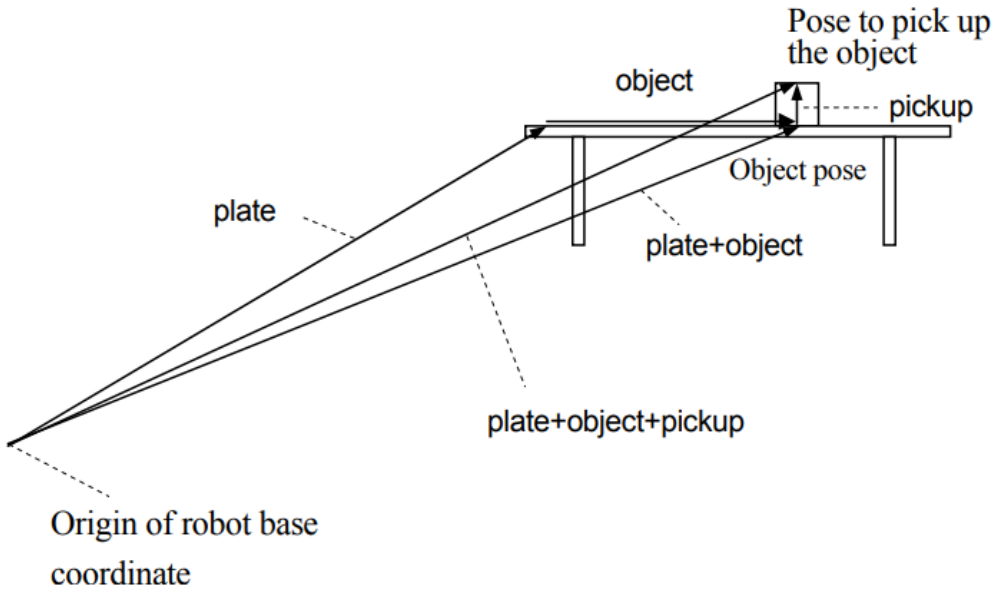
4.1.1 USE COMPLEX TRANSFORMATION VALUES

Transformation values between two coordinates can be expressed as a combination of transformation values between two or more transition coordinates. This can be called the relationship of transformation values or the relative values of the transformation.

For example, suppose "plate" is the name of a variable defined by transformation values relative to the base coordinates that describe the coordinates at the table where the object is located.

Then, if the position of the object relative to the "plate" position is defined as "object", the relationship of the transformation values of the object relative to the coordinates of the robot base can be described as "plate+object".

In the following example, even if the position "plate" changes (e.g. the table moves), only the transformation values for "plate" will need to be changed, and the rest can be used unchanged.



If you repeatedly use the value of a compound transformation, use the POINT command to reduce the time it takes to calculate the value of the compound transformation. For example, to approach the "pickup" position, and then to go to this position, you can type:

POINT x = plate+object+pickup - calculate the target pose

JAPPRO x, 100 - approaching 100 mm above target

LMOVE x - linear movement to the target

[**NOTE**]

Do not change the order in which the relative transformation is expressed. For example, if the transform value of the position variable "b" is defined relative to the transform value of the position variable "a", "a+b" gives the expected result, but "b+a" gives a different result.

For robots with 7 axes, the following points should be noted:

- Using the POINT command, note the JT7 value. For example, in POINT, p=p1+p2, the JT7 value assigned to "p" would be the JT7 value for "p2".
- When assigning a specific value to JT7, add "/7" to the end of POINT. For example, POINT/7 p = TRANS(0,0,0,0,0,0,value) assigns a "value" to the variable "p" as the value of JT7.

4.1.2 USING A FORWARD AND INVERSE KINEMATICS TASK

The AS language allows you to use the forward and inverse kinematics to calculate point variables. The system allows you to convert axes displacement point variables into transformation point variables and works in the opposite direction.

Example:

POINT P0 = #P0 conversion of axes displacement variable into a transformation variable

POINT #P0 = P0 conversion of transformation variable into axes displacement variable

4.2 NUMERICAL DATA

In the AS system, the use of numeric values and expressions is allowed. A numeric expression is a value expressed in numbers, variables, and operators and functions. Numeric expressions are used not only in mathematical calculations, but also as parameters for console commands and program instructions.

For example, the **DRIVE** command requires three parameters, the axis number, the amount of displacement and the speed. These parameters can be specified as numeric values or expressions, as shown in the following example:

DRIVE 3,45,75 Shift of axis No. 3 by angle 45, at a speed of 75%

DRIVE joint, (start+30)/2, 75 If joint=2, start=30, axis 2 will be shifted by +30 at a speed of 75%.

Numeric values in the AS system are divided into three types:

1. Real numbers

Real numbers can be integers or fractions. Added and negative values between −3.4 E+38 and 3.4 E+38 (−3.4×1038 to 3.4×1038) and zero are allowed.

Actual values without fractions are whole values. The allowed range is −16,777,216 to +16,777,215. Integer values are entered in the decimal system. The character separating the integer from the fractional part is the dot "." E.g. 8.5

2. Boolean values

Boolean values can have only two states: TRUE and FALSE. A value of 1.0 is equivalent to TRUE and a value of 0 (or 0.0) is equivalent to FALSE. TRUE and FALSE are reserved words in AS.

Boolean value True = TRUE, 1.0

Boolean value False = FALSE, 0.0

4.3 TEXT EXPRESSIONS

Text expressions can consist of numbers, variables, functions, and other text expressions linked by operators. They can be used as parameters for console commands or instructions. The interpretation of values depends on the context in which the expression is used. The text expression is preceded by the character "***$***", e.g. ***$TEKST = "HELLO WORLD"***

Text expressions can be added, for example ***, $TEST = "HELLO " + "WORLD"***

5 VARIABLES

In simplified AS, you can assign names to position, numeric, and text data. These are called variables. Variables that contain position data are referred to as position variables, respectively, actual. When a variable is defined, it is stored in memory along with the value assigned to it. You can then use such a variable in the program.

5.1 VARIABLE TYPES

5.1.1 GLOBAL VARIABLES

Global variables are seen within the entire robot system, they can be used in the terminal and various programs.

E.g. ***repetitions = 10***

5.1.2 LOCAL VARIABLES

Local variables are seen within only one robot program, they cannot be used in the terminal. The variable is deleted when the program finishes. They are defined by adding a period (".") before the variable name

E.g. ***.num = 10***

5.2 VARIABLE NAMES

Variable names must begin with an alphabetic character and can contain characters, letters, and numbers. The use of both lowercase and uppercase characters is allowed.

The following are examples of disallowed variable names:

3p2a	first character cannot be a number
part#2	The "#" character cannot be used in the middle of a variable name.
random	Word reserved

5.3 DEFINING POSITION VARIABLES

Variables that contain position information are called position variables. An item variable is defined only if its name and value are specified. If you do not specify a value, it remains undefined, and executing the program with an undefined variable generates an error.

In the simplified version of AS, you can use two types of position variables.

- Variables named P1.. P99 and #P1.. #P99
- Variables with any name, e.g. download.

Position variables are a very convenient solution for many reasons:

You can use the same position data many times without reteaching the position each time.
The defined position variable can be used in various programs.
A defined position variable can be used to define another position.
You can enter numeric values directly to determine the position, instead of very time-consuming position teaching.
Item variables can have any name, which increases the readability of programs.

5.4 ARRAY VARIABLES

Variables of all types: numeric, text and point variables can be used as arrays. Arrays are defined by adding the character "[" after the variable name, then specifying the index of the array and ending with the sign "]". Example:

x[0] = 10

x[1] = 12

x[2] = x[0] + x[1]

POINT test[0] = P0

LMOVE test[0]

$text[0] = "Hello"

$text[1] = "World"

5.5 DEFINE POSITIONS USING ON-SCREEN COMMANDS

1. The **HERE** command used in the terminal allows you to save the current position of the robot in a variable position with a given name.
2. The **POINT** command used in the terminal allows you to assign new data to a position variable, whether by function or other position variable.

Example 1: Using axis angle values

The variable name must begin with # to distinguish it from transformation coordinates. After the command, the displacement values in the axes for the current position are displayed:

> ***HERE #pose*** ↲

Example 2: Using transformation coordinates

>***HERE pose*** ↲

- Overwrite the value of an already defined variable

>***POINT pose = P1*** ↲

6 NUMERIC EXPRESSIONS

Numeric expressions can consist of digits, variables, functions, and other numeric expressions connected by operators. All numeric expressions determined by the system are actual values. Numeric expressions can always be specified in place of numeric values. They can be used as parameters for console commands or instructions. The interpretation of values depends on the context in which the expression is used. For example, an expression given in place of a logical value has a Boolean value of False if it evaluates to 0 or True if it evaluates to 1.

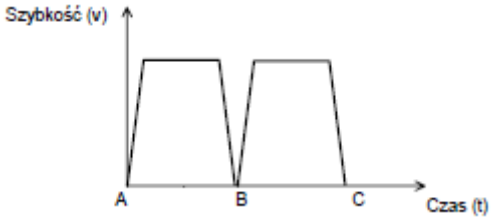
7 OPERATORS

You can use arithmetic and logical operators in expressions. All operators use two values and return one result value. The table below provides a list of operators.

Arithmetic operators	+ – * / ^ MOD	Addition Subtraction or negation Multiplication Division Exponentiation The rest from splitting
Comparison operators	< <=, =< == <> >=, => >	Smaller Less than or equal to Equal Different Greater than or equal to Bigger
Boolean operators	AND NOT OR XOR	Logical AND Logical complement Logical OR Logical conjunction OR

8 ROBOT MOVEMENT

Acceleration for the second segment begins after the execution of the first segment is completed, when the current position is at the target point. The slope of the speed rise is determined by the ACCEL parameter and the braking edge by the DECEL parameter.



Astorino robot can move in three different ways. These ways are called interpolations. We can distinguish:

- Linear interpolation
- Joint interpolation
- Circular interpolation

In an anthropomorphic robot arms (6 axis) there exists some positions that are called singularities. A singular position where problem of structurally uncontrollable position might occur exists when for example JT4 and JT6 are parallel to each other, or JT1 and JT6 are parallel to each other. These configurations return multiple mathematical solution of inverse kinematics and therefore the motion through these points might be unpredictable and introduce a lot of very fast joint movements.

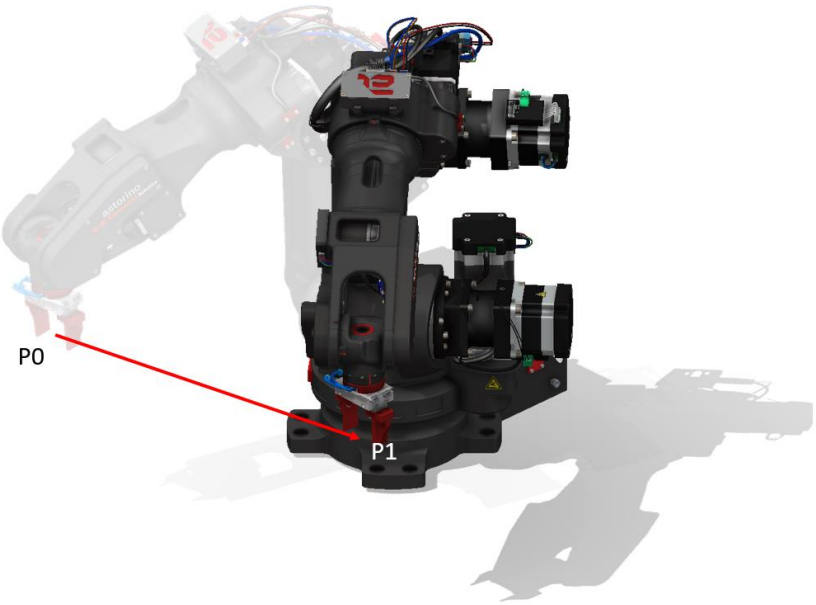
Examples of singular positions

JT4 and JT6 are parallel

JT1 and JT6 are parallel

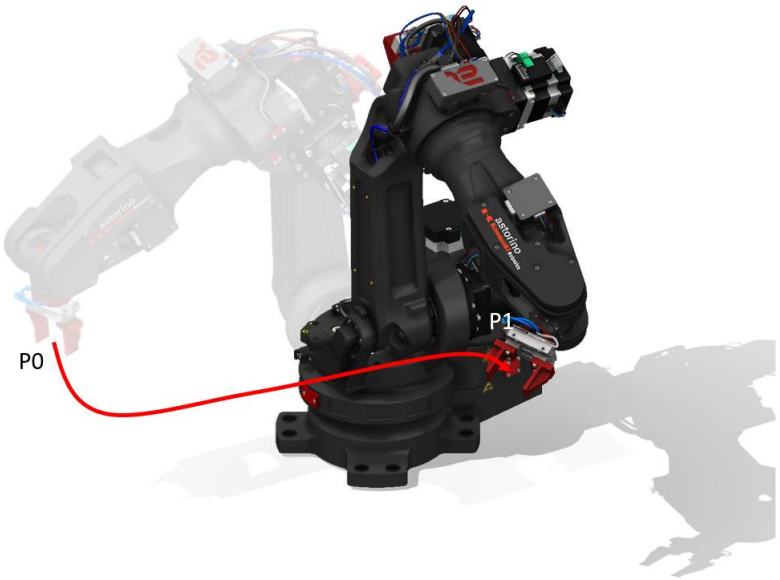
8.1 LINEAR INTERPOLATION

In this type of interpolation robot moves from the current position to the destination in that way that the TCP moves along straight line in a 3D space.



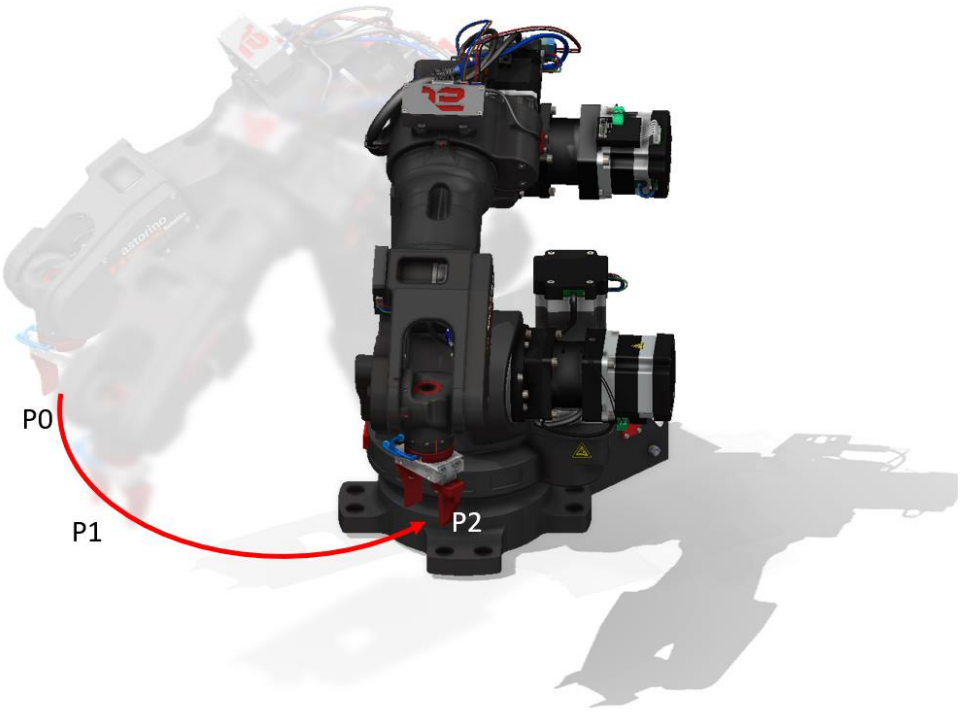
8.2 JOINT INTERPOLATION

In this type of interpolation robot moves form the current position to the destination in that way that all axes end motion at the same time. This movement creates an unpredictable TCP path in a 3D space. This motion in not effected by singularity points.



8.3 CIRCULAR INTERPOLATION

In this type of motion robot moves from the current position to the destination through the middle point in that way that the TCP creates a 3D circular line in a 3D space.



9 PROGRAM EXECUTION FLOW

Program instructions are executed in order from top to bottom of the program.

The **CALL** statement calls and executes another program, but it does not change the flow order. When you execute the **RETURN** statement or the end of the subroutine, processing returns to the calling program and resumes from where it exited.

The **WAIT** statement stops the program before proceeding to the next step until the specified condition is true. The **HOLD** statements stop the program at the stage where the instructions are.

9.1 SUBROUTINES (SUBROUTINES)

A main program can be temporarily paused, and another program, called a subroutine, can be called and executed. By using the subroutine, you can transform the program into a modular structure that is easier to understand. For example, create one program called "**INIT**" that you can then call at the beginning of other programs. In the "**INIT**" program, you can specify speeds, reset signals and variables.

Example:

```
.PROGRAM MAIN
  CALL INIT
  LMOVE P1
.END

.PROGRAM INIT
  SPEED 50 MM/S ALWAYS
  X = 10
  SIGNAL -1
.END
```

10 AS LANGUAGE FEATURES AVAILABLE IN ASTORINO

10.1 COMMANDS TO CONTROL THE PROGRAM

SPEED	Sets the monitoring speed.
PRIME	Prepares the program for execution
EXECUTE	Executes program
HOLD	Stops program execution
CONTINUE	Resumes program execution
DO	Executes a single motion instruction

SPEED *monitoring speed*

Function

Sets the monitoring speed, expressed as a percentage.

Parameter

Monitoring speed

Sets the speed as a percentage. If this parameter is set to 100, the speed will be set to 100% of the maximum speed. If this value is equal to 50, the speed will be set to half the maximum speed.

Explanation

The robot movement speed is the product of the monitoring speed (set with this command) and the program speed (set in the program using the SPEED instruction). For example, if the monitoring speed is set to 50 and the speed in the program is set to 60, the maximum robot speed will be 30%.

WARNING

If the product of the monitoring speed and the speed of the program is higher than 100, the maximum robot speed is automaticallv set to 100%.

The default monitoring speed value is 80%. This command does not affect the speed of the movement that is currently being executed. The newly entered speed is applied when the current movement and scheduled movement are completed.

Example

If the program speed is set to 100%:

>***SPEED*** 30 ↲ The speed of the robot's movement is set to 30 % of the maximum speed.

>***SPEED*** 50 ↲ The robot movement speed is set to 50 % of the maximum speed.

>**SPEED** 100 ↲ The speed of the robot's movement is set to 100 % of the maximum speed.

>***SPEED 200*** ↲ The speed of the robot's movement is set to 100 % of the maximum speed.

PRIME *program name*

Function

It prepares the system in such a way that the program can be executed using the CYCLE START button. This command used alone does not execute the program.

Parameter

Name of the program
Selects the program to be prepared for execution.

Explanation

This command prepares only the system for execution. It does not execute the program. After preparing the system by the PRIME command, the program can be executed using the EXECUTE command. The program can also be executed using the CYCLE START button.

EXECUTE *program name*

Function

Executes the robot program.

Parameter

Name of the program
Selects the program to execute.

Example

>**EXECUTE** test ↲	Executes a program called "test" continuously. (The program executes continuously until it stops, such as using the HOLD command or until an error occurs.)

HOLD

Function

Immediately stops the execution of the robot program.

Explanation

The movement of the robot is immediately stopped. Unlike the use of the EMERGENCY STOP switch, there is no sudden stop of motion.

CONTINUE

Function

Resumes execution of the program stopped by the HOLD statement.

DO *movement instruction*

Function

Executes a single program instruction. (Some program instructions cannot be used with this command.)

Parameter

Movement instruction
Executes the specified program instruction.

Explanation

Program instructions are typically typed into the program and executed as program steps.
Nevertheless, the DO command allows you to execute a single instruction without the need to create a program specifically for this purpose.

Example

>***DO JMOVE*** safe ↲	Transition of the robot to the "safe" position in motion with axial interpolation.
>***DO HOME*** ↲	Moving the robot to the home position in motion with axial interpolation.

10.2 VARIABLE POSITION COMMANDS

HERE	Assigns the current position to the specified variable
POINT	Defines the position variable.
POINT/X	Specifies the X value of the position variable.
POINT/Y	Specifies the Y value of the position variable.
POINT/Z	Specifies the Z value of the position variable.
POINT/OAT	Specifies the OAT values of the position variable.
POINT/O	Specifies the value of O variable position.
POINT/A	Specifies the A value of the position variable.
POINT/T	Specifies the T value of the position variable.
POINT/7	Specifies the value of 7 of the position variable.
POINT/8	Specifies the value of 8 of the position variable (Conveyor 1).
POINT/9	Specifies the value of 9 of the variable position (Conveyor 2).
DECOMPOSE	Allocates elements of the position variable to an array variable.
FRAME	Creates a new coordinate system reference point
NULL	Returns zero transformation
TRANS	Returns the coordinates of the transformation, calculated from the given components
#PPOINT	Returns axis displacement values calculated from the specified elements
SHIFT	Returns the transformation coordinate obtained by moving the original position
RX,RY,RZ	Returns a transformation value expressing rotation about an axis

HERE *position variable*

#HERE *position variable*

Function

This command assigns the current position to the specified position variable. The position can be expressed in transformation values or axis angles.

Parameter

Position variable
The value of a variable can be given using coordinates of transformations or axis shifts.

Explanation

Position can be expressed in transformation values, axis angles, or complex transformation values.

The values of the variable are displayed on the terminal
If you enter a variable by specifying axis angles (the variable name begins with #), the axis values for the current position are displayed. If the variable contains transformation coordinates, XYZ OAT values are displayed. XYZ values determine the location of the endpoint (TCP) of the coordinate system relative to the base coordinate system. OAT values determine the orientation of the tool's coordinate system.

Example

>**HERE** #pick ↲	Assigns the current robot position to the variable "#pick" (axis angles).
>**HERE** place ↲	Assigns the current position of the robot to the variable "place" (transformation coordinates).
>**POINT PICK = HERE** ↲	Assigns the current position of the robot to the variable "pick" (transformation coordinates).

POINT *position variable name=position values*

Function

Allocates position information to the right of "=" to the position variable to the left of "=".

Parameter

1. ***Name of the position variable***

Specifies the name of the position variable to specify (can be specified in axis shift or transform values).

2. ***Position values***

A point variable or function that contains or returns transformation values or angles on axes.

[**NOTE**]

If the types of values to the right and left of the "=" sign differ, this command works as follows:
1. POINT trans=#joint (transformation values = axis angle Values)
The axis offset values on the right are converted to transformation values and assigned to the variable on the left.
2. POINT #joint = trans (axis angle Values = transformation values)
The transformation values on the right are converted to axis angles and assigned to the variable on the left. The transformation values are transformed by the robot in its current configuration.

Example:

POINT TEST = P0

POINT #P0 = TEST

POINT TEST[0] = TRANS(0,0,0,0,0,0,0)

POINT #TEST = #PPOINT(0.90,90,45,0,0)

POINT/X *name of pose variable =position values*

POINT/Y *name of pose variable =position values*

POINT/Z *name of pose variable =position values*

POINT/OAT *name of pose variable =position values*

POINT/O *name of pose variable =position values*

POINT/A *name of pose variable =position values*

POINT/T *name of pose variable =position values*

POINT/7 *name of pose variable=position value*

POINT/8 *name of pose variable=position value*

POINT/9 *name of pose variable=position value*

Function

Allocates the position information to the right of "=" to the position variable to the left of "=".

Parameter

1. ***Name of the pose variable***

Specifies the name of the position variable to be specified (can be specified in axis displacement values, transform values, or complex transformation values).

2. ***Position values***

If not specified, the "=" character is omitted.

Explanation

Allocates only specific elements (X, Y, Z, O, A, T) transformation values.

Example:

POINT/OAT a1 = a2 point a2 values will be assigned to the OAT value of point a1

DECOMPOSE *array variable[item number] = position variable*

Function

Saves each element of the specified position variable as an element of an array variable (X, Y, Z, O, A, T for the transformation value; JT1, JT2, JT3, JT4, JT5, JT6 for axis angle values).

Parameter

1. ***Array variable - name of array variable***

Specifies the name of the array variable in which to store the values of each element.

2. ***Item number***

Specifies the first array index in which to save items from the variable position.

3. ***Position variable***

Specifies the name of the position variable from which to extract each element (transformation values, axis offset values).

Example

DECOMPOSE X[0] = part	Allocates elements of the transformation value "part" to the first 9 elements in the array variable " **x".**
DECOMPOSE ang[4] = #pick	Allocates the elements of the value of the variable of point "#pick" to index number 4 and to the indexes following it in the array variable "ang".

For example, in the statement above, if the values of #pick are equal to 10, 20, 30, 40, 50, 60, then:

ang[4]=10 ang[7]=40

ang[5]=20 ang[8]=50

ang[6]=30 ang[9]=60

FRAME (*pose 1, pose 2, pose 3, pose 4*)

Function

Returns the transformation values of FRAME coordinates relative to BASE coordinates. Note that only the translational transform components of position variables are used to calculate the coordinates of the new FRAME.

Parameter

Value of position variable 1, value of position variable 2

They specify transformation values to determine the direction of the X-axis. The X-axis of the FRAME coordinates is set to pass through these two positions. The positive direction of the X-axis is set from the position determined by the transform values of variable 1 to the position determined by the transformation of the value of variable 2.

Value of position variable 1, value of position variable 3

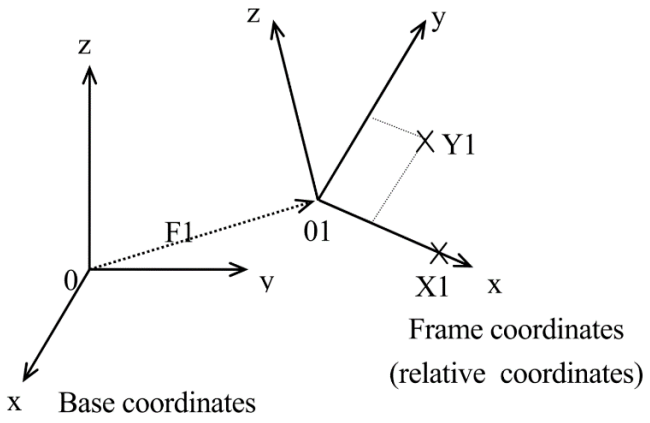
They specify transformation values to determine the direction of the Y axis. The Y-axis of the FRAME coordinates is set to pass through these two positions. The positive direction of the y-axis is set from the position determined by the transform values of variable 1 to the position determined by the transformation of the value of variable 3.

Value of position variable 4

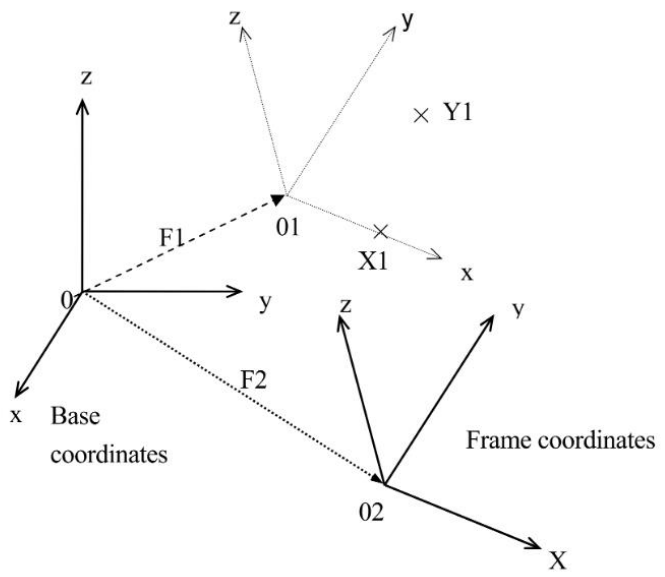
Specifies a transform value variable to determine the origin of FRAME coordinates that equals the X,Y,Z values returned by this function.

Explanation

POINT F1 = FRAME(O1,X1,Y1,O1)



POINT F2 = FRAME(O1,X1,Y1,O2)



REMARK!

Pay attention to the points when defining a new FRAME.

POINT F = FRAME(a1,a2,a3,a1)

The Y and Z axes in the above examples are directed in opposite directions depending on where point a3 is taught. Therefore, if point a3 is taught too close to point a1, this may cause an erroneous calculation of the Z-axis of the system. For a better result of calculations, the points a2 and a3 should be taught far from point a1 (at least 50mm).

NULL

Function

Returns zero point transformation

Explanation

Returns a point whose coordinates are all zeros (Z=Y=Z=0, O=A=T=0)

Example

POINT new = SHIFT(NULL BY 100,0,0)

Dist = DISTANCE(NULL,P1)

TRANS (*var X, var Y, var Z, var O, var A, var T, var JT7*)

Function

Returns the transformation coordinates for the specified components that specify shifts and rotations. The JT7 component may be omitted.

Parameter

Component X, Component Y, Component Z
Components of the translations X, Y, Z. If these parameters are not specified, the values 0 are assumed.
O Component, A Compoment, T Component
Specifies the rotation components O, A, T. If these parameters are not specified, the values 0 are assumed.
JT7 component
Specifies the offset component of the JT7 axis in mm, if this argument is not provided, the values 0 are used.

Explanation

This statement calculates the transformation coordinates based on the values specified by the parameters. New transformation coordinates can be used to define position variables, complex transformation coordinates, or in motion statements. This function is very useful in combination with the DECOMPOSE.

Example

POINT temppos = TRANS(a, b+100,c, d, e, f)

#PPOINT (*JT1, JT2, JT3, JT4, JT5, JT6, JT7*)

Function

Returns axis angles values.

Parameter

JT1, JT2, JT3, JT4, JT5, JT6, JT7
Specifies the value of each angle (in degrees or mm).

Explanation

This function returns axis displacement values. These values represent the displacements in each axis, from axis 1 to the last axis (not necessarily the sixth).

WARNING

The #PPOINT function uses only axis angle Values . For this reason, always at the beginning of this function you should include the prefix "#".

Example

POINT #temp = #PPOINT (0, a, a/2, 0, 0,0)

SHIFT(*transformation variable* BY *var X, var Y, var Z*)

Function

Returns the position obtained by shifting a variable specified by transformation coordinates shifted by a given distance relative to each axis of the BASE coordinate system (X,Y,Z).

Parameter

Transformation coordinate variable
A transformation coordinate variable that specifies the position to be shifted.

X offset, Y offset, Z offset
Offset values in the X, Y, Z axes of the BASE coordinate system.

Explanation

The X, Y, and Z offset values are added to the X, Y, Z displacement components of the specified variable specified by the transformation coordinates. The result is returned as transformation coordinates.

Example

If the variable of the coordinates of the transformation **x** has values (200,150,100,10,20,30), then after executing the statement

POINT y=SHIFT(x BY 5, -5, 10)

The position **X** is moved by the specified values to the position (205,145,110,10,20,30), which is then assigned to the variable **Y**.

RX (*angle*)

RY (*angle*)

RZ (*angle*)

Function

Returns transformation values that represent rotation around the specified axis.

Parameter

Angle
Specifies the rotation value in degrees.

Explanation

X, Y, Z in this function represent the coordinate axes. Returns the rotation value around the specified axis. Translational values are not returned by this function (X, Y, Z = 0).

Example

POINT x_rev =RX(30) *Returns a* transformation value representing a 30° rotation around the X axis and assigns a value to "x_rev".

10.3 SYSTEM MANAGEMENT COMMANDS

TOOL	Displays the TOOL layout currently in use
POINT	Displays point data
SETHOME	Configures the home position.
ERESET	Deletes error
ZZERO	Starts the process of resetting the axle
ZPOWER	Turns drives on or off
CPUTEMP	Displays the current CPU operating temperature
FREE	You knowthe background of currently free RAM
Z_OUTSOURCE	Sets the operating mode of 3.3V outputs
Z_INPULL	Sets the operating mode of the inputs 3.3V
Z_USER	Sets the user access mode
STATUS	Displays the current status of the robot
WHERE	Displays the current position of the robot

TOOL

Function

The command displays the transformation coordinates for the tool, specifying the position of the tool's coordinate system relative to the zero coordinate system.

Example

>***TOOL*** ↲ entered in Terminal will display:

>X[mm]: 0.000 Y[mm]: 0.000 Z[mm]: 40.000
>O[deg]: 0.000 A[deg]: 0.000 T[deg]: 0.000

SETHOME *position*

Function

This command sets and displays the HOME position.

Parameter

Position
Point variable or HERE command (Defines the current position as the HOME position.)

Example

>***SETHOME HERE*** ↲ Configures the current entry as the HOME item.

ERESET

Function

Deletes the error. The operation of this command is identical to pressing the RESET button in astorino or astorinoIDE software.

Explanation

When the ERESET command is executed, a RESET signal is sent. However, this command has no effect if an error occurs at the same time. The function can also be used in abbreviated form by typing "***ERE".***

STATUS

Function

Displays the current status of the robot in the terminal.

WHERE

Function

Displays the current position of the robot in the axial and transformation angles (XYZOAT JT7)

10.4COMMANDS FOR BINARY SIGNALS

RESET	Turns off all external I/O signals
SIGNAL	Turns output signals on (or off)
PULSE	Sets the signal status for a specified time
DLYSIG	Sets the signal status after a specified time
OPENI	Sets predefined signals to a specific state
CLOSEI	Sets predefined signals to a specific state
BITS	Sets the signal group to a specified value (max. 16 signals)

RESET

Function

Turns OFF all external output signals. This command does not affect dedicated signals.

> **WARNING**
>
> **Note that this command disables all signals except those listed above, even in playback mode.**

SIGNAL *signal number1,..,signal number6*

Function

Turns specific external or internal output signals ON or OFF.

Parameter

Signal number1..6
The number of the external output signals or internal signals. A positive number turns ON the signal, and a negative number turns OFF the signal. Up to 6 signal number can be listed at the same time. Not all 6 numbers need to be typed.

Explanation

The signal number determines whether it is external or internal.
Acceptable signal numbers

External output signals	1 – 8
MODBUS	9-56 signals
Signals on the JT3	57-58 axle arm (Robot version B)
Internal signals	2001−2016

External input signals must not be given
If the signal number is a positive number, the signal is turned on; If it is a negative number, the signal is turned off.

Example

>***SIGNAL -***1 ↲	Switching off external output signal 1,
>***SIGNAL -***1,2 ↲	Switching off external output signal 1 and ON output 2
>**SIGNAL** 2005 ↲	Activation of internal signal 2005,
>***SIGNAL -***res ↲	If the variable "res" has a positive value, the output signal determined by this value is set OFF.

PULSE *signal number, time*

Function

Turns ON a specified external output signal or internal signal for a specified period of time.

Parameter

Signal number

Sets the numbers of external output or internal signals. Selecting a dedicated signal causes an error.

Acceptable signal numbers	
External output signal	1 - 58
Internal signal	2001-2016

Time

Sets the signal issuance time (in seconds). If not specified, it will be automatically set to 0.0 seconds.

Example

PULSE 3, 0.4 Turns a 3rd output ON for 0.4s

DLYSIG *signal number, time*

Function

Sends the specified signal after a specified period of time.

Parameter

Signal number

Dials the number of the external output signal or the internal signal. If the signal number is positive, the signal is turned on; If it is negative, the signal is turned off. Selecting a dedicated signal causes an error.

Acceptable signal numbers	
External output signal	1 - actual number of signals
Internal signal	2001-2016

Time

Specifies the delay time in seconds for signal output.

Example

DLYSIG 1, 0.8 Puts signal 1 in the high state after 0.8s from the instruction call.

OPENI

Function

The function works only in version B of the robot, it allows one command to put output 57 in high state and output 58 in low state.

CLOSEI

Function

The function works only in version B of the robot, it allows one command to put output 58 in high state, and output 57 in low state.

BITS *initial signal number, number of signals = decimal value*

Function

Distributes a group of external output signals or internal signals in a binary pattern. Signal states are set as ON/OFF according to the binary equivalent of the specified decimal value.

Parameter

Starting signal number
Specifies the first signal to set the signal state.

Acceptable signal numbers	
External output signal	1 - actual number of signals
Internal signal	2001-2016

Number of signals

Specifies the number of signals to be set to ON/OFF. The maximum number allowed is 16.

Decimal value

of the specified signal number) is issued, and the remaining bits are Specifies the decimal value used to set the desired ON/OFF signal states. The decimal value is converted to binary notation, and each bit of the binary value sets the state of the signal, starting with the least significant bit. If the binary notation of this value has more bits than the number of signals, then only the state of the given number of signals (starting with ignored.

Example

BITS 9.16 = 12082 Issues a 9 to 16-bit value of 12082 on signals

10.5 COMMANDS FOR PROGRAMS AND DATA

DELETE	Removes a program or point from the robot's memory
DELETE/P	Removes the program from the robot's memory
DELETE/L	Deletes a point from the robot's memory

DELETE *program name/point name*

DELETE/P *program name*

DELETE/L *point name*

Function

Deletes specified data from the robot's memory

Parameters

/P – program name, /L point name

Example:

> ***DELETE TEST***↲ Removes the test program, if the test program is not found, it tries to delete a point variable named test

> ***DELETE/L P1***↲ Removes point P1 from robot memory.

10.6 COMMANDS TO DISPLAY MESSAGES

PRINT	Displays data in a terminal
TYPE	Displays numeric type data in a terminal

PRINT *data to display*

Function

Displays the desired data on the terminal.

Parameter

Data to display
Select one or more of the following options.
(1) String, e.g. ***PRINT $TST***
(2) Actual type expression (value is first calculated and then displayed, e.g. ***PRINT SIN(30)***

Example

In this example, the value of the real variable "i" is equal to 5
>***PRINT and*** ↲
The display should look like this:
>***5:00 a.m***.

>***PRINT "HELLO"***
>***HELLO***

>***$TEMP = "MY TEXT"*** ↲
>**PRINT $TEMP**↲
>***MY TEXT***

TYPE *data to display*

Function

Displays numeric type data (numbers) on the terminal.

Parameter

Data to display
Select one or more of the following options.
(1) String, e.g. ***TYPE TST***
(2) Actual type expression (the value is first calculated and then displayed, e.g. ***TYPE SIN(30)***

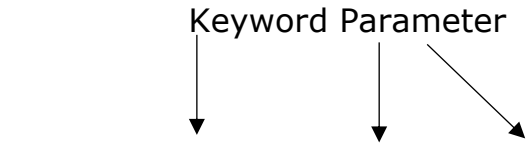
Example

In this example, the value of the real variable "i" is equal to 5
>***TYPE i*** ↲
The display should look like this:
>***5.00***

11 PROGRAM INSTRUCTIONS

A program statement consists of a keyword expressing the command and a parameter(s), as shown in the example below.

Example



LAPPRO *position variable, distance*

The keyword and parameter must be separated by a space.

11.1 MOVEMENT INSTRUCTIONS

TOOL	Defines the pose of TCP
JMOVE	Robot movement with joint interpolation
LMOVE	Robot movement with linear interpolation
XMOVE	Linear interpolation robot movement with interruption
DELAY	Stops the movement of the robot for a specified period of time
BRAKE	Stops the movement of the robot
JAPPRO	Approach the destination with joint interpolation
LAPPRO	Approach the destination with linear interpolation
JDEPART	Leaves the current position with joint interpolation
LDEPART	Leaves the current position with linear interpolation
HOME	Moves to the home position
DRIVE	Moves in the direction of a single axis
DRAW	Move a specified distance in the direction of the X, Y, Z axis of the global coordinate system
TDRAW	Move a specified distance in the X, Y,Z direction of the tool's coordinate system
C1MOVE	Circular interpolation movement
C2MOVE	Circular interpolation movement
ALIGN	Aligns the robot orientation (TOOL z-axis) to the nearest BASE axis

TOOL *layout number tool*

TOOL *transform value variable*

Function

This command defines transformation coordinates for the tool, specifying the position of the tool's coordinate system relative to the zero coordinate system.

Parameter

Layout number tool
One of 3 saved tool layouts.
Transformation value variable
A point variable containing **X,Y,Z**, and **O,A,T** data for tool shifts and rotations

Value

1
2
3
4 – values reserved for the program tool calls

Explanation

If a value from 1 to 3 is specified as a parameter, the coordinates of the transformation of the tool's coordinate system are set to values stored in the robot's memory. The value is reserved for the TOOL system called from within and is specified as a point variable. The zero coordinate system of the tool has a center on the tool mounting flange and the axes are parallel to the axis of the last kinematic pair of the robot. During system initialization, the coordinates of the tool transformation are automatically set to zero.
After the robot moves to the setting defined by the transformation coordinates or by manual control in the base system or the coordinate system of the tools, the system automatically calculates the position of the robot taking into account the entered coordinates of the tool transformation.

Parameter 4 is defined for the situation when the TOOL system data is downloaded from the robot program:

POINT T1 = TRANS(0,0,100,0,0,0)
TOOL T1

Example

>TOOL **1** ↲ Changes the position of the tool's coordinate system to the position specified in the TOOL tab.

JMOVE *position variable*

LMOVE *position variable*

Function

Movement of the robot to a specific position.
JMOVE: Motion with joint interpolation.
LMOVE: Motion with linear interpolation.

Parameter

Position variable
Specifies the target position of the robot.

Explanation

The JMOVE instruction causes the robot to move with joint interpolation. The robot moves in such a way that the ratio of the distance travelled to the total distance remains equal for all axes, throughout the duration of the movement, from the starting position to the end position. When executing the LMOVE instruction, the robot moves with linear interpolation. The starting point of the tool's coordinate system (TCP) moves along a linear path.

Example

JMOVE #pick - Movement with axial interpolation to the position determined by the displacement values of the "#pick" axis.
LMOVE P1 Motion with linear interpolation to a position determined by the coordinates of the transformation P1.
LMOVE #pick Motion with linear interpolation to the position determined by the values of the "#pick" axis offsets.

XMOVE *position variable* TILL *signal number*

Function

Movement of the robot to a certain position until the receipt of a signal is met.

Parameter

Position variable
Specifies the target position of the robot.
Signal number

Explanation

The JMOVE instruction causes the robot to move with axial interpolation. The robot moves in such a way that the ratio of the distance travelled to the total distance remains equal for all axes, throughout the duration of the movement, from the starting position to the end position. When executing the LMOVE instruction, the robot moves with linear interpolation. The starting point of the tool's coordinate system (TCP) moves along a linear path.

Example

JMOVE #pick - Movement with axial interpolation to the position determined by the displacement values of the "#pick" axis.
LMOVE P1 Motion with linear interpolation to a position determined by the coordinates of the transformation P1.
LMOVE #pick Motion with linear interpolation to the position determined by the values of the "#pick" axis offsets.

DELAY *time*

Function

It stops the movement of the robot for a specified period of time.

Parameter

Time - Specifies in seconds the time to stop the robot's movement.

Explanation

In AS, the DELAY instruction is treated as a motion instruction that "does not change position".
Even if the robot's movement is stopped by the DELAY instruction, all program steps before the next motion statement are performed before stopping.

Example

DELAY 2.5 Stops the robot movement for 2.5 seconds.

JAPPRO *position variable, distance*

LAPPRO *variable position , distance*

Function

Movement in the direction of the Z-axis of the tool at a given distance from the learned position.
JAPPRO: Joint interpolation motion.
LAPPRO: Motion with linear interpolation.

Parameter

Position variable
Specifies the end position (in transformation values or axis shift values).
Distance
Specifies the distance (in millimeters) between the end position and the actual position obtained by the robot in the direction of the Z-axis of the tool's coordinate system. If the specified distance is positive, the robot moves in the negative direction of the Z tool axis.

Explanation

In these commands, the tool is oriented in a specific position, and the position is set at a specified distance from the given position in the direction of the Z-axis of the tool's coordinate system.

Example

JAPPRO place,100
Movement with joint interpolation to the position 100 mm from the "place" position, in the direction of the Z axis of the tool coordinate system. The position "place" is given using the coordinates of the transformation.

LAPPRO place, offset
Motion with linear interpolation to a position away from the "place" position, given by the coordinates of the transformation, by the distance specified by the variable "offset", in the direction of the Z-axis of the tool's coordinate system.

BRAKE

Function

Stops the robot motion

JDEPART *distance*

LDEPART *distance*

Function

Movement of the robot to a position away from the current position by a specified distance, along the Z-axis of the tool's coordinate system.
JDEPART: Motion with joint interpolation.
LDEPART : Motion with linear interpolation.

Parameter

Distance
Specifies the distance, in millimeters, between the current position and the target position in the direction of the Z-axis of the tool's coordinate system. If the specified distance is positive, the robot moves "backwards", or in other words, in the direction of the negative values of the Z tool axis.

Example

JDEPART 80
Robot movement with joint interpolation back by 80 mm in the direction of the −Z axis of the tool coordinate system.

LDEPART 2*offset
Robot movement with linear interpolation, at a distance of 2∗offset (200 mm if offset = 100) in the direction of the axis − Z of the tool's coordinate system.

HOME

Function

Movement with joint interpolation to a position defined as HOME.

Example

HOME Movement with joint interpolation to the home position specified by the SETHOME command/instruction or in the astorino or astorinoIDE software.

DRIVE *axis number, offset, speed*

Function

Movement of one robot axis.

Parameter

Axis number
The number of the axis to be moved. (For robots equipped with six axes, they are numbered from 1 to 6, starting with the axis furthest from the tool mounting flange.)
Offset
The amount of axis displacements, given as a positive or negative value.
This value is expressed in the same units as the description of the position of the axis, i.e. if the axis is a rotary axis, this value is expressed in degrees (°), and if the axis is a traverse axis, this value is expressed in a unit of distance (mm).
Speed
Speed of movement. As with the standard movement speed, it is expressed as a percentage of the maximum speed.

Explanation

This instruction moves only one axis.
The monitoring speed of this manual is the combination of the speed given in this manual and monitoring speeds. The program speed set in the program does not affect this statement.

Example

DRIVE 2,-10.75 Offset axis 2 (JT2) from its current position by −10° with a speed of 75%.

DRAW *X offset, Y offset, Z travel, X rotation, Y rotation, Z* rotation

TDRAW X travel, Y travel, *Z travel, X rotation, Y rotation, Z* rotation

Function

Movement of the robot with linear interpolation, at a specific speed, from the current position to a given distance in the direction of the X, Y, Z axis and rotation by a given angle around each axis. The DRAW statement moves the robot in the global coordinate system, and the TDRAW statement moves the robot in the coordinate system of the tool.

Parameter

X offset
The amount of the X-axis offset expressed in mm. If this parameter is not specified, a value of 0 mm is entered.
Y offset
The amount of displacement in the Y axis expressed in mm. If this parameter is not specified, a value of 0 mm is entered.
Z offset
The amount of displacement in the Z axis expressed in mm. If this parameter is not specified, the value 0 mm is entered.
Rotation around X
The angle of rotation about the X axis, expressed in degrees. The permissible range of values is less than ±180°. If this parameter is not specified, the value of 0 degrees is assumed.
Rotation around Y
The angle of rotation about the Y axis expressed in degrees. The permissible range of values is less than ±180°. If this parameter is not specified, the value of 0 degrees is assumed.
Rotation around Z
The angle of rotation about the Z axis expressed in degrees. The permissible range of values is less than ±180°. If this parameter is not specified, the value of 0 degrees is assumed.

Explanation

The robot moves in a linear motion from the current position to the set position.

Example

DRAW 50,0,-30	Robot movement with linear interpolation of 50 mm in the direction of the X axis and by –30 mm in the direction of the Z axis of the global coordinate system.
TDRAW 0,0,50	Robot movement with linear interpolation of 50 mm in the direction of the tool Z-axis.

C1MOVE *position variable*

C2MOVE *position variable*

Function

Movement of the robot to a specific position with circular interpolation.

Parameter

Position variable
Specifies the target position of the robot's movement. (Can be expressed in transformation coordinates, complex transformation values, joint values, or position functions)

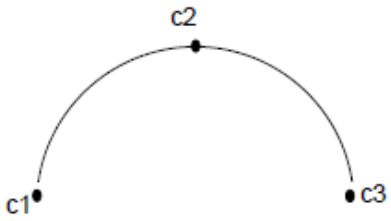
Explanation

The C1MOVE instruction moves the robot to a point halfway on the circular path, and the C2MOVE instruction moves to the end of the circular path.
To move the robot with circular interpolation, it is necessary to learn three positions. These three items are different for C1MOVE and C2MOVE.

1. Position from the last movement instruction or current position.
2. The position that will be used as the parameter of the C1MOVE statement.
3. The position of the next C2MOVE movement instruction.

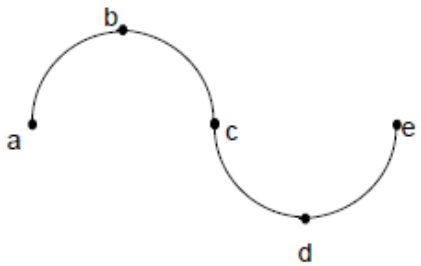
Przykład

```
JMOVE   c1
C1MOVE c2
C2MOVE c3
```



The robot moves in joint interpolated motion to c1 and then moves in a circular interpolated motion following the arc created by c1, c2, c3.

```
JMOVE   #a
  C1MOVE #b   ] arc a,b,c
  C2MOVE #c
C1MOVE #d     ] arc c,d,e
  C2MOVE #e
```



ALIGN

Function

Moves the robot that the Z-axis of the tool's coordinate system so that it is parallel to the nearest axis of the base coordinates.

Explanation

If you want the direction of movement to be aligned along the Z direction of the tool, the **DO ALIGN** function allows you to easily align the direction of the tool to the base coordinates before training the points.

11.2 MOVEMENT INSTRUCTIONS IN COOPERATION WITH THE CONVEYOR BELT

CVLMOVE	Linear motion in cooperation with conveyor belt
CVLAPPRO	Approaches the target point linearly in cooperation with the conveyor belt
CVLDEPART	Leaves the current position linearly in cooperation with the conveyor belt
CVDELAY	Stops the movement of the robot for a specified period of time in cooperation with the conveyor belt
CVWAIT	Stops the robot until the conveyor reaches the set value
CVRESET	Overwrites the current position of the conveyor belt
CVPOS	Reads the current position of the conveyor 1
CVPOS2	Reads current position of the conveyor 2
CVCOOPJT	Turns on robot cooperation with conveyor 1 or 2

CVLMOVE *position variable*

Function

The movement of the robot to a specific position in linear interpolation with synchronization with the conveyor.

Parameter

Position variable

Specifies the target position of the robot's movement. (It can be in transformation values, composite transformation values, or joint values.)

Explanation

TCP follows a linear trajectory from the start to the end position, synchronizing with the conveyor.

Example

CVLMOVE #pick	Linear motion to the defined by the values of the joints angles (#pick) during synchronization with the conveyor.
CVLMOVE Place	Linear motion to the position defined by the transformation variable "place" when synchronizing with the conveyor.

CVLAPPRO *position variable, distance*

Function

Movement in linear interpolation to a specified distance from a certain position, synchronizing with the conveyor.

Parameter

Position variable

Specifies the target position (in transformation values or axis angles)

Distance

Determines the distance in the direction of the tool's Z axis between the target position specified above and the position that the robot actually achieves. (Unit: mm)

Providing a positive distance value moves the robot away from the target position (negative direction of the tool's Z-axis). Entering a negative value moves the robot towards the target position (positive direction of the tool's Z-axis).

Explanation

In this manual, the orientation of the tool in the position in which the robot actually reaches is determined by the position variable given. The position of the

tool becomes a position away from the specified position by a specified distance in the positive or negative direction of the Z axis of the tool.

Example

CVLAPPRO Place,100 The robot synchronizes with the conveyor and moves in linear interpolation to a position 100 mm away in the direction of the tool Z axis from the defined by the values of the "Place" transformation.

CVLDEPART distance

Function

Movement in linear interpolation to a specified distance from the current position, synchronizing with the conveyor.

Parameter

Distance

Determines the distance in the direction of the tool's Z axis between the current position and the position that the robot actually achieves. (Unit: mm)

Providing a positive distance value moves the robot away from its current position (negative direction of the tool's Z-axis). Entering a negative value moves the robot towards the current position (positive direction of the Z axis of the tool).

Explanation

In this manual, the orientation of the tool in the position in which the robot actually reaches is determined by the current position of the robot. The position of the tool becomes a position away from the current position by a specified distance in the positive or negative direction of the tool Z-axis.

Example

CVLDEPART 100 The robot synchronizes with the conveyor and moves in linear interpolation to a position 100 mm away in the direction of the Z-axis of the tool from the current position.

CVDELAY *time*

Function

It "stops" the movement of the robot as seen from the conveyor reference view point for a specified period of time.

Parameter

Time

Specifies the amount of time during which the robot must remain "still" as seen from the conveyor's perspective (Unit: seconds)

Explanation

The **CVDELAY** instruction is a robot movement instruction. After following this instruction, the movement of the robot is controlled so as to maintain the same position relative to the moving conveyor, and therefore, when viewed from the conveyor side, the robot seems to stop. (Looking from externally, the robot moves in accordance with the conveyor, so that the same position is maintained in relation to the workpiece/workpiece on the conveyor).

Example

CVDELAY 2.5 The robot remains stationary for 2.5 seconds from the conveyor's perspective.

CVWAIT *starting position*

Function

Pauses the execution of the program until the conveyor reaches the specified position. (Unit: mm) .

Parameter

Starting position

The robot starts again when the conveyor reaches this position.

Explanation

When this instruction is executed in the program, the robot stops and waits (does not perform the next step in the program) until the conveyor reaches the specified position. The robot resumes work when the conveyor reaches a certain position.

Example

CVWAIT 50 Further execution of the program is suspended until the 50mm conveyor belt position is reached.

CVRESET *axis number*

Function

Resets the position value of the currently cooperating conveyor belt.

Parameter

Axis number

Specifies the conveyor number (axis number) which needs to be reset. This function overwrites the value of conveyor to 0. Possible parameter values are 8 (conveyor 1) and 9 (conveyor 2)

Explanation

The CVRESET statement resets the value of a specified conveyor

Example

CVRESET 8 Resets the Conveyor 1 to 0mm

CVPOS

Function

Assigns the current position of the first conveyor to a variable.

Example

CONV1 = CVPOS Assigns to the variable CONV1 the current value of the position of the first conveyor

CVPOS2

Function

Assigns the current position of the second conveyor to a variable.

Example

CONV2 = CVPOS2 Assigns to the variable **CONV2** the current value of the position of the second conveyor

CVCOOPJT *axis number*

Function

Enables the robot to cooperate with the conveyor belt.

Parameter

Axis number

Specifies which robot conveyor belt is currently to work with. Possible values 8 and 9

Explanation

If the function has not been used, the robot cooperates with conveyor number one.

Example

CVCOOPJT 8	Activates the cooperation of the robot with the first conveyor

11.3 SPEED AND ACCURACY CONTROL INSTRUCTIONS

SPEED	Sets the speed of movement (program speed)
ACCEL	Sets acceleration
DECEL	Sets the deceleration

SPEED *speed*

SPEED *speed* ALWAYS

Function

Determines the speed of the robot's movement.

Parameter

Speed
Specifies the speed of the program. It is usually given as a percentage between 0.01 and 100 (%). You can also specify an absolute value by adding units: MM/S
ALWAYS
The speed will apply to each subsequent motion command until the next SPEED command

Explanation

The actual robot movement speed is the product of the monitoring speed and the motion speed set with this instruction (monitoring speed x program speed). Nevertheless, full speed cannot be guaranteed in the following cases:
1. if the distance between two learned positions is too small
2. If a linear interpolation movement that exceeds max. angular velocity of the axis.
The speed of motion is determined differently in motion with joint interpolation and motion with linear interpolation. In the case of joint interpolation, the speed of movement is defined as a percentage of the maximum speed of each axis. In linear interpolation motion, the speed of motion is defined as a percentage of the maximum speed at the starting point of the tool's coordinate system. If the speed is given in a unit distance per unit of time, the linear motion velocity at the starting point of the tool's coordinate system is configured. When moving with axial interpolation, the speed is set as a percentage. (Even if the speed is set as an absolute value or movement time, the robot will not move at this speed. The speed will be processed as a percentage of the maximum speed.)
The absolute speed is expressed in mm/s. A decrease in monitoring speed shall result in a proportional reduction in these speeds.

WARNING

Even if the product of the program speed and the speed set by the SPEED command exceeds 100%, the actual speed of the movement does not exceed 100%.

Example

If the speed is set as given below and the monitoring speed is 100%:

SPEED 50 Sets the speed of the next movement to 50 % of the maximum speed.

SPEED 100 Sets the speed of the next movement to 100 % of the maximum speed.

SPEED 200	Sets the speed of the subsequent movement to 100 % of the maximum speed (a speed exceeding 100 % is read as 100 %).
SPEED 20 MM/S	The starting point speed of the tool coordinate system (TCP) is set to 20 mm/sec (if the monitoring speed is 100 %).
SPEED 100 MM/S ALWAYS	The starting point speed of the tool coordinate system (TCP) is set to 100 mm/sec (if the monitoring speed is 100 %), any movement after this command will be performed at a speed of 100 MM/s or adequate for the junction movements.

ACCEL value

ACCEL value ALWAYS

DECEL value

DECEL value ALWAYS

Function

Sets the acceleration (or deceleration) of the robot's movement.

Parameter

Value
Sets the acceleration or deceleration of the robot's movement as a percentage of maximum acceleration. The acceptable range of values is from 0.01 to 100. Values outside this range are treated as 100, and values below the range are treated as 0.01.
ALWAYS
The set acceleration/deceleration will apply to each subsequent motion command until the next ACCEL/DECEL command

Explanation

The ACCEL manual sets the acceleration at the moment of starting the robot's movement as a percentage of the maximum acceleration. The DECEL manual sets the deceleration at the end of the robot movement as a percentage of maximum deceleration.

Example

ACCEL 80 ALWAYS	Acceleration is set to 80% for all movements following this instruction.
DECEL 50	The deceleration for subsequent driving instructions is set to 50%.

11.4 PROGRAM CONTROL INSTRUCTIONS

TWAIT	Pauses the execution of the program until the specified time has elapsed
CALL	Calls a subroutine
RETURN	Comes out of a subroutine

TWAIT time

Function

Pauses the program until the specified amount of time has elapsed.

Parameter

Time
Specifies the time in seconds to suspend program execution.

Explanation

This instruction suspends the execution of the program until the specified time has elapsed.

Example

TWAIT 0.5	Wait for 0.5 seconds.
TWAIT delta	Wait until the time specified by the variable "delta" has elapsed.

CALL *program name*

Function

Pauses the current program and jumps to the new program (subroutine). When the execution of the subroutine is complete, processing returns to the original program and follows the step of the CALL statement.

Parameters

Name of the program

Specifies the name of the subroutine to be invoked

Explanation

This statement temporarily pauses the execution of the current program and jumps to the first step of the specified subroutine.

WARNING

In the astorino robot, it is possible to nest the program up to a maximum of 5 degrees.

You cannot CALL parent programs, recursion is not allowed

Example:

CALL INIT

RETURN

Function

Completes the execution of the subroutine and returns to the step after the CALL statement in the program that called the subroutine.

Explanation

This statement completes the subroutine and returns to the program that called the subroutine. If the subroutine is not called from another program (for example, if the subroutine is executed by the EXECUTE command), the program will exit.

At the end of the subroutine, the execution of the program reverts to the original program, even if no RETURN statement exists. However, the RETURN statement should be written as the last subroutine statement (or wherever you want the subroutine to end).

11.5 PROGRAM STRUCTURE INSTRUCTIONS

IF..... THEN... ELSE..... END

WHILE..... DO..... END

DO..... UNTIL

FOR..... END

CASE... OF .... VALUE ... ANY .... END

IF Logical expression THEN

Program instructions(1)

ELSE

Program Instructions(2)

END

Function

Performs one of the groups of program steps, depending on the value of the logical expression.

Parameter

Logical expression
A logical expression or expression based on real values. This expression returns True (1) or False (0).
Program Instructions (1)
Program instructions executed if the logical expression has a Boolean value of TRUE.
Program Instructions (2)
Program instructions executed if the logical expression has a Boolean value of FALSE.

Explanation

This statement executes one of two groups of statements, depending on the value of the logical expression. Here's how to do it:
1. Calculate the value of the logical expression and go to step 4 if the expression is 0 (FALSE).
2. Calculate the value of the logical expression and proceed to execute the program statement (1) if the expression is set to 1 (TRUE).
3. Go to 5.
4. If there is an ELSE statement, follow the program instructions (2).
5. Continue the program from the step after the keyword END.

WARNING

1. The ELSE and END statements must be the only keywords in the lines in which they are located.

2. The IF... THEN must end with the keyword END.

Example

In the example program below, if n is greater than 5, the program speed is set to 10%, and if not, then to 20%.

```
IF n>5 THEN
        sp=10
ELSE
        sp=20
END
SPEED sp
```

The program below first checks the value of the variable "m". If the variable "m" is different from 0, the program checks the external input signal 1001 and displays the corresponding message, depending on the status of this signal. In this example, the external IF structure does not have an ELSE statement.

```
IF m THEN
        IF SIG(1001) THEN
                PRINT 1.0
        ELSE
                PRINT 0.0
        END
END
```

***WHILE* condition DO**

Program instructions

END

Function

If the specified condition is TRUE, the program instructions are executed. If the condition is FALSE, the WHILE statement is omitted.

Parameter

Condition
A logical expression or expression based on real values. Checks what value the expression is: TRUE (1) or FALSE (0).
Program instructions
The group of statements to be executed if the condition is TRUE.

Explanation

This structure repeats specific program steps all the time when the condition is True. The following is the procedure for execution:
1. Calculate the value of the Boolean expression and go to step 4 if the expression is 0 (FALSE).
2. Calculate the logical expression and execute the program instructions if the expression has a value of 1 (TRUE).
3. Go to 1.
4. Continue the program from the step after the keyword END.

WARNING

Unlike DO statements, if the expression is FALSE, steps inside the WHILE construct are not executed. When you use this statement, the condition may change from TRUE to FALSE.

Example

In the example below, the input signal 1001 is monitored and the movement of the robot is stopped depending on its condition. If the signal coming from the parts feeder changes to 0 (the feeder is empty), the robot stops and the execution continues from step after the END statement (step 27 in this example).
If one of the feeders is empty when the WHILE structure starts (external signal input = 0), no step of the structure is performed, and processing continues from step after the word END.

```
WHILE SIG(1001) TO
        LMOVE P1
        TWAIT 1
        LMOVE part
END
```

DO

Program instructions

UNTIL *logical expression*

Function

Creates an iterative DO loop

Parameter

Program instructions
These statements are repeated all the time when the Boolean expression is FALSE.
Logical expression
A logical expression or expression based on real values. If this logical expression changes the value to TRUE, the instructions inside the loop are no longer executed.

Explanation

This structure executes a group of program instructions if the specified condition (logical expression) is FALSE.
Here's how to do it:
1. Follow the program instructions.
2. Check the value of the Boolean expression and if it is FALSE, repeat step 1.

If the Boolean expression is TRUE, go to step 3.
3. Continue executing the program from the step after the UNTIL statement.
The execution of the DO structure is completed when the value of the logical expression changes from FALSE to TRUE.

WARNING

Unlike WHILE constructions, DO construction instructions are always executed at least once. When the value of this expression changes to TRUE, the loop is exited and the step placed after the DO construct is moved.

The DO construction must always end with the UNTIL keyword.

Example

In the example below, the DO construct performs the following task: the part is lifted and transferred to the buffer. After filling the buffer, the binary input signal "buffor" is switched on. Activating this signal forces the robot to stop and move on to another task.

```
DO
        JMOVE get
        JMOVE put
UNTIL (SIG(buffer))
```

FOR *control variable* = *initial* value TO *final value* STEP *step*

Program instructions

END

Function

Repeat execution of program instructions.

Parameter

Control variable
This variable is set to the initial value at the very beginning and is then incremented by 1 after each loop execution.
Initial value
Value or expression. Use this parameter to configure the initial value of the variable that controls the loop.

Final value
Value or expression. This value is compared to the current value of the loop control variable, and when it reaches this value, the loop exit occurs.

STEP

The step by which the control variable is changed

Explanation

This construction repeats the execution of program instructions between the keywords FOR and END. The loop control variable is magnified by the specified step value each time the loop is executed.

Here's how to do it:
1. Assign the initial value to the loop control variable.
2. Calculate the final value and the step value.
3. Compare the value of the loop control variable with the final value.
a. If the step value is positive and the loop control variable is greater than the final value, go to step 7.
b. If the step value is negative and the loop control variable is less than the final value, go to step 7.
Otherwise, go to step 4.
4. Follow the program instructions after the FOR keyword.
5. When you get to the END statement, add the step value to the loop control variable.
6. Return to step 3.
7. Follow the program instructions after the END statement. (Control variable value loop at the time of comparison in step 3 does not change.)

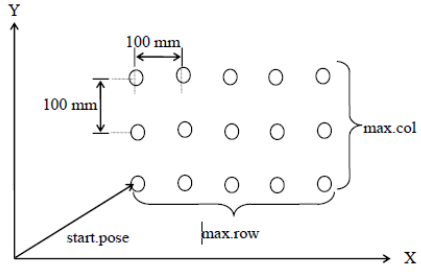
WARNING

Each FOR keyword must have a corresponding END keyword. Note that if the loop control variable is greater than the final value when the values are first compared, none of the program instructions between the FOR and END keywords will be executed.

Example

The "pick" program picks up the part and places it in the "hole" position. The parts are placed as shown in the figure below. (The pallet is placed parallel to the X and Y axes of the global coordinate system, and the distance between the parts is 100 mm.)

```
FOR row = 1 TO maxrow
        POINT hole = SHIFT (startpose BY(row-1) * 100, 0, 0)
        FOR col = 1 TO maxcol
                JMOVE pick
                POINT hole = SHIFT (hole BY 0, 100, 0)
        END
END
```



CASE *control variable* OF

VALUE *value 1,...* :

Program instructions

VALUE *value 2, value 3,...* :

Program instructions

.

.

VALUE *value n, ...* :

Program instructions

ANY :

Program instructions

END

Function

Executes the program according to the specific value of the control variable

Parameter

Control variable
A variable or real value. Its value determines which VALUE case will run.

Explanation

This structure allows the program to choose from several groups and execute instructions. It is a powerful tool in the AS language that provides a convenient method that allows several alternatives in the program.

The procedure for implementation is as follows:

1. Checks the value of the control variable entered after the **CASE** statement.

2. It checks the **VALUE** steps and finds the first step which contains an equal value of the value of the control variable.

3. Follows step-by-step instructions.

4. After executing the program instructions, it goes to the **END** statement.

5. Parameter **ANY** can be omitted

If there is no value that matches the control variable, executes the program instructions under ANY statements. If there is no **ANY** statement and no value is found, none of the steps in **CASE** will be performed.

Example

The following program is executed according to these 3 cases:

1. If the value is an even number between 0 and 10
2. If the value is an odd number between 1 and 9
3. If the value is a number other than the above.

```
CASE x OF
VALUE 0,2,4,6,8,10:
PRINT "The number x is EVEN"
VALUE 1,3,5,7,9:
PRINT "The number x is ODD"
ANY :
PRINT "The number x is larger than 10"
END
```

11.6 INSTRUCTIONS FOR BINARY SIGNALS

SWAIT Hangs the program until the specified signal state condition is set

SWAIT signal number

Function

Waits until the specified external input signal or internal signal has the desired state.

Parameter

Signal number
The number of the external input signal or internal signal to be monitored.
A negative number means that the condition is met if the signal is off.
Acceptable signal numbers

External input signals	1001-1008
Signals MODBUS	1009-1056
Signals on axis arm JT3	1057-1058 (Robot version B)
Internal signals	2001−2016

Example

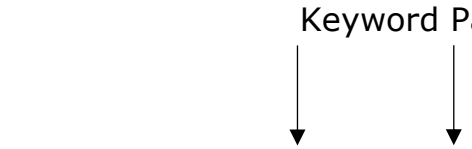
SWAIT 1001 Expects external first input signal to be turned on.

SWAIT -2001 Waiting for the internal signal 2001 to be switched off.

12 FUNCTIONS

This chapter describes the functions used in AS. These functions are typically used in conjunction with on-screen commands and program instructions. The following is the format of the function. The keyword specifies the function, and the parameters entered in parentheses specify the values on which the function operates.

Example



SIG *signal number*

The keyword and parameter must be separated by a space.

12.1 FUNCTIONS THAT OPERATE ON REAL VALUES

SIG	Performs a logical AND operation on the specified signal
BITS	Returns the bit pattern of the signal (up to 16 signals).
DISTANCE	Returns the distance between two points
DEXT	Returns the specified point component
DX,DY,DZ	Returns the displacement value (X,Y,Z)
TRUE	Returns logical one (1.0)
FALSE	Returns logical zero (0.0)
VAL	Returns the numeric value from the string variable
INT	Returns the integer portion of the specified number
EXISTREAL	Returns whether a given numeric variable exists
EXISTJOINT	Returns whether a given position connector variable exists
EXISTTRANS	Returns whether a given transformation variable exists
EXISTCOM	Returns whether there is data to read from serial communication buffer (Serial)
INRANGE	Returns whether a position is within working range
ROUND	Returns the rounded value of a number
TIMER	Sets/Returns the current value of the timer

SIG(*signal number*)

Function

Performs a logical AND operation on the specified signal.

Parameter

Signal number
Numbers of external or internal input signals.

Explanation

Performs a logical AND operation on the specified signal and returns the resulting value. If the signal is Boolean TRUE, it returns Boolean 1 (True). Otherwise, 0 (FALSE) is returned. External or internal input signals are specified by the corresponding numbers as shown below.
Acceptable signal numbers

External input signals	1001 – 1008
Signals MODBUS	1009-1056
Signals on axis arm JT3	1057-1058 (Robot version B)
Internal signals	2001−2016

Positive number signals are assumed to be TRUE when ON, while negative number signals are TRUE when OFF.

Example

If the binary input signal 1001=ON, 1004=ON, 2004=OFF, then:
SIG(1001) == TRUE
SIG(2004) == FALSE
SIG(−1004) == FALSE

BITS(*Starting Signal Number, Number of Signals*)

Function

Reads consecutive binary signals and returns the decimal value corresponding to the bit patterns of the specified binary signals.

Parameter

Starting signal number
Specifies the first signal to read.
Number of signals
Specifies the number of signals to read. The maximum accepted number is 16.
Explanation
This function returns the decimal value corresponding to the bit pattern of the specified signals.
In binary expression of the value returned by this function, the least significant bit corresponds to the initial number of the signal.

Acceptable signal numbers	
External output signal	1 - actual number of signals
External input signal	1001 - actual number of signals
Internal signal	2001-2016

Example
If the signal states are as follows, the result of the following expression will be 5.

x = BITS(1003.4)

The logical values of the 4 signals starting with 1003 (i.e. 1003, 1004, 1005 and 1006) are read as Bit pattern 0101 or 5 in decimal notation.

Signals:	1008	1007	**1006**	**1005**	**1004**	**1003**	1002	1001
Status:	1	1	**0**	**1**	**0**	**1**	0	0

DISTANCE(pose *1, pose 2*)

Function

Calculates the distance between two items, which are expressed in transformation values.

Parameter

Pose 1, Pose 2
Specifies the names of the two transform value variables for which you want to calculate the distance between them.

Explanation

Returns the distance between two positions in millimeters. (Both items can be entered in any order)

Example

k=DISTANCE(HERE,part)
Calculates the distance between the current TCP and the "part" orientation and converts the result to k.

DEXT(*position variable, item number*)

Function

Returns the specified item of the specified item.

Parameter

Position variable
Specifies the name of the pose variable defined by transformation values or junction values.

Item number

Specifies the element to be returned in real numbers, as shown in the following figure.

Item number	Position	
	Transformation values	Axis values
1	X	JT1
2	Y	JT2
3	Z	JT3
4	O	JT4
5	A	JT5
6	T	JT6
7	JT7	JT7

Example
If the transformation values for "aa" are (0, 0, 0, -160, 0, 0, 300), then using this function:

type DEXT(aa, 7)

It will print in the 300 terminal the value of JT7 of this point.

DX(*pose*)

DY(*pose*)

DZ(*pose)*

Function

Returns the transformation values (X, Y, Z) of a position defined by the specified position variable.

Parameter

Pose
Specifies the name of the transformation value variable whose X, Y, or Z component is required.

Explanation

Each of these three functions returns the X, Y, or Z component of the specified position.

Each component of transformation values can also be obtained by using the DECOMPOSE statement. The O, A, and T values are obtained using the DECOMPOSE statement.

Example

If "start" has transformation values:

X	Y	Z	O	A	T
125.00	250.00	-50.0	135.00	50.00	75.00

This:
x=DX(start) DX returns **x = 125.00**
y=DY(start) DY returns **y = 250.00**
z=DZ(start) The function returns **z = -50.00**

TRUE

Function

Returns logical one (1.0)

Explanation

This function is convenient when you need to specify a logical condition TRUE.

FALSE

Function

Returns logical zero (0.0)

Explanation

This function is convenient when you need to specify a logical condition FALSE.

VAL(*$string variable*)

Function

Returns the actual value in the specified string.

Parameter

$String variable
Specifies a string, string variable, or string expression.

Example

DATA = VAL("123") Returns the actual value of 123.

INT(*numeric expression*)

Function

Returns the integer of the specified numeric expression.

Parameter

Numeric expression
Number or variable

Explanation
Returns an integer (that is, the left side of a decimal point)
A negative sign remains with an integer unless an integer is 0.

Example

INT(0.123)	Returns 0.
INT(10.8)	Returns 10.
INT(-5.462)	-5 is returned.

EXISTREAL(*real variable*)

Function

Checks whether the specified variable exists.

Parameter

Real variable
Specifies a string actual variable.

Explanation

If the variable name exists, it returns 1. If it does not exist, returns 0.

Example

ret=EXISTREAL(pp)	If there is a real variable pp, ret=-1. If not, ret=0.

EXISTJOINT(*variable connector name*)

Function

Checks whether the specified connector variable exists.

Parameter

Variable connector name
Specifies the name of the connector displacement variable as a string. Start the name with the # sign.

Explanation

If the variable exists, it returns 1. If it does not exist, returns 0.

Example

ret=EXISTJOINT("#pos")	If there is a variable #pos, ret= 1. If not, ret=0.

EXISTTRANS(*transformation value variable*)

Function

Checks whether the specified variable is defined by transformation values.

Parameter

Transformation value variable
Specifies the name of the transform value variable as a string.

Explanation

If the variable exists, it returns 1. If it does not exist, returns 0.

Example

ret=EXISTTRANS(pos1)

If the value variable of the pos1 transformation defined by the transformation then the values exist, ret=1. If not, ret=0.

EXISTCOM

Function

Checks for input in the serial communication buffer.

Explanation

If there is input in the serial communication buffer, TRUE (1.0) is returned, if the data does not exist, FALSE (0.0) is returned. The function is useful when the robot has to wait for data received from an external device.

Example

```
WHILE (EXISTCOM == FALSE)
        TWAIT 0.1
END
```

The above example loops the program until the data enters the serial communication buffer.

INRANGE(*position variable*)

Function

Checks if the position is within the robot's range of movement and returns a result-dependent value (1.0 or 0.0).

Parameter

Position variable
Specifies which position to check.

ROUND(*numeric value*)

Function

Returns a value rounded to the first decimal place.

Parameter

Numerical value
This value is rounded to the first decimal place.

Explanation

Returns a value rounded to the first decimal place of the value specified as a parameter. When the specified value is a negative value, the value is rounded as an absolute value, and then a negative character is added. The character of the numeric value specified as a parameter remains unchanged unless the result is 0.

Example

ROUND (0.123)	Returns 0.
ROUND (10.8)	Returns 11.
ROUND (-5.462)	Returns -5.
ROUND (-5. 662)	Returns -6.

TIMER(*numeric value*)

TIMER numeric value = value

Function

Sets/Returns a value of the specific timer in seconds.

Parameter

Numerical value
Specifies which timer to read/write. Acceptable numbers are 0 to 10.
Value
Time in seconds written to timer.

Explanation

By using the TIMER function the timer value can be read at any time. Read and returns the time (in seconds) elapsed from the value previously set by TIMER instruction. If no value has been set by the TIMER instruction, the value of TIMER returns the time that elapsed from turning on power.

When used without brackets (), the timer is immediately set to the specific time.

Example

TIMER 1 = 0	Sets Timer 1 to 0.
LMOVE P1	Executes motion.
PRINT TIMER(1)	Prints elapsed time of motion.

12.2 MATHEMATICAL FUNCTIONS

ABS	Returns the absolute value from a numeric expression.
SIN	Returns the sine value of a numeric expression
COS	Returns the cosine value of a numeric expression
ATAN2	Returns arctangents from a numeric expression
PI	Returns PI
SQRT	Returns the square root of a numeric expression
RANDOM	Returns a random value from 0.0 to 1.0

ABS(*Value*)

Function

Returns the absolute value from a numeric expression.

Example

X = ABS(Y)

SIN(*Value*)

Function

Returns the sine value of a numeric expression

Example

X = SIN(Y)

COS(*Value*)

Function

Returns the cosine value of a numeric expression

Example

X = COS(Y)

ATAN2(*value1, value2*)

Function

Returns the arctangents value of a numeric expression

Explanation

$$\mathrm{ATAN2} = \begin{cases} \tan^{-1}(dy/dx) & dx > 0,\ dy > 0 \\ \tan^{-1}(dy/dx)+\pi & dx > 0,\ dy < 0 \\ \tan^{-1}(dy/dx)-\pi & dx < 0,\ dy < 0 \\ +\pi/2 & dx > 0,\ dy = 0 \\ -\pi/2 & dx < 0,\ dy = 0 \\ \text{undefined} & dx = 0,\ dy = 0 \end{cases}$$

Example

X = ATAN2(valueY,valueX)

PI

Function

Returns PI

Example

D = 2*PI*R

SQRT(*value*)

Function

Returns the square root of a numeric expression

Example

X = SQRT(Y)

RANDOM

Function

Returns a pseudorandom value from 0.0 to 1.0

Example

X = RANDOM * 10

12.3 STRING FUNCTIONS

$DECODE	Extracts characters separated by specified characters (char).
$ENCODE	Returns a string created from an actual value.

$DECODE ($*string variable, $separator*)

Function

Extracts characters separated by specific characters

Parameter

$string variable

Specifies the string from which the characters are retrieved. Characters extracted as a result of this function are removed from this string.

$separator

Specifies the character to read as a separator. (You can specify any character in the ASCII array as a separator.)

All characters starting with the first character of the string variable to the separator are returned. The returned string is removed from the string variable. The separator is removed from the string variable.

Explanation

This function looks up the separator character in the specified string and extracts the characters from the beginning of the string to the separator. Extracted characters are returned as a result of the function, while being removed from the original string.

Example

The following example breaks the variable of the string $TEMP into three separate variables **$VAL1**, **$VAL2** , **$VAL3** so that **$VAL*1*** = "123", **$VAL2** = "*456", **$VAL3 =** "789".* The ***VAL*** function then replaces these string variables with the actual value type ***DATAX, DATAY***, ***DATAZ.*** The numeric value data is then used to create a **TEST** point

```
$TEMP = “123/456/789/”

$COMMAND = $DECODE($TEMP,"/")

$VAL1 = $DECODE($TEMP,"/")
$VAL2 = $DECODE($TEMP,"/")
$VAL3 = $DECODE($TEMP,"/")
DATAX = VAL($VAL1)
DATAY = VAL($VAL2)
DATAZ = VAL($VAL3)
POINT TEST = TRANS(DATAX,DATAY,DATAZ,180,0,90)
LMOVE TEST
```

$ENCODE (*value*)

Function

Returns a string based on the actual value. The string is created in the same way as when you use the **TYPE/PRINT** command

Parameter

value

Numeric value

Explanation

This feature allows you to create strings in programs using the actual value.

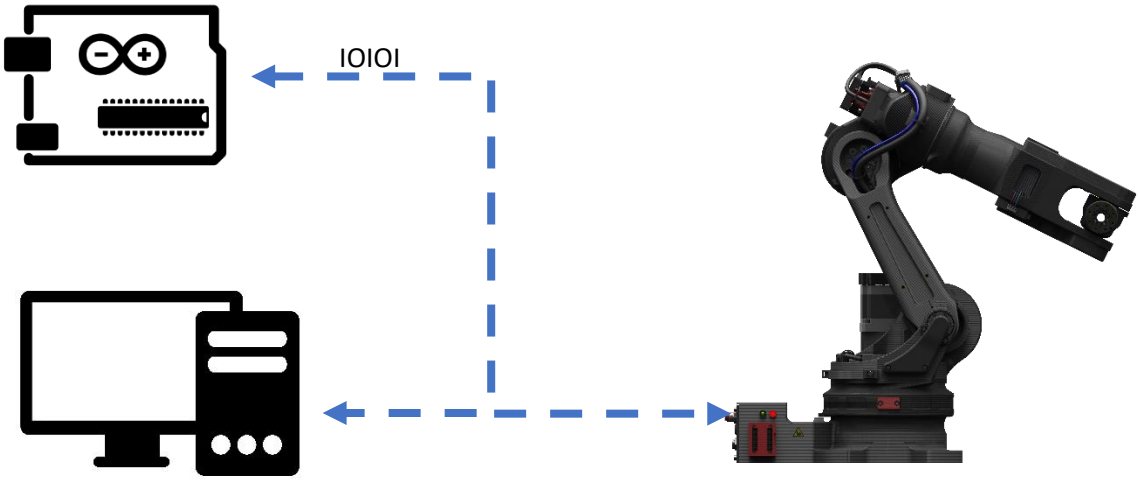
Example

$output = $output + $ENCODE(count)

The value of the real variable "count" is converted to a string and added to the end of "$output". The combined string is then replaced back in the "$output" variable.

12.4 SERIAL COMMUNICATION

Serial communication is used to transfer data between the robot and external equipment such as microcontrollers (e.g. Arduino, ESP32), peripheral equipment (e.g. OpenMV, sensors) or a PC or e.g. Raspberry PI.



REMARK

Serial communication operates at 3.3V, please use 3.3V compatible electronics or voltage level converters.

5V voltage can damage the main CPU chip!

[NOTE]

The parameters of serial communication are:

- Baudrate: 115200
- Dara size: 8
- Parity: None
- Handshake: OFF

Serial communication can also be used to exchange data between accessories using other types of serial communication, such as RS232 and RS485, or between an astorino robot and a PC. The table below shows the required accessory equipment for data exchange.

RS232	**UART (TTL) Converter 3.3V -> RS232**
RS485	**UART (TTL) Converter 3.3V -> RS485**
PC	**UART (TTL) Converter 3.3V -> USB**

12.5 SERIAL COMMUNICATION FUNCTIONS

SEND	Sends data via serial communication (Serial/UART)
RECEIVE	Receives data received from serial communication (Serial/UART)
EXISTCOM	Returns whether there is data to be read from serial communication (Serial/UART)

SEND $*string variable*

Function

It sends data through serial communication.

Explanation

When a function is called, data is sent via serial communication (Serial/UART)

Example

$data = "Hello"
SEND $data

[**NOTE**]

The parameters of serial communication are:

- Baudrate: 115200
- Date size: 8
- Parity: None
- Handshake: OFF

RECEIVE

Function

Receives data received from serial communication (Serial/UART).

Explanation

When a function is called, the available data from the serial communication buffer is retrieved. If there is no available data in the data buffer, a **5s Timeout** is activated, if no data appears in the buffer within **5 seconds** of the function call, the function returns an error and the program is stopped.

Example

$data = RECEIVE

[**NOTE**]

The parameters of serial communication are:

- Baudrate: 115200
- Date size: 8
- Parity: None
- Handshake: OFF

EXISTCOM

Function

Checks for input in the serial communication buffer.

Explanation

If there is input in the serial communication buffer, TRUE (1.0) is returned, if the data does not exist, FALSE (0.0) is returned. The function is useful when the robot has to wait for data received from an external device.

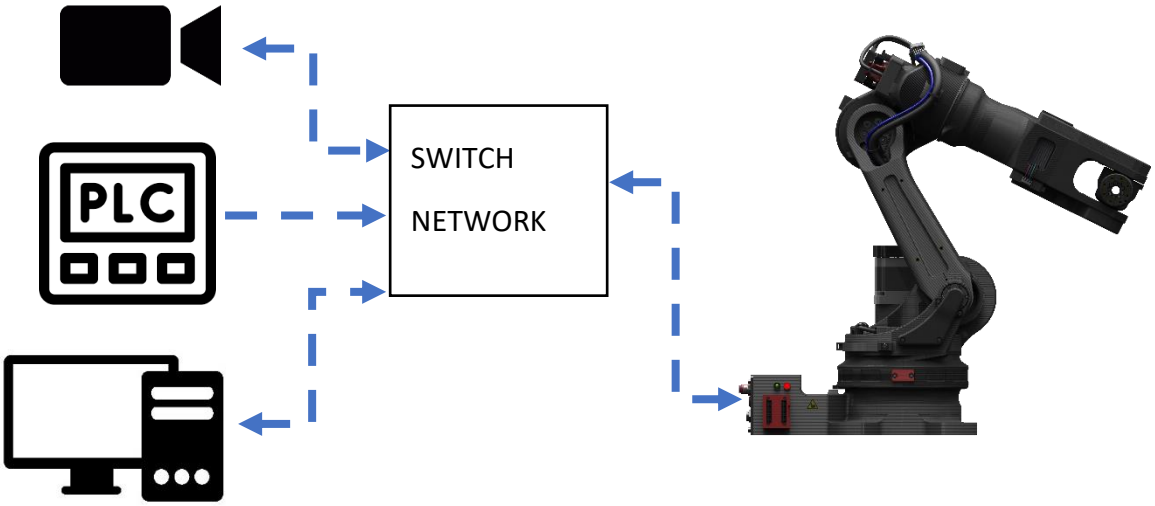
Example

```
WHILE (EXISTCOM == FALSE)
        TWAIT 0.1
END
```

The above example loops the program until the data enters the serial communication buffer.

12.6 TCP/IP AND UDP COMMUNICATION

TCP/IP and UDP communication is used to transfer data between the robot and external equipment such as PLCs, operator panels, accessories such as sensors, vision devices and PCs



The astorino robot can act as a server as well as a client in communication

12.7 TCP/IP COMMUNICATION FUNCTIONS

TCP_ACCEPT	Checks whether a connection request has been received
TCP_CLOSE	Interrupts communication
TCP_CONNECT	Creates a socket and sends a connection request
TCP_LISTEN	Creates a socket and waits for connection requests
TCP_END_LISTEN	Ends waiting for a connection request
TCP_SEND	Sends a string of data
TCP_RECEIVE	Receives a string of data

TCP_ACCEPT *return variable, port number*

Function

Program instruction for checking the connection request (used by the server to start the communication service).

Checks whether a connection request for robot communication has been received through the specified port and, if so, establishes a connection. The connection is set when this instruction completes normally. If a communication error occurs during execution, an error code (-1.0) is returned.

It is possible to connect up to eight (8) clients to the astorino robot at the same time.

Parameter

Return variable

Sets a variable that stores the results of the connection attempt. Stores the socketID (0-7) of the connected client. If the connection attempt fails, a value (-1.0) is returned.

The value -1.0 is returned when a communication error occurs. However, the execution of the program does not stop at a communication error.

Port number

Specifies the port number that points to the channel at the other end of the connection. The acceptable range is **8192 – 65535**, otherwise an error will occur.

Example:

```
WHILE socketID < 0 DO
    TCP_ACCEPT socketID port
    TWAIT 0.2
END
```

The above example loops the program until the client connects to the robot.

TCP_CLOSE *return variable, port number*

Function

Program instruction to terminate the connection (used to terminate the communication service). Closes the connection for TCP/IP communication and closes the port. If a communication error occurs, an error code (-1.0) is returned, and the program does not stop.

Parameter

Return variable

Sets a variable that stores the execution results.

0 is stored when execution is performed normally.

-1 when an error occurred in the execution of functions

Port number

Specifies the socketID that you receive as a result of **TCP_ACCEPT** or **TCP_CONNECT**

Example

TCP_CLOSE .ret, socketID

TCP_CONNECT *return variable, port number, IP address*

Function

Program instruction for requesting a connection (used by the client to start the communication service with the server) Creates a socket and connects to the specified port number. Then, a connection request is sent to the specified node and a connection is established. The node is determined by the IP address of the server. If a communication error occurs during execution, -1.0 is returned and program execution is not stopped.

Parameter

Return variable

Specifies a variable that stores the execution results. A value of 8 is returned when execution is performed normally.

-1 is returned when an error occurs when you try to connect.

Port number

Specifies the port number that points to the channel at the other end of the connection. The socket is associated with this port number.

The acceptable range is 8192 – 65535, otherwise an error will occur.

IP address

Specifies an array variable that stores the IP address of the server (32 bits)

The IP address is stored in 8-bit increments to each element of an array variable in order from the beginning of the IP address.

Example:

IP[1] = 192

IP[2] = 168

IP[3] = 0

IP[4] = 100

WHILE socketID < 0 DO

    TCP_CONNECT socketID,port,IP[1]

    TWAIT 0.2

END

The above example loops the program until the robot connects to the server.

TCP_LISTEN *return variable, port number*

Function

Program instruction to start waiting for a connection request (used by the server to start the communication service) Creates a socket and binds it to a specific port number and waits for a connection request to that socket. If a communication error occurs during execution, -1.0 is returned and program execution is not stopped.

Parameter

Return variable

Specifies a variable that stores the execution results.

A value of 0 is returned when execution is performed normally.

The value -1 is returned when an error occurs during execution.

Port number

Specifies the port number that points to the channel at the other end of the connection. The socket is associated with this port number.

The acceptable range is 8192 – 65535, otherwise an error will occur.

Example:

TCP_LISTEN .ret, port

TCP_END_LISTEN *return variable, port number*

Function

Program instructions for terminating the connection. Terminates waiting for a connection request on the socket specified by the TCP_LISTEN and closes the socket. If a communication error occurs, -1.0 is returned and the program does not stop.

Parameter

Return variable

Specifies a variable that stores the execution results. A value of 0 is returned when execution is performed normally. The value -1 is returned when an error occurs at execution

Port number

Specifies the socket currently waiting for a connection request (the socket specified in **the TCP_LISTEN** statement).

Example:

TCP_END_LISTEN .ret, port

TCP_SEND *return variable, socketID, string $string variable*

Function

Instruction of the program to send data. Sends data based on the TCP protocol. The data to be sent is specified as string variables. If a communication error occurs, -1.0 is returned and program execution does not stop.

Parameter

Return variable

Sets a variable that stores the execution results. A value of 0 is returned when execution is performed normally. The value -1.0 is returned when an error occurs during execution.

SocketID

Specifies the socket ID number obtained after executing **a TCP_ACCEPT** or **TCP_CONNECT** statement.

$string variable

Specifies the string variable in which the data to be sent is stored. Elements of a variable string are sent in order from first to last. Numeric data can be sent after conversion to string format by using **the $ENCODE** function.

Example:

TCP_SEND .ret, socketID, $data

TCP_RECV *return variable, socketID, $string variable*

Function

Program manual for receiving data. Receives data sent over TCP and stores it in the specified string variable. If a communication error occurs, -1 is returned and program execution does not stop.

Parameter

Return value

Specifies a variable that stores the execution results. A value of 0 is returned when execution is performed normally. The value -1.0 is returned when an error occurs during execution.

socketID

Specifies the socket ID number obtained after executing a **TCP_ACCEPT** or **TCP_CONNECT** statement.

$string variable

Specifies the string variable in which the received data is stored. The data is received as character data with a length of 1 byte each. When receiving numeric data, you must convert it from a string variable to a numeric variable by using the **VAL** function.

Example:

```
WHILE .ret < 0 TO
    TCP_RECV .ret, socketID, $data
    TWAIT 0.2
END
PRINT $data
```

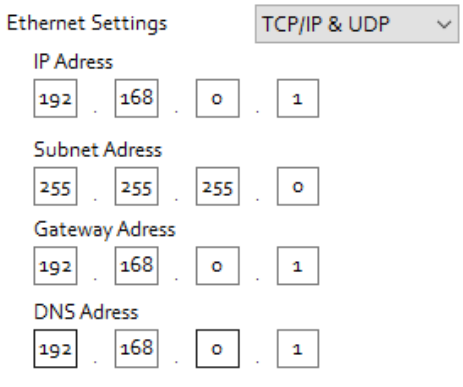
The above example loops the program until it receives the data.

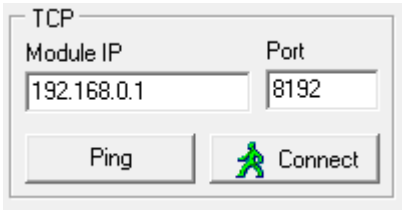
12.8 TCP/IP SERVER EXAMPLE

In the following example, astorino acts as a TCP/IP communication server. In order to test the operation of the program, the Hercules program from HW-group.com was used

(https://www.hw-group.com/software/hercules-setup-utility)

Robot and Hercules settings





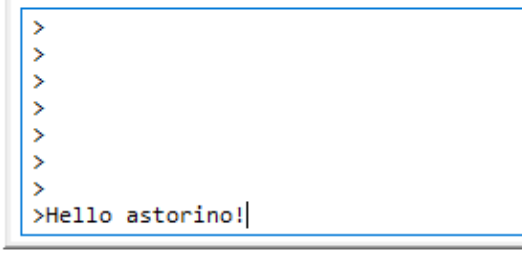
Program code executed by the robot:

```
.PROGRAM TCP1
  .ret = -1
  .ret2 = -1
  socketID= -1
  port = 8192
  $data = "Hello PC!"
  $data2 = ""
  TCP_LISTEN .ret, port
  WHILE socketID < 0 DO
    TCP_ACCEPT socketID, port
    TWAIT 0.2
  END
  TCP_SEND .ret, socketID, $data
  WHILE .ret2 < 0 DO
    TCP_RECV .ret2, socketID, $data2
    TWAIT 0.2
  END
  PRINT $data2
  TCP_CLOSE .ret,socketID
  TCP_END_LISTEN .ret, port
.END
```

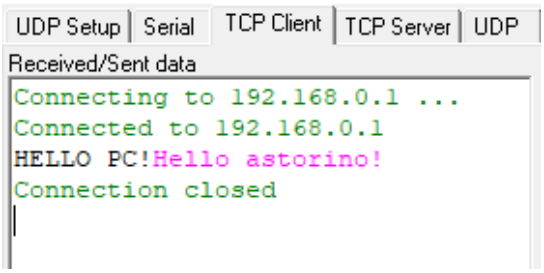
Data sent from Hercules software:

Results of the above program:

View of the astorino software terminal



View of the Hercules window

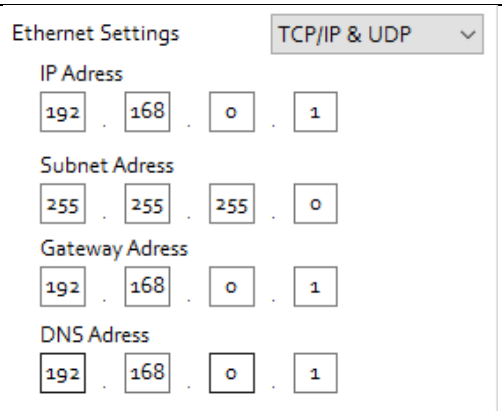


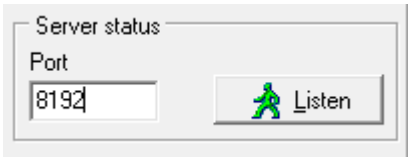
12.9 TCP/IP CLIENT EXAMPLE

In the following example, astorino acts as a TCP/IP communication client. In order to test the operation of the program, the Hercules program from HW-group.com was used

(https://www.hw-group.com/software/hercules-setup-utility)

Robot and Hercules settings





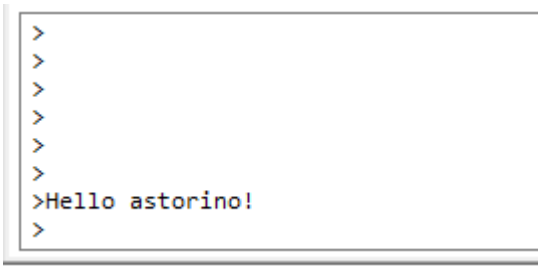
Program code executed by the robot:

```
.PROGRAM TCP
  .ret = -1
  .ret2 = -1
  socketID = -1
  port = 8192
  IP[1] = 192
  IP[2] = 168
  IP[3] = 0
  IP[4] = 100
  $data = "Hello PC"
  $data2 = ""
  WHILE socketID < 0 DO
    TCP_CONNECT socketID,port,IP[1]
    TWAIT 0.2
  END
  WHILE .ret< 0 DO
    TCP_SEND .ret, socketID, $data
    TWAIT 0.2
  END
  WHILE .ret2 < 0 DO
    TCP_RECV .ret2, socketID, $data2
    TWAIT 0.2
  END
  PRINT $data2
  TCP_CLOSE .ret,socketID
.END
```

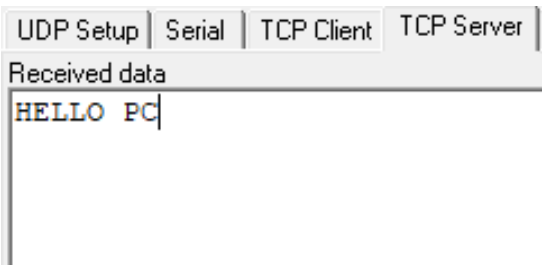
Data sent from Hercules software:

Results of the above program:

View of the astorino software terminal



View of the Hercules window



12.10 UDP COMMUNICATION FUNCTIONS

UDP_SENDTO	Sends specified string data over UDP/IP.
UDP_RECVFROM	Receives and stores data in the specified string variable via UDP/IP.

UDP_SENDTO *return variable, IP address, port number, $string variable*

Function

Sends a string of data over the UDP protocol. The data to be sent is specified in a variable string. This statement creates a socket, sends data, and closes the socket in a single sequence.

If a communication error occurs, -1 is returned and program execution does not stop.

[**NOTE**]

The **UDP_SENDTO** function opens the socket on port **8192**

Parameter

Return variable

Specifies a variable that stores the execution results.

A value of 0 is returned when execution is performed normally.

The value -1.0 is returned when an error occurs during execution.

IP address

Specifies an array variable that stores the IP address of the server (32 bits)

The IP address is stored in 8-bit increments to each element of an array variable in order from the beginning of the IP address.

Port number

Specifies the port number that points to the channel at the other end of the connection. The socket is associated with this port number.

The acceptable range is **8192 - 65535**, otherwise an error will occur.

$string variable

Specifies the string variable in which the data to be sent is stored. Elements of a variable string are sent in order from the first to the last. Numeric data can be sent after conversion to string format by using the **$ENCODE** function.

Example:

UDP_SENDTO .ret, IP[1],port,$data

UPD_RECVFROM return variable, IP address, port number, $string variable

Function

Receives a string of data sent over the UDP protocol. The received data is saved in a variable string. This statement creates a socket, sends data, and closes the socket in a single sequence.

If a communication error occurs, -1 is returned and program execution does not stop.

Parameter

Return variable

Specifies a variable that stores the execution results.

A value of 0 is returned when execution is performed normally.

The value -1.0 is returned when an error occurs during execution.

IP address

Specifies an array variable that stores the IP address of the server (32 bits)

The IP address is stored in 8-bit increments to each element of an array variable in order from the beginning of the IP address.

Port number

Specifies the port number that points to the channel at the other end of the connection. The socket is associated with this port number.

The acceptable range is **8192 - 65535**, otherwise an error will occur.

$string variable

Specifies the string variable in which the data to be sent is stored. Elements of a variable string are sent in order from the first to the last. Numeric data can be sent after conversion to string format by using the **$ENCODE** function.

Example:

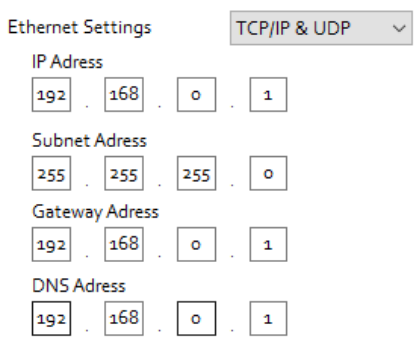
UDP_RECVFROM .ret,port, $data

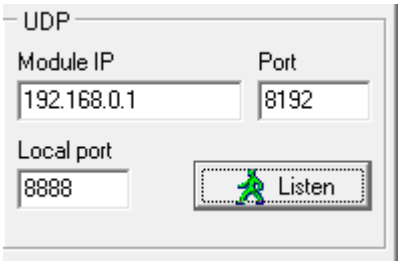
12.11 UDP EXAMPLE - SENDING DATA

In the following example, astorino acts as a sender of UDP data. In order to test the operation of the program, the Hercules program from HW-group.com was used

(https://www.hw-group.com/software/hercules-setup-utility)

Robot and Hercules settings



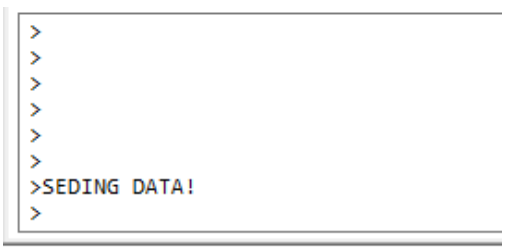


Program code executed by the robot:

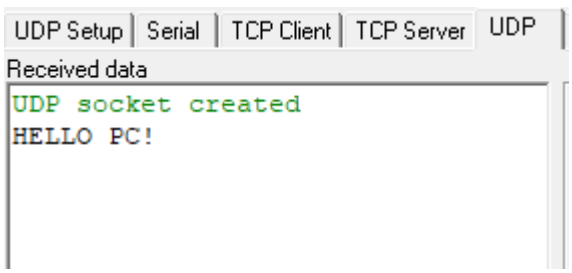
```
.PROGRAM UDP2
  .ret = 0
  $data = "Hello PC!"
  IP[1] = 192 ;computer IP
  IP[2] = 168
  IP[3] = 0
  IP[4] = 100
  port = 8888 ;localport
  PRINT "SEDING DATA!"
  UDP_SENDTO .ret, IP[1],port,$data
.END
```

Results of the above program:

View of the astorino software terminal



View of the Hercules window

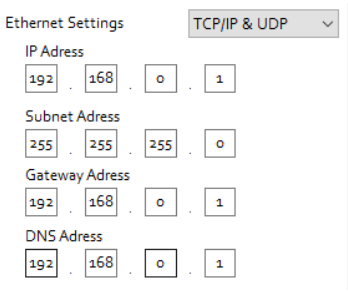


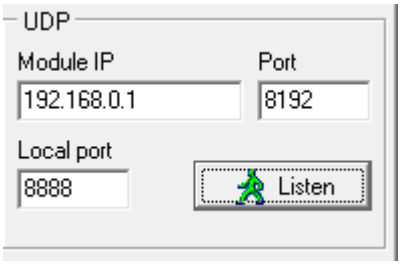
12.12 UDP EXAMPLE - RECEIVING DATA

In the following example, astorino acts as a recipient of UDP data. In order to test the operation of the program, the Hercules program from HW-group.com was used

(https://www.hw-group.com/software/hercules-setup-utility)

Robot and Hercules settings





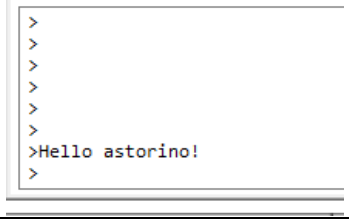
Program code executed by the robot:

```
.PROGRAM UDP
  port = 8192
  $data = ""
  .ret = -1
  WHILE .ret < 0 DO
    UDP_RECVFROM .ret,port, $data
    TWAIT 0.1
  END
  PRINT $data
.END
```

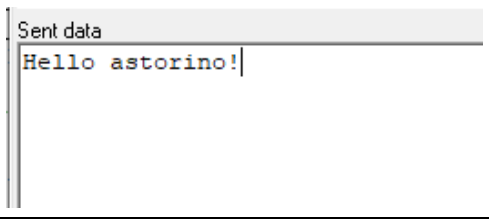
Data sent from Hercules software:

Results of the above program:

View of the astorino software terminal



View of the Hercules window

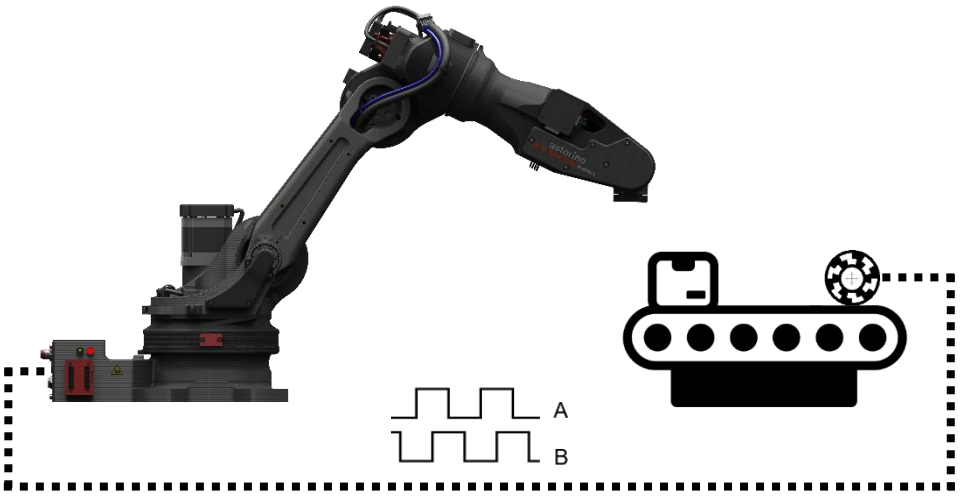


12.13 COOPERATION WITH EXTERNAL ENCODER

In standard robot operations, the workpiece/workpiece remains stationary during operation. The conveyor synchronization function allows operations on objects moving on the conveyor belt.
Using the function of cooperation with an external encoder, the robot moves, synchronizing its movement with a moving object on the conveyor belt. To synchronize with the moving workpiece, the robot can use up to two external incremental encoders.
Taking into account the sequence of movements and the flow of the program should be avoided:

- movements that will cause going beyond the working range of the robot,
- unnecessary pause of work (stopping the robot while the object passes by the robot)

Before using the synchronous conveyor function, the parameters of the resolution and direction of movement of the conveyor belt must be set. Set the data in the Astorino software.

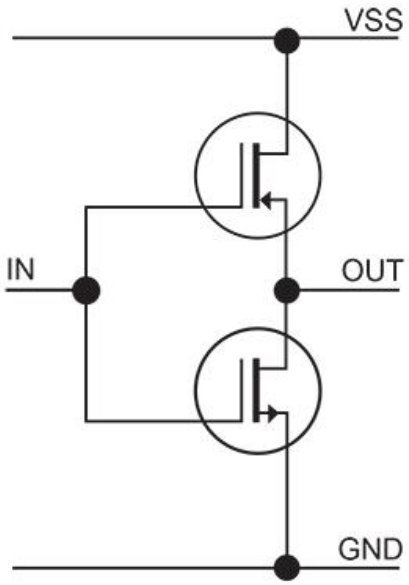


12.14 SUPPORTED ENCODERS

Astorino is able to handle up to two additional incremental encoders at the same time.

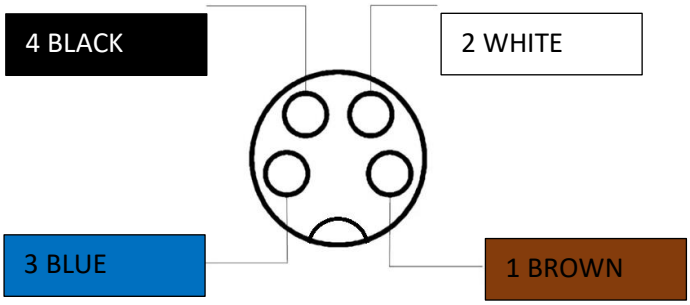
The basic parameters of the supported encoders are:

- Operating voltage 24V,
- Signal outputs A and B,
- Recommended resolution of no more than 300 PPR (pulses per revolution),
- Outputs operating in PUSH-PULL configuration.



Output in PUSH-PULL configuration

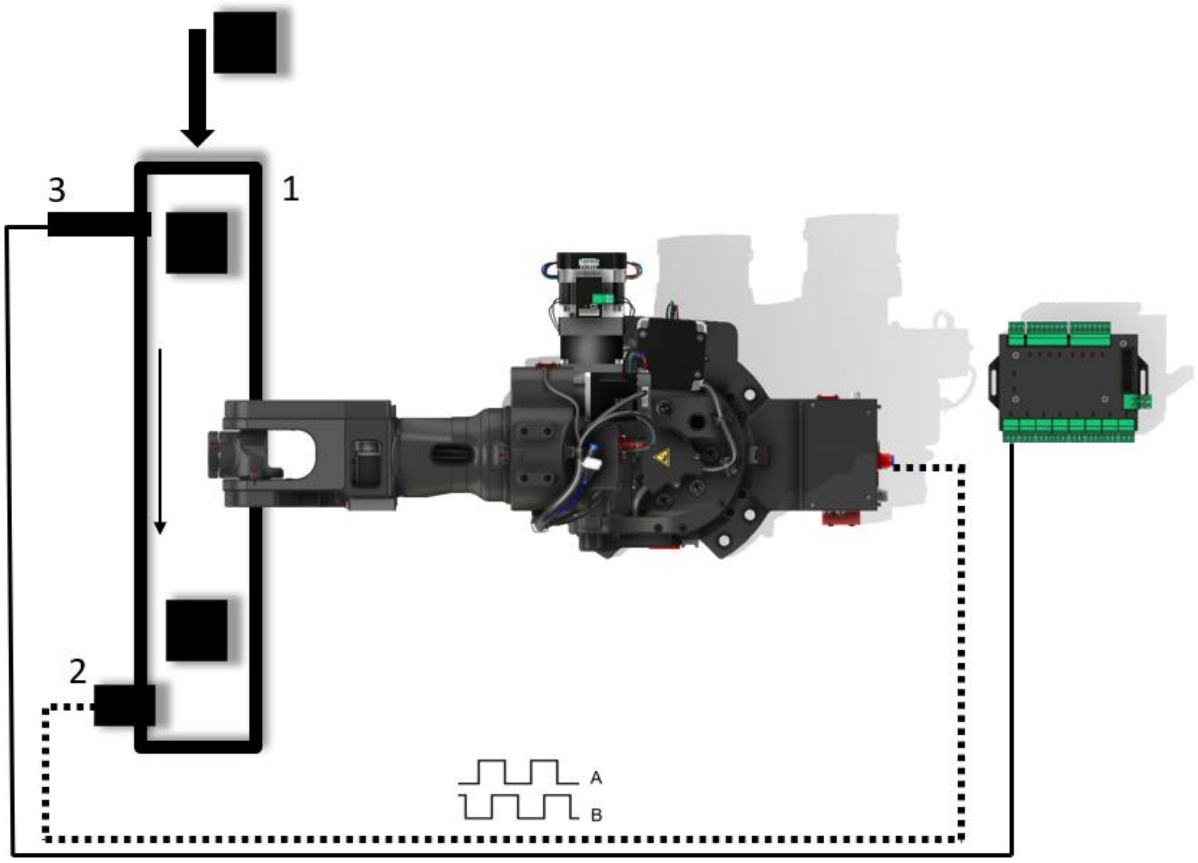
Every encoder in the robot is connected on the M8 four-pole plug.



Entry No.	1 BROWN	2 WHITE	3 BLUE	4 BLACK
1	GND	24V	A	B
2	GND	24V	A	B

12.15 EXAMPLE OF A CONVEYOR BELT APPLICATION

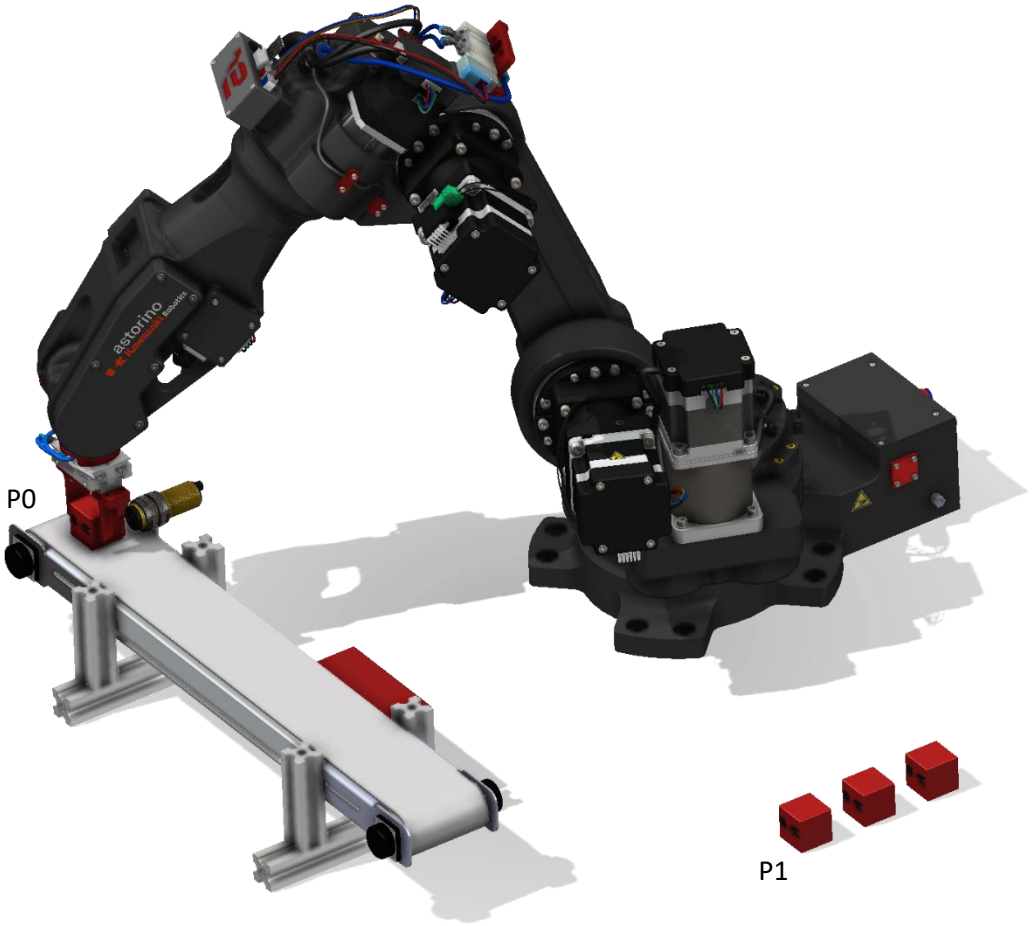
In the example below, the robot is equipped with a conveyor belt (1), a 24V pulse encoder (2) and a proximity sensor (3). The encoder has been connected to the input of the first encoder and the proximity sensor to the first input in the 24V I/O module. In order to clarify the diagram, the connection of the 24V IO module with the robot is not shown. The following example assumes that the conveyor has its own control and its movement matches the arrow in the drawing.



In the above application, the user gives the workpieces (cubes) at the beginning of the conveyor belt, when the sensor triggers, a point is saved, to which the robot then goes and picks the detail at the same time synchronizing with the conveyor belt. It then puts the items in a different location.

The first thing to do is to configure the conveyor setting in the robot settings. In this example, the resolution is 0.1mm/bit and the direction is set to X-.

Before starting, record the point where the workpiece is located at the time of detection by the sensor (3) and the deposit point P1



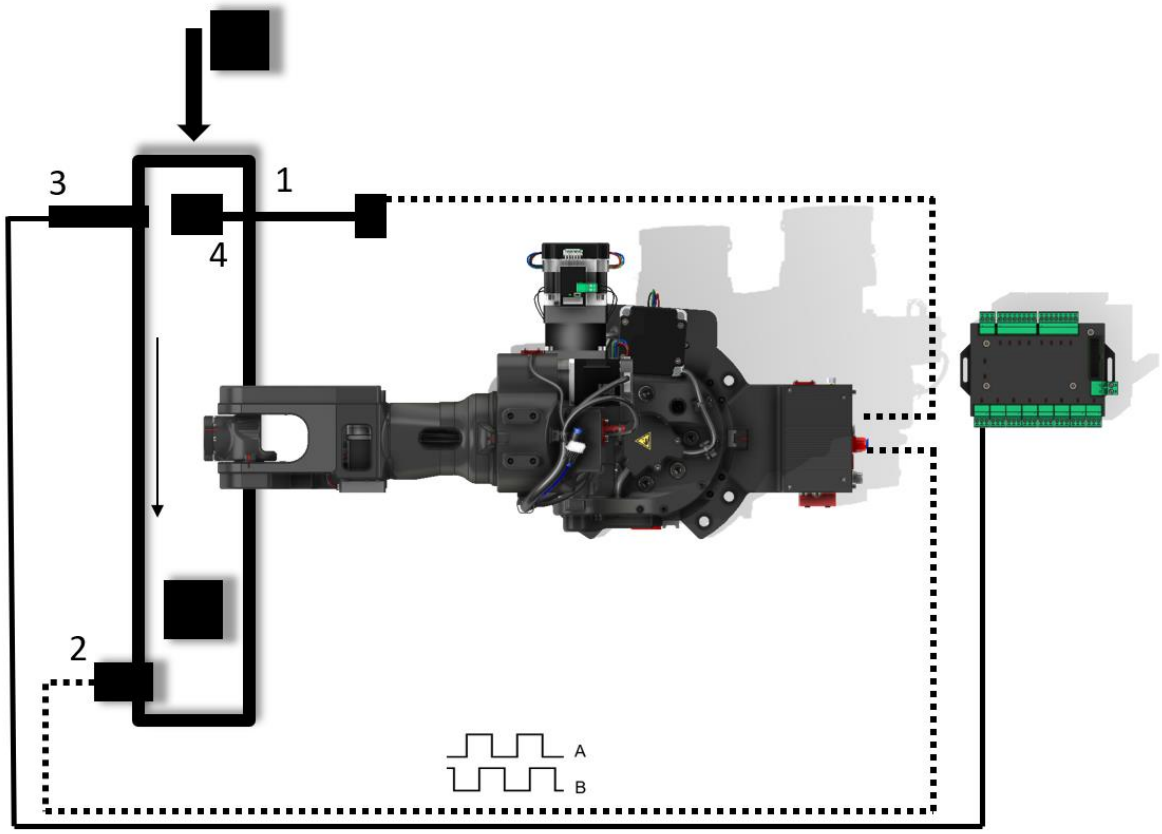
In order for the above application to work properly, the workpieces should be given in the same orientation and position on the conveyor belt (relative to the width), this can be done by designing appropriate bumpers that will automatically position the detail in the middle of the conveyor belt

Example program:

```
.PROGRAM CONV
  SPEED 100 MM/S ALWAYS
  TOOL 1
  POINT PICK = P0 ;P0 saved point at sensor
  POINT PLACE = P1 ;P1 saved put away point
  HOME
  CVCOOPJT 8; synch with 1st conv
  CVRESET 8
  WHILE SIG(1002) == TRUE DO
    SWAIT 1001 ;wait conv sensor signal
    ENC = CVPOS
    POINT/8 PICK = ENC ;store current encoder value to PICK
    CVWAIT 50 ; wait till conv moved 50 mm
    CVLAPPRO PICK,50
    SPEED 50 MM/S
    CVLMOVE PICK ;move to PICK
    CVDELAY 0.5 ;wait above conv 0.5s
    SIGNAL 1 ;close gripper
    CVDELAY 1 ;wait above conv 1s
    CVLDEPART 50
    JAPPRO PLACE, 50
    SPEED 20 MM/S
    LMOVE PLACE
    TWAIT 0.5
    SIGNAL -1
    TWAIT 1
    LDEPART 50
    POINT PLACE = SHIFT(PLACE BY 0,-50,0)
    IF CVPOS > 5000 THEN
      CVRESET 8 ; reset encoder if too big
    END
  END
.END
```

12.16 EXAMPLE OF A CONVEYOR BELT AND VISION SYSTEM APPLICATION

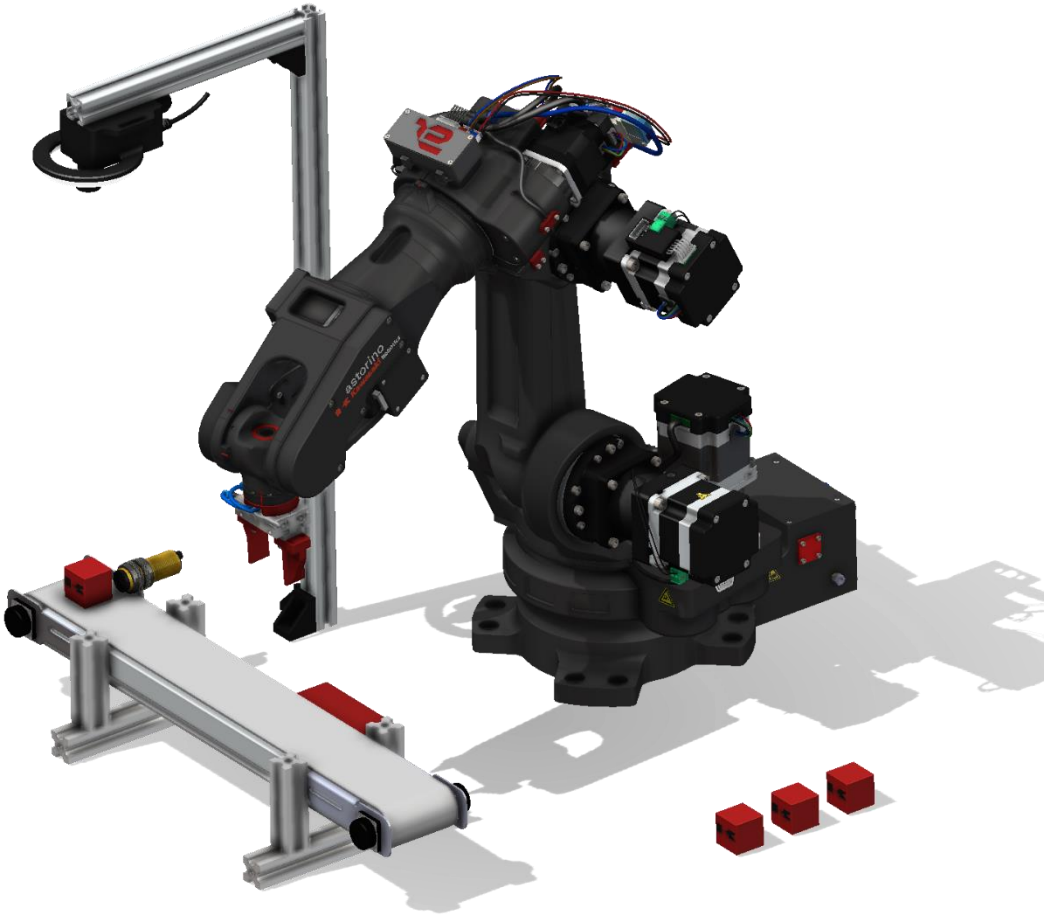
In the example below, the robot is equipped with a conveyor belt (1), a 24V pulse encoder (2), a proximity sensor (3) and a vision system (4). The encoder was connected to the first encoder input, the proximity sensor to the first input in the 24V I/O module, and the vision system to the Serial input in the robot base. In order to clarify the diagram, the connection of the 24V IO module with the robot is not shown. The following example assumes that the conveyor has its own control and its movement matches the arrow in the drawing.



In the above application, the user puts workpieces (cubes) at the beginning of the conveyor belt, when the sensor triggers, the camera is activated, which detects the object on the conveyor belt and sends the coordinates to the robot. A saved point is used to determine the location of the workpiece. The robot then goes and picks up the cube while synchronizing with the conveyor belt. It then puts the picked up items in a different location.

The first thing to do is to configure the conveyor setting in the robot settings. In this example, the resolution is 0.1mm/bit and the direction is set to X-.

Before starting, calibrate the camera according to the instructions of the vision system and teach the P1 place point. It is also necessary, as in the previous example, to reach the point P0 at any position of the conveyor belt so as to read the coordinate Z of the intake position. Then enter it in the program in the line **POINT PICK = TRANS(dataX, dataY, height,0,0,0)**. In the example program, the value is 100mm.



```
. .PROGRAM CONV
  SPEED 100 MM/S ALWAYS
  TOOL 1
  POINT PLACE = P1 ;P1 saved put down point
  HOME
  CVCOOPJT 8; synch with 1st conv
  CVRESET 8
  WHILE SIG(1002) == TRUE DO
    SWAIT 1001 ;wait conv sensor signal
    SEND "T"
    WHILE EXISTCOM == false DO
      TWAIT 0.05
    END
    $temp = RECEIVE
    $temp2 = $DECODE($temp,"/")
    $temp3 = $DECODE($temp, "/")
    $temp4 = $DECODE($temp, "/")
    dataX = VAL($temp2)
    dataY = VAL($temp3)
    dataA = VAL($temp4)
    IF ((dataX <> 0) OR (dataY <> 0)) THEN
      POINT PICK = TRANS(dataX,dataY,100,0,0,0)
      POINT/OAT PICK = P0
      POINT PICK = PICK + RZ(dataA)
      ENC = CVPOS
      POINT/8 = ENC
      CVWAIT 100 ; wait till conv moved 50 mm
      SPEED 100 MM/S ALWAYS
      CVLAPPRO PICK, 40
      SPEED 40 MM/S ALWAYS
      CVLMOVE PICK ;move to PICK
      CVDELAY 0.5 ;wait above conv 0.5s
      SIGNAL 1 ;close gripper
      CVDELAY 1 ;wait above conv 1s
      CVLDEPART 50
      JAPPRO PLACE, 50
      SPEED 20 MM/S
      LMOVE PLACE
      TWAIT 0.5
      SIGNAL -1
      TWAIT 1
      LDEPART 50
      POINT PLACE = SHIFT(PLACE BY 0,-50,0)
      IF CVPOS > 5000 THEN
        CVRESET 8 ; reset encoder if too big
      END
    ELSE
      PRINT "No workpiece"
      CVRESET 8
    END
  END
.END
```

13 SAMPLE PROGRAMS

This chapter provides sample programs in AS.

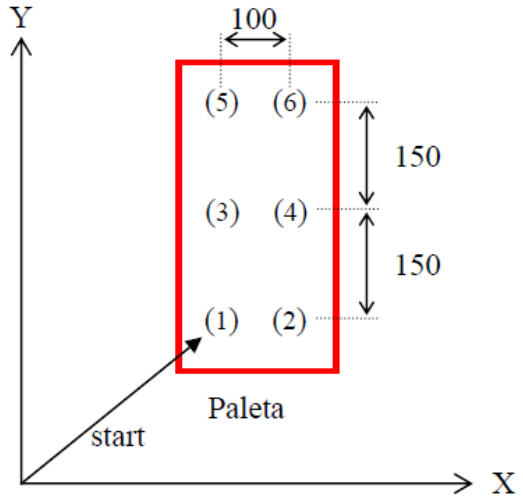
13.1 INITIAL CONFIGURATION OF PROGRAMS

Performing the following steps before using any robot functions makes programming easier.

- Move the robot to the home position.
- Define the variables you need for each task. (e.g. for palletizing, enter the number of items on the pallet)
- Initialize the counter, flag, etc.
- Set the coordinate system of the tool for the task at hand.
- Set the global coordinate system for the task.

The following is an example of a subroutine for initializing settings for the palletizing job shown in the figure.

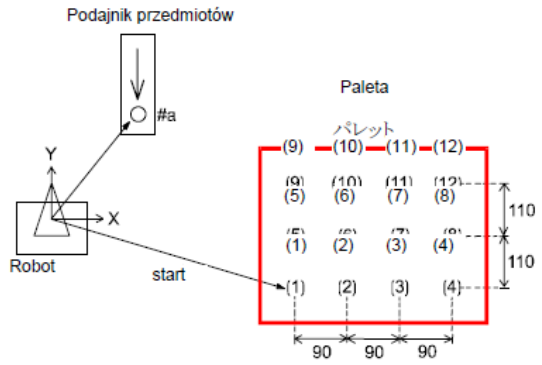
```
TOOL 1 ;tool (1)
rowmax = 3  ; 3 rows
colmax = 2   ; 2 collums
xs = 100  ; X(ΔX=100mm).
ys = 150  ; Y(ΔY=150mm).
POINT put = start
HOME
```



In the following example, items are palletized in order from (1) to (6). Enter the initial settings listed below. The palette is positioned parallel to the robot's global coordinate system.

13.2 PALLETIZING

In the following example, items are picked from the feeder and placed on pallets in three rows (at a distance of 110 mm) and four columns (at a distance of 90 mm). To simplify the example, both the palette and the objects on the palette are aligned parallel to the XY plane in the robot's global coordinate system. The procedure of synchronizing the feeder and the robot using external I/O signals (SWAIT instructions, SIGNAL, etc.) was also omitted.



- The pallet is positioned parallel to the XY.
- Item #P0 (Item Feeder) and "P0" (where the first item is placed) must be defined before the program can be executed.

Sample program

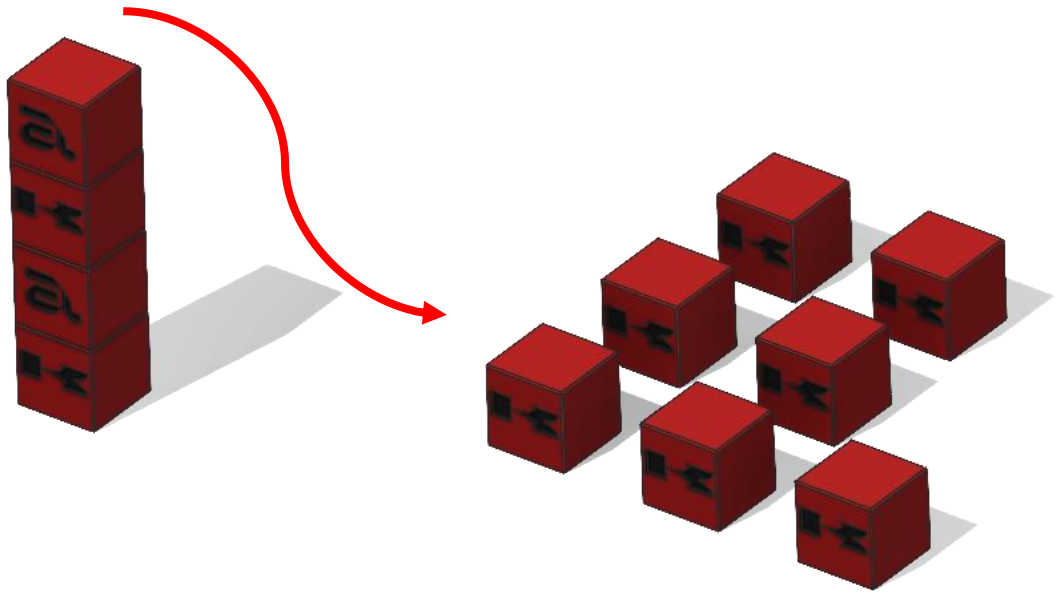
```
.PROGRAM palletize
  ; initial settings (3 rows, 4 columns)
  ; span X=90, Y=110
  rowmax = 3
  colmax = 4
  xs = 90
  ys = 110
  gripper = 1 ; 1st signal - gripper
  SPEED 100 MM/S ALWAYS
  POINT put = P0
  SIGNAL -gripper ;open gripper
  ; Palletization start
  FOR row = 1 TO rowmax
    FOR col = 1 TO colmax
      JAPPRO #P0, 100
      SPEED 30
      LAPPRO #P0, 50
      SPEED 30 MM/S
      LMOVE #P0
      TWAIT 0.2
      SIGNAL gripper
      TWAIT 1
      LDEPART 200
      JAPPRO put, 200
      SPEED 30 MM/S
      LMOVE put
      TWAIT 0.2
      SIGNAL -gripper
      TWAIT 1
      LDEPART 200
      ;next place point calculation - row
      POINT put=SHIFT (put BY xs, 0, 0)
    END
    ;next place point calculation - column
    POINT put = SHIFT (P0 by 0, ys * row, 0)
  END
.END
```

13.3 PICK&PLACE – AN EXAMPLE OF PALLETIZING

This program takes cubes from a single tower and then places them by the number of rows, the number of columns and the number of levels.

You can customize:

- Item size (cubes)
- Distance between cubes,
- Number of rows, columns and levels,



```
.PROGRAM PAL1
;-------------- Init ---------
deltaX = 60 ;distance between workpieces X
deltaY = 60 ;distance between workpieces Y
deltaZ = 30 ;layer height
numLev = 2
numRow = 1
numCol = 2
numPcs = numLev*numCol*numRow  ;pieces count
height = 25   ;height of a workpiece (25 mm)
;-------------- variable init ---------
x = 0
y = 0
z = 1
SIGNAL -1
SPEED 100 mm/s always
POINT place = p2
POINT pick = P1
POINT pick = SHIFT(p1 BY 0,0,numPcs*height)
;P1 on the table, pick shifted by number of pieces in Z
HOME
LAPPRO pick, 40
;---------------------------------- Pal-----------------
FOR z = 0 TO (numLev-1)
  FOR y = 0 TO (numRow-1)       ; rows in Y
    FOR x = 0 TO (numCol-1)      ;col in X
      POINT pick = SHIFT(pick BY 0,0,-height);calc new pick pose
      JAPPRO pick,40
      speed 20 mm/s
      LMOVE pick
      TWAIT 0.5
      SIGNAL 1 ;close the gripper
      TWAIT 0.5
      LDEPART 50
      LMOVE P3
      POINT place = p2
      POINT place = SHIFT(p2 BY deltax*x,deltay*y,deltaz*z)
      LAPPRO place,30
      speed 20 mm/s
      LMOVE place
      TWAIT 0.5
      SIGNAL -1 ;open the gripper
      TWAIT 0.5
      LDEPART 30
      LMOVE P3
    END
  END
END
.END
```

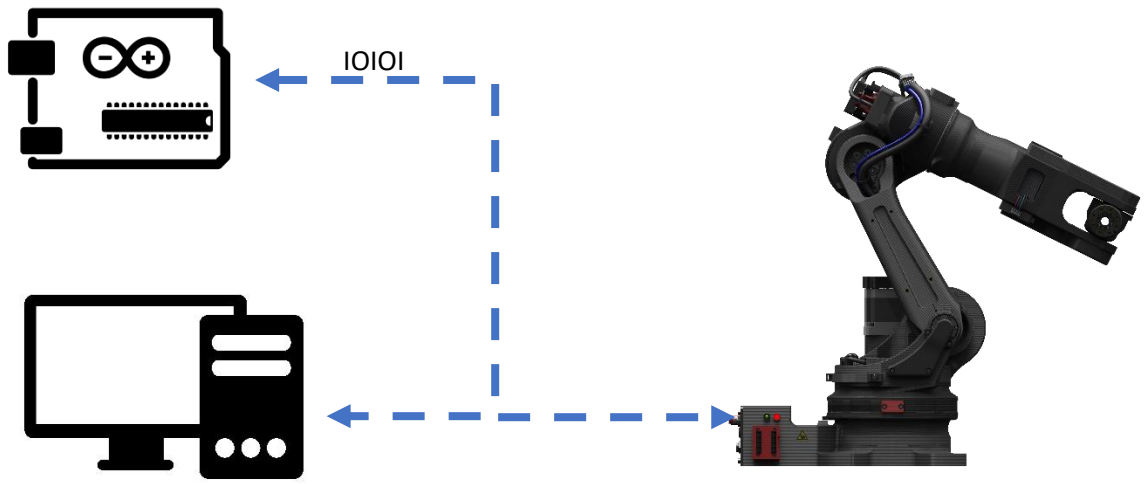
13.4 SAMPLE I/O PROGRAM

This sample program shows you how to use signals in many ways.

```
.PROGRAM IO
; ----- IO example program
; ----- Robot reads and sets IOs
  sensor = 1002 ;sets variable
  SWAIT 2001 ;wait until internal 1 signal is on
  SIGNAL 8 ;sets 8 output HIGH
  IF SIG(sensor) == TRUE THEN
    ;checks if sensor(2 input) is high
    SIGNAL 2002 ; sets 2 internal HIGH
  ELSE
    IF SIG(1001) == FALSE THEN
      SIGNAL -8 ;sets 1 output LOW
    END
  END
  BITS 1,4 = 12
  ;changes 12 to 4bit binary and sets that on out    puts from 1
  data = BITS(1004,4) ;read binary data from inputs
  ;4 bit from   4th output and changes that to decimal
  PRINT data
.END
```

13.5 SAMPLE SERIAL COMMUNICATION PROGRAM

This example shows how to use serial communication. The program can exchange data between the astorino robot and a PC (e.g. Matlab or SerialTerminal) or microcontroller (e.g. Arduino or ESP32).



```
.PROGRAM serial
; ----- Serial communication example program
; ----- Robot command frame form Serial Port
; ----- frames: P/ or L/x/y/z/
; ----- From X,Y,Z point is created
; ----- Sends current location if frame is P/
  SPEED 150 MM/S ALWAYS
  $S_FRAME = "XYZ"
  $S_FRAME2 = "JT"
  WHILE EXISTCOM == FALSE DO
    TWAIT 0.1
  END
  $TEMP = RECEIVE
  $COMMAND = $DECODE($TEMP,"/")
  PRINT $COMMAND
  ;RECEIVE DATA FROM SERIAL AND CREATE A POINT
  IF $COMMAND == "L" THEN
    $VAL1 = $DECODE($TEMP,"/")
    $VAL2 = $DECODE($TEMP,"/")
    $VAL3 = $DECODE($TEMP,"/")
    DATAX = VAL($VAL1)
    DATAY = VAL($VAL2)
    DATAZ = VAL($VAL3)
    POINT TEST = TRANS(DATAX,DATAY,DATAZ,0,0,0)
    POINT/OAT TEST = P0
    LMOVE TEST
    SEND "OK"
  END
  ;SEND CURRENT LOCATION TO SERIAL PORT
  IF $COMMAND == "P" THEN
    HERE TEMP
    HERE #TEMP
    DECOMPOSE TAB[0] = TEMP
    DECOMPOSE TAB2[0] = #TEMP
    FOR I = 0 TO 5
      TAB2[I] = TAB2[I]*180/PI
      $S_FRAME = $S_FRAME + $ENCODE(TAB[I]) + "/"
      $S_FRAME2 = $S_FRAME2 + $ENCODE(TAB2[I]) + "/"
    END
    SEND $S_FRAME
    SEND $S_FRAME2
  END
.END
```

⚠ REMARK

Serial communication operates at 3.3V, please use 3.3V compatible electronics or voltage level converters.

5V voltage can damage the main CPU chip!

14 MANUFACTURER INFORMATION

Kawasaki Robot
ASTORINO AS LANGUAGE INSTRUCTION

2024-01: 3rd edition

Publication: KAWASAKI Robotics GmbH