Machine and Robot in Harmony

RPS: Reference guide

Trio Motion Technology

Trio Robotics Series

First Edition • 2020

Revision: 1.4

Trio Programming Guides are designed to aid learning of the TrioBASIC language through description and examples. Each one will cover a particular topic and discuss which commands and parameters in the TrioBASIC are required to complete the task.

A general understanding of TrioBASIC is required and it is recommended to attend an introduction to TrioBASIC training course. The programming guides are not a replacement for the TrioBASIC help files which can be found in *Motion* Perfect as well as the manual which cover each command and parameter in more detail and should be referenced when required.

Any examples given in the programming guide will work and have been tested on an isolated controller. If you choose to use these examples on a machine please take care that it will not cause damage or injury and that they are correctly included in the project changing parameters and values where required.

UK | USA | CHINA | INDIA

www.triomotion.com

SAFETY WARNING

During the installation or use of a control system, users of Trio products must ensure there is no possibility of injury to any person, or damage to machinery.

Control systems, especially during installation, can malfunction or behave unexpectedly.

Bearing this in mind, users must ensure that even in the event of a malfunction or unexpected behaviour the safety of an operator or programmer is never compromised.

This document uses the following icons for your reference:

Information that relates to safety issues and critical software information.	Information to highlight key features or methods.	Useful tips and techniques.	Example programs.

Contents

1. RPS language extension **1**

1.1. Definition 1

2. Kinematic group **2**

2.1. Syntax 2

2.2. Behaviour 2

3. Robot selection and program types **4**

3.1. Robot selection 4

3.2. Program types 4

3.2.1 Robot Program 4

3.2.2 Robot Basic Program 5

3.3.3 Robot Library 5

3.1. Program Pointer 5

4. Frames and Tools **6**

4.1. Definition 6

4.2. Syntax 7

4.3. Behaviour 8

5. TARGET data type **11**

5.1. Definition 11

5.2. Cartesian 11

5.2.1. TARGET 11

5.2.2. GTA 11

Interaction with MOVEs 11

5.3. Joint 12

5.3.1. TARGETJ 12

5.3.2. GTAJ 12

Interaction with MOVEs 12

5.4. Syntax 12

5.5. Behaviour 13

5.1. Robot Position 15

5.1.1. ROBOT_LPOS 15

5.1.2. ROBOT_JPOS 15

6. Move commands **16**

6.1. General overview 16

Speed 17
Accel 17
Decel 17
Configuration 17
Interrupt by path percentage 18
Interrupt by input number 18
Interrupt by time 18
Routine Output 18
6.2. Precision zone 19
6.3. MOVEJ 20
Definition 20
Syntax 20
Examples 20
6.4. MOVEL 24
Definition 24
Syntax 24
Examples 24
6.5. MOVEC 28
Definition 28
Syntax 28
Examples 28
6.6. MOVEJREL 32
Definition 32
Syntax 32
Examples 32
6.7. MOVELREL 33
Definition 33
Syntax 33
Examples 33
7. Jogging 34
7.1. Definition 34
7.2. Syntax 34
7.3. Behaviour 34
8. Singularities 37
9. Robot configurations 39
10. Limits 41

10.1. Definition 41
11. Parasitic Compensation 42
11.1. Syntax 42
11.2. Description 42
11.3. Parameters 42
12. Errors 44
12.1. AXISSTATUS bit definitions 44
12.2. ROBOTSTATUS bit definitions 45
12.3. Robot log 45
Syntax 45
13. Collision detection 47
13.1. Definition 47
13.2. Syntax 48
14. Synchronisation 50
14.1. Attach Object Frame 50
Attach Object Frame with a conveyor 50
Attach Object Frame with a multipurpose machine 52
Detach Object Frame 52
General behaviour 52
14.2. Sync to object frame 53
Syntax 53
14.3. Sync to GTA 54
Syntax 54

1. RPS language extension

1.1. Definition

The Robotic Programming System or RPS, is an open and adaptable package of tools and software that offers a complete control and management of a robot system, reducing programming and optimizing productivity.

In this series, the Language Extension block is covered.

The language extension is a dedicated language for robotics base on Trio basic, which improves the efficiency by reducing the number of lines in your programs. It matches perfectly with the standard Trio Basic.

RPS has a group of kinematics transformations, advanced control algorithms deliver the possibility to program in one coordinate system when the machine does not have a direct mechanical connection to his coordinate system. Once RPS is set in the controller, a selected group of axes will be set as a robot.

In order to activate RPS in the controller, a configuration data to specify the lengths of mechanical links, operating modes and other information is required.

2. Kinematic group

KINEMATIC_GROUP is the command that activate RPS and links the required data to the specified robot. This command must be executed to have RPS set.

Although 8 kinematics can be initialised on a controller it may not be possible to process all 8 at a given SERVO_PERIOD. The number that can be run depends on many factors including, which kinematic is selected, drive connection method, if OBJECT_FRAME, ROBOT_FRAME, TOOL_OFFSET or COLLISION_OBJECT is enabled and additional factory communications.

The number of axes in the group must match the number of axes used by the KINEMATICS. The axes must also be ascending order though they do not have to be contiguous.

For more information and details about type of supported robots and how to set up the data for a robot refer to pick and learn series: Kinematics transformation block.

2.1. Syntax

KINEMATIC_GROUP(index, table offset, kinematic number, axis0 - axisn)

Keyword	Description
`Index`	From 0 to 7, it means up to 8 robots can be set in the same controller.
`Table offset`	Where the robot parameters are set.
`Kinematic number`	Type of robot.
`Axis0-axisN`	Axes belong the robot. They have to be in increase order

The number of axes in KINEMATIC_GROUP must match the number of axes used by the kinematic number selected. The axes must also be ascending order though they do not have to be contiguous.

2.2. Behaviour

It is possible to know if RPS is set and all information related to it by executing KINEMATIC_GROUP(index). The format shown is as below:

Group index [table_offset] [kinematic number]: axes TO={index [name] : TOOL_OFFSET parameters} RF={index [name] : ROBOT_FRAME parameters} OB={index [name] : OBJECT_FRAME parameters} CO{active collision objects}

The active COLLISION_OBJECTS value is in binary shown as a decimal format.

Example:

```
KINEMATIC_GROUP(0)

Terminal:
0 [0] [19] : 10, 11, 12, 13, 14, 15 TO={0[TO_default]:0.00000, 0.00000,
0.00000, 0.00000, 0.00000, 0.00000} RF={0[RF_default]:0.00000, 0.00000,
0.00000, 0.00000, 0.00000, 0.00000} OF={0[OF_default]:0.00000, 0.00000,
0.00000, 0.00000, 0.00000, 0.00000} CO{3}
```

RPS can easily be disabled by using KINEMATIC_GROUP(index,-1), all the axes in the group will be released.

All the axes of the group have to be IDLE, if not, the run time error “Operation cannot be performed until IDLE” will be thrown.

3. Robot selection and program types

3.1. Robot selection

RPS is capable of have configured up to 8 robots in the same system. In order to let the program knows what Robot should direct all subsequent motion commands and other parameters to read/write, a robot has to be selected first. To do this, **ROBOT(**robot ID**)** command has to be used.

Once a Robot is selected, it will stick to that process until the movements produced by that process are finished and program stops. So, if a program stops (process finishes) then the Robot will not be released until buffers are idle.

It is possible to check what process hold a particular Robot by reading **ROBOT(**robot ID**)** command as follow:

```
my_robot = ROBOT(0)
```

Deselect a robot from its process is possible using -1 index as follow:

```
ROBOT(-1)
```

> *This will release the Robot from the process that is selected only if ROBOT(-1) is executed in the same process than the selected Robot.*

STOP_ROBOT(robot ID**)** will cancel moves, clear buffers and stop process belonging from the selected robot index. This command can be called from other process, making easier to control a robot or group of robots from a main program.

DISABLE_ROBOT(boolean flag) :

This command will inhibit any robot programs from running. If any programs are already running on a process, it will be stopped, and the buffers will be cleared.

DISABLE_ROBOT(TRUE) will activate the command and DISABLE_ROBOT(false) will deactivate it.

3.2. Program types

There are two different robot program types: robot programs and basic robot programs. All robot program types can be fully edited by Motion Perfect.

3.2.1 Robot Program

Robot programs contain just the instructions that TPS can handle. All the MOVE instruction, Structures (While, If, Repeat, For structures), function calls, labels among other robot instructions. This type of program will have a '.ROB' file extension.

3.2.2 Robot Basic Program

Robot Basic Programs can contain all the existing Trio instructions (same as a BASIC program). Also, Robot Basic Program can be edited by TPS but for user levels 1 and the commands that are not in a guided form will be inserted by “type in” form.

Users can write complex programs as Robot Basic programs to have more control over their application. It is important to use this type of program and NOT a BASIC program as the controller can kill this type of program in case of an emergency.

This type of program will have ‘.RBS’ file extension

3.3.3 Robot Library

Robot Library contains user functions for the robot application. TPS can read this file and list all the functions in the Teach pendant. The allowed commands in this type of file are the same as the syntax allowed in a Basic Library.

This type of library will have a ‘.ROBLIB’ file extension.

3.1. Program Pointer

A robot program or a robot basic program can be run in two ways.

A normal RUN operation will run the selected program normally. The running program can be paused or stopped anytime while it is running.

Additionally, users can select a line and start running the program from the line. If multiple robots are present, the currently active robot will be selected, and all MOVE commands will be run on the selected robot. This can be changed by selecting another robot from the program or from the UI (MP or TPS). This allows more flexibility while testing.

This feature is only available in TPS and can be used only on the Teach Pendant. It will be coming to Motion Perfect soon.

4. Frames and Tools

4.1. Definition

Frames and tools are a 4x4 orthogonal matrix by which a position and orientation is completely defined. They are defined for a 3-axis translation and rotation. The rotations are applied using the Euler ZYX convention. This means that the Z rotation is applied first, then the Y is applied on the new coordinate system and finally the X is applied. The coordinate system is defined using the 'right hand rule' and the rotation of the origin is defined using the 'right hand turn'.

Frames are applied on the axis KINEMATIC_GROUP. If no KINEMATIC_GROUP is defined then a runtime error will be generated.

Movements are loaded with the selected frame or tool. This means that you can buffer a sequence of movements on different frames or tools. The active frame or tool is the one associated with the movement in the MTYPE. If the KINEMATIC_GROUP is IDLE then the active frame or tool is the selected frame or tool.

The system is compound of the following frames:

- **WORLD_FRAME**: represent the world coordinate that uniquely fix the system (0, 0, 0 position and identity matrix for orientation). It specifies the relationship between a moving observer and the object under observation. It cannot be overwritten but the coordinate system of RPS can be modified by using an OBJECT_FRAME.
- **OBJECT_FRAME**: on an object, it can change the coordinate system to program from. Once an OBJECT_FRAME is applied, the robot and all the points will be related to it.
- **ROBOT_FRAME**: on the robot base, it can change the position and orientation of the robot related to the active coordinate system (WORLD_FRAME as default or OBJECT_FRAME). Setting this frame the TCP (Tool Centre Point) is always related to WORLD_FRAME or OBJECT_FRAME, no matter what position the robot is.
- **TOOL_OFFSET**: offset between the TCP and the end-effector of the robot. It sets a distance and orientation of a tool from the end effector to the TCP. If no TOOL_OFFSET is applied, the TCP is at the end-effector of the robot.
- **TOOL_COLLISION**: data to build a bound box around the tool. For more information, please refer to Collision detection section.

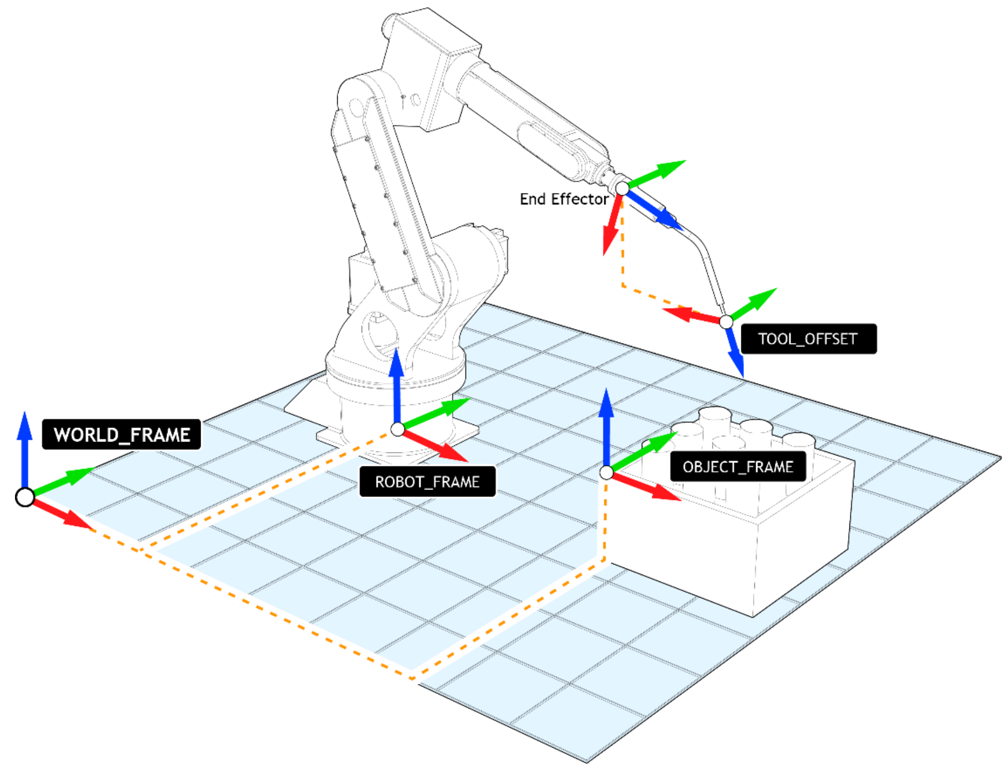


Figure 4-1: Coordinate system scenario.

4.2. Syntax

OBJECT_FRAME (identity, name, x offset, y offset, z offset, x rotation, y rotation, z rotation)

X offset, y offset and z offset have to be specified in mm.

X rotation, y rotation and z rotation have to be specified in degrees.

Example:

```
OBJECT_FRAME(1,"camera",200,0,400,0,180,0)
```

Same syntax for ROBOT_FRAME and TOOL_OFFSET.

TOOL_COLLISION(identity, name, x centre, y centre, z centre, half distance x, half distance y, half distance z)

The dimension of the tool collision should be taken as the maximum dimension that the tool can be. For example, if the tool is a gripper, use the gripper open dimension as described in the picture below.

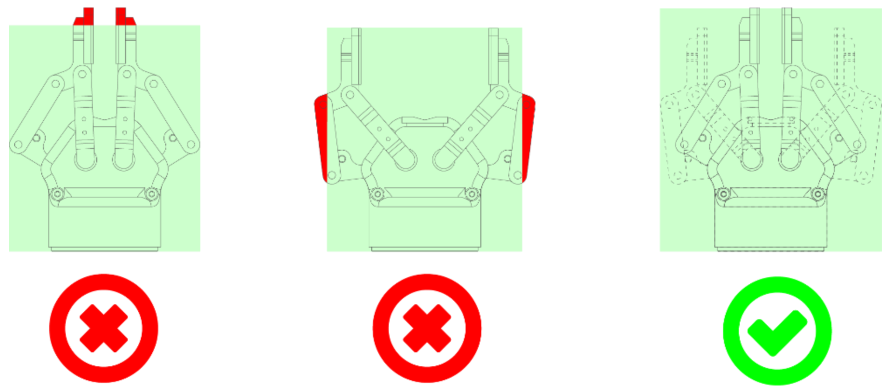


Figure 4-2: Tool collision dimensions.

4.3. Behaviour

There are up to 32 entries for every frame or tool offset, starting at index 1 until 31. Index 0 is the default one with values `"TO_default" : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000` and it is not possible to modify or remove.

Frames and tool offset have to be created, using the corresponding command and the desired values, before using them.

Example:

```
TOOL_OFFSET(1,"Gripper",100,50,0,0,0,0)
```

Parameter name is optional and unique, if not used the system leaves it empty.

If an entry with different index is created with the name of an existing entry, the system throws the run time error "Duplicate Identifier".

The entries can be modified at any time (except the active ones) with a new values.

Example:

```
Terminal:
0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
1 [Gripper] : 100.00000, 50.00000, 0.00000, 0.00000, 0.00000, 0.00000
```

```
TOOL_OFFSET(1,"Torch",200,0,50,0,45,0)
```

```
Terminal:
0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
1 [Torch] : 200.00000, 0.00000, 50.00000, 0.00000, 45.00000, 0.00000
```

Remove an entry is possible using the parameters -2 and index.

```
Terminal:
0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
1 [Torch] : 200.00000, 0.00000, 50.00000, 0.00000, 45.00000, 0.00000
2 [Gripper] : 150.00000, 50.00000, 0.00000, 0.00000, 0.00000, 0.00000
```

```
5 [Pencil] : 200.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
16 [Needle] : 300.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
```

```
TOOL_OFFSET(-2,2)

Terminal:
0 [TO_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
1 [Torch] : 200.00000, 0.00000, 50.00000, 0.00000, 45.00000, 0.00000
5 [Pencil] : 200.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
16 [Needle] : 300.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
```

Remove all entries using just the parameter -2.

If an entry is being selected it cannot be modifying or deleted, throwing the runtime error "TOOL cannot be modifying or delete while selected.

Request for existing entries is possible with parameter -1. It returns the existing ones plus the entry 0 (default). If the parameters are (-1, index) it returns only the requested index.

Example:

```
OBJECT_FRAME(-1)

Terminal:
0 [OF_default] : 0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000
1 [of2] : 517.35967, 0.41128, 950.75573, 28.23244, -177.09862, -22.97674
2 [of3] : 633.29181, 188.67548, 799.24254, 0.00000, 0.00000, 0.00000
3 [of4] : 633.29181, 188.67548, 799.24254, 0.00000, 90.00000, 0.00000
```

```
OBJECT_FRAME(-1,2)

Terminal:
2 [of3] : 633.29181, 188.67548, 799.24254, 0.00000, 0.00000, 0.00000
```

If you wish to check which OBJECT_FRAME, ROBOT_FRAME or TOOL_OFFSET are active you can print the details to the terminal using KINEMATIC_GROUP(group).

In order to use a frame or tool, it has to be selected either as a standalone command for a particular robot or as an embedded parameter (explained in MOVE section). Once it is selected, the system will use it as the active ones. The movements executed after a frame or tool selection will be related to the active one until it is finished. After finalize the movement a new frame or tool can be selected but it will be active once the movement is finished. With this methodology a "on the fly" selection of frame or tool is possible and gives to the programmer more flexibility.

To select a frame or tool just simply invoke the command with the index or name desired as a parameter.

Example:

```
OBJECT_FRAME("of3")
```

It is possible to request which tool or frame is active by assigning the return value of the commands TOOL_OFFSET, OBJECT_FRAME or ROBOT_FRAME with no parameters.

Example:

```
i = OBJECT_FRAME
```

5. TARGET data type

5.1. Definition

Target is a datatype to store position and orientation in the space and commonly used in move commands to move the robot at. Targets can be local or global, storing Cartesian or Joint positions.

Targets are related with the active coordinate system at the moment of its use. If a target has been stored related to another coordinate system than the active one, the robot will not move to the expected target. Make sure the selected frame or tool offset is the same used at the moment to store the target.

5.2. Cartesian

Target points can be taught/declared in cartesian workspace by using GTA for global use and TARGET for local use.

5.2.1. TARGET

TARGET is a local variable that stores a cartesian point. Local variables are deleted when the program that declared them stops running.

5.2.2. GTA

Global Target Array is global data type that stores cartesian target points (position and orientation) and is accessible from all programs. GTAs are saved in flash memory and are not lost on restarting the controller (similar to VRs).

Interaction with MOVEs

The following combinations are possible and will behave in the described way:

MOVEL GTA

This instruction will move the end effector of the robot to the target point in a straight(linear) line.

MOVEJ GTA

This instruction will move the end effector of the robot to the target point non-linearly. In other words, the robot will travel linearly in joint space.

The robot will go the cartesian target in its current configuration (and not the configuration the point was taught in)

MOVEC GTA GTA

This instruction will move the end effector of the robot in a circle.

All other combinations will trigger an error.

5.3. Joint

Target points can be taught/declared in cartesian workspace by using GTAJ for global use and TARGETJ for local use.

5.3.1. TARGETJ

TARGET is a local variable that stores a cartesian point. The data format is same as GTA. Local variables are deleted when the program that declared them stops running

5.3.2. GTAJ

Global Target Array Joint is a global variable that stores joint target points and is accessible from all programs. GTAJs are saved in flash memory and are not lost on restarting the controller (similar to VRs). In joint mode, GTAJs have the following data format:

GTA('ID number') = θ1, θ2, θ3, θ4, θ5, θ6, "point name as string"

Interaction with MOVEs

The following combinations are possible and will behave in the described way:

MOVEL GTAJ
The controller will calculate the linear target point using the joint co-ordinates. Then end effector of the robot will then move to the target point linearly.

If the robot is in a different orientation (as compared to the taught joint position) it will NOT change its configuration. The robot will move to the calculated cartesian position in the current configuration.

MOVEJ GTAJ

The robot will move to the joint target point in joint mode. It will travel linearly in joint space.

All other combinations will produce an error.

GTA and GTAJ share the same workspace, meaning there can only be a maximum of 1000 Global Targets (cartesian and joint combined). Users cannot have separate indices, for example:
GTA(1) = X1, Y1, Z1, U1, V1, W1, "pt1"

GTAJ(2) = t1, t2, t3, t4, t5, t6, "pt2"

In this example, GTAJ cannot have an index 1 and GTA cannot have index 2.

5.4. Syntax

- Global target:

GTA(index number) = float x, float y, float z, float u, float v, float w, string name

GTAJ(index numb) = float θ1, float θ2, float θ3, float θ4, float θ5, float θ6, string name

Example:

```
GTA(10) = 100,0,50,180,0,180,"Cartesian_point1"
GTAJ(20) = -10,50,10,0,15,0,"Joint_point1"
```

Name is optional and unique.

A GTA cannot be declare using a name instead of an id number. If so, the run time error “ID not defined” will be thrown.

- Local target:

name = float x, float y, float z, float u, float v, float w

Example:

```
my_local_target = 250,0,150,180,0,180
```

5.5. Behaviour

Local targets have to be declared in the program from they will be called using the DIM declaration and can be only seen from it.

Example:

```
DIM point1 AS TARGET
```

It is possible to have up to one-dimension array of local targets by defining the size of it in the declaration statement.

Example:

```
DIM point4 AS TARGET(10)
```

Global target (GTA) is a fixed array of 1000 values and can be defined and called from all the programs.

By default, all GTAs are deactivated until are declared and executed. Even if they are declared but not executed, they will not be seen from any command, throwing a run time error “Target point not activated”.

It is possible to have read and write access to a bit part of a GTA or local target using “.” at the end of the command. It could be x, y, z, u, v, w, id or active.

Keyword	Description
`x, y, z, u, v, w`	64bit floating point number
`Id (GTA only)`	String name
`Active (GTA only)`	Boolean

Example:

```
GTA(10).x = 250
my_variable = GTA(10).x
```

Throught the Boolean property “active”, it is possible to activate or deactivate and clear a GTA. If active property is assigned to false, the GTA will be erased (all values to 0 and empty name) and deactivated (not seen from any program).

Example:

```
GTA(10).active = FALSE
```

A range of GTAs can be requested or assigned by specifying the start and end index separated by comma.

Example:

```
GTA(10,13) = 250,0,150,180,0,180
PRINT GTA(10,13)
Terminal:
      250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

GTA(10,12).y = 35
PRINT GTA(10,13)
Terminal:
      250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""

GTA(20,23) = GTA(10)
PRINT GTA(20,23)
Terminal:
      250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,35.00000,150.00000,180.00000,0.00000,180.00000,""
      250.00000,0.00000,150.00000,180.00000,0.00000,180.00000,""
```

If the name parameter is specified range assignment cannot be done due to incompatibility of duplicated identified names.

Assignments between GTA – GTA, GTA – Local Target and Local target – Local target is possible.

Example:

```
GTA(10).x = GTA(11).x
my_target = GTA(10)
```

5.1. Robot Position

The current position of the robot is stored in two system variables, ROBOT_LPOS and ROBOT_JPOS.

5.1.1. ROBOT_LPOS

ROBOTL_POS is of datatype TARGET and stores the cartesian position of the end effector. Individual elements can be accessed, for example, TARGET_LPOS.x will return the current X position of the end effector. This is a read-only system variable.

This can be used in programs to know the position of the robot.

Example:

```
DIM my_variable AS TARGET
DIM x_pos AS FLOAT
my_variable = ROBOT_LPOS
x_pos = my_variable.x
```

5.1.2. ROBOT_JPOS

ROBOTJ_POS is of datatype TARGET and stores the joint position of the end effector. Individual joint positions can be accessed, for example, TARGET_JPOS.x will return the current position of Joint 1 of the robot. This is a read-only system variable.

This can be used in programs to know the position of the axes of the robot.

Example:

```
DIM my_variable AS TARGETJ
DIM j3_pos AS FLOAT
my_variable = ROBOT_JPOS
x_pos = my_variable.z
```

6. Move commands

6.1. General overview

RPS has a group of move commands by which is possible to move a robot from one position to another in multiples ways.

It is possible to modify the behaviour of the movement according to some parameters. Those parameters could be set embedded into the move instruction or as a standalone command. As standalone commands overwrite the default ones and are applied to all subsequent motion instructions, so all the moves after that action will be performed with the new parameter value. As embedded parameter only affect to the belonging move instruction. If a move instruction has no parameter or just a few it will use the default values for any missing parameter. The order of embedded parameters does not affect to the move instruction.

Standalone	Embedded parameters	Parameter Range
WORLD_POS_SPEED: linear speed mm/s	S	MOVEL :[0, WORLD_POS_MAX_SPEED] MOVEC :[0, WORLD_POS_MAX_SPEED]
WORLD_ORI_SPEED: joint speed deg/s	S	MOVEJ :[0, WORLD_ORI_MAX_SPEED]
WORLD_MAX_ACC	A	Percentage of WORLD_MAX_ACC [1 - 100]
WORLD_MAX_ACC	D	Percentage of WORLD_MAX_ACC [1 – 100]
TOOL_OFFSET	T	[0 - 31]
OBJECT_FRAME	O	[0 - 31]

ROBOT CONFIGURATION	C	SCARA : {0, 1}
		6 Axis Anthropomorphic : {0,1,2,3,4,5,6,7}
		Palletizing : {0}

PRECISION_ZONE	ZF: orientation changes from start until the end of blend curve. ZT: orientation changes along blend curve.	[0 - Length of move]

Interrupt by path percentage	IP	[0 - 200]
Interrupt by input number	II	Input number depending on the controller.
Interrupt by time	IT	[0, ∞) mS
Routine Output	RO	Output number depending on the controller

Speed

Speed can be specified as default parameter for linear and for orientation movements, using WORLD_POS_SPEED for linear speed in mm/s and WORLD_ORI_SPEED in deg/s for joint speed. Its embedded parameter is ‘S’ and it will mean linear speed if it is used in linear movements, such as MOVEL, MOVEC or MOVELREL. In the same way, ‘S’ will mean joint speed if it is used with joint movements, like MOVEJ or MOVEJREL.

Accel

Acceleration is specified as default parameter in ROBOT_DEFINITIONS file through the commands WORLD_POS_MAX_ACC for linear axes and WORLD_ORI_MAX_ACC for orientation axes. Its embedded parameter is ‘A’ and it can be set as a % of the acceleration specified in ROBOT_DEFINITIONS. The values go between 1 (1 % of the specified acceleration) to 100 (100% of the specified acceleration).

Decel

Deceleration is specified as default parameter in ROBOT_DEFINITIONS file through the commands WORLD_POS_MAX_ACC for linear axes and WORLD_ORI_MAX_ACC for orientation axes. Its embedded parameter is ‘D’ and it can be set as a % of the deceleration specified in ROBOT_DEFINITIONS. The values go between 1 (1 % of the specified deceleration) to 100 (100% of the specified deceleration).

Configuration

The robot configuration is always overwritten by the current configuration and it is only possible to use it in MOVEJ. It means, if a MOVEJ command is executed with a different configuration than the current one, the next move instruction will use the new robot configuration. This will prevent singularities because consecutive linear moves cannot have different robot configurations.

Interrupt by path percentage

An interrupt will be generated when the robot reaches the specified path percentage. The interrupt will turn ON / OFF the specified Routine Output.

Example:

```
MOVEL GTA(10) IP:=50 RO:=9,ON
```

In this example, output number 9 will be turned ON when the robot finishes 50% of the move to GTA 10.

The range of this parameter is from 0 to 200. The interrupt can be generated even while the robot is blending two moves. This can be done by specifying IP from 100 to 200.

Example:

```
MOVEL GTA(10) Z:=30 IP:=130 RO:=9,ON
```

In this example, the output 9 will be turned ON when the robot finishes 30% of the blend curve.

If there is no blending in a move and an IP > 100 is specified, no action will be taken.

Interrupt by input number

The specified Routine output will be turned ON / OFF when the input specified in II (input interrupt) gives a rising / falling edge.

Example:

```
MOVEL GTA(10) II:=3,ON RO:=9,ON
```

In this example, when input 3 gives a rising edge (goes from low to high) output 9 will be turned ON while the MOVE is being executed.

If input 3 turns ON after the move finishes, no action will be taken.

Interrupt by time

The specified routine output will be turned ON / OFF after the time specified in IT (input time) is reached.

Example:

```
MOVEL GTA(10) IT:=700 RO:=9,ON
```

In this example, output 9 will be turned ON after 700mS after the move starts.

If the move finishes earlier than 700mS, no action will be taken.

Routine Output

The output specified in RO (routine output) will be turned ON / OFF when any of the conditions specified by IP (interrupt percentage), II (Interrupt input) or IT (interrupt time) is met. If the output is already in the specified status, no further actions will be taken if the condition is meet.

6.2. Precision zone

Precision zone parameter is the degree of approximation of the manipulator's end effector to a taught point.

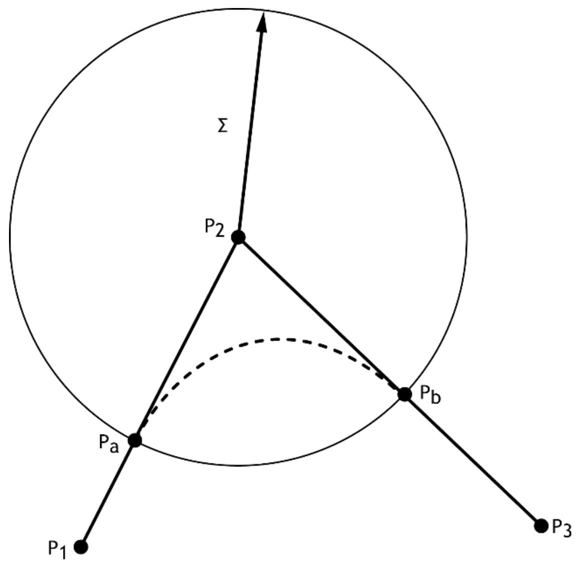


Figure 6-1: precision zone curve

If a point to point trajectory must pass exactly through every point, then the trajectory must stop at each point where the direction of the path changes. Otherwise, there will be discontinuities in the velocity profile of the trajectory. These stops can be avoided by allowing the path to deviate slightly from the points, using blends curves to smooth the path near the point while changing directions but not stopping.

The system allows use this method to smooth transition between any two straight lines generated by MOVEL instructions.

It is not possible to control the transition shape. However, a tightness parameter is provided to measure how closely the trajectory must approach a given point before blending to the nest one.

There are two precision parameters: ZF and ZT. Both specify the tightness of the blend curve in mm but ZF also indicates to the system that the orientation changes will be performed from the start of the straight line until the end of the blend curve, meanwhile ZT indicates that the orientation changes will be done entirely along the blend curve.

6.3. MOVEJ

Definition

MOVEJ is used to move the robot from one point to another along a non-linear path. All axes reach the destination position at the same time. It is the fastest type of movement due to the axes move the exact amount of degrees needed to reach the desired position.

The speed of the move is an interpolated speed and NOT the speed of any individual axis or the end effector.

For Example: a robot has to move from 10.5,0,20,0,0,0 in joint position to 90.21,44.2,24.1,4.1,0.0,12.2 with the speed 50.

The speed 50 will be applied to the total position change, (90.21 - 10.5 + 44.22-0 + 24.1-20 + 4.1-0 + 0-0 + 12.2 - 0) = 144.33

The robot will cover the distance 144.33 with the speed 50. Individual axes may travel at a higher or a lower speed than 50 but all of them will reach end position at the same time.

Syntax

MOVEJ target [S:= T:= O:= C:= Z:= A:= D:=]

Examples

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)

ROBOT(0)
MOVEJ GTA("p1")
MOVEJ GTA("p2")
MOVEJ GTA("p3")
STOP
```

This example commands Robot 0 to Move linearly to target point 0. Note that there are no other embedded instructions in the MOVE. This instruction will MOVE the robot to target point 1 using the following parameters.

Type	Parameter
Object Frame	Currently active object frame

Tool Offset	Currently active tool offset
Speed $[deg/s]$	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR
Acceleration $[deg/s^2]$	WORLD_ORI_MAX_ACC
Jerk $[deg/s^3]$	WORLD_ORI_MAX_JERK
Blend radius $[mm]$	0
Configuration	0

MOVEJ GTA("p1") S:=100

Type	Parameter
Speed$[deg/s]$	100*LOOKAHEAD_FACTOR

MOVEJ GTA("p1") O:="of1"

Type	Parameter
Object Frame	Object Frame 1
Speed$[deg/s]$	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR

Frame 1 is not set as the active object frame. This instruction only takes Frame 1 into consideration for this move

MOVEJ GTA("p1") T:="tool1"

Type	Parameter
Object Frame	Currently active object frame
Tool Offset	Tool Frame 1

Frame 1 is not set as the active tool frame. This instruction only takes Frame 1 into consideration for this move.

```
MOVEJ GTA("p1") Z:=67
MOVEJ GTA("p2")
```

Type	Parameter
Tool Offset	Currently active tool offset
Blend radius[mm]	67mm

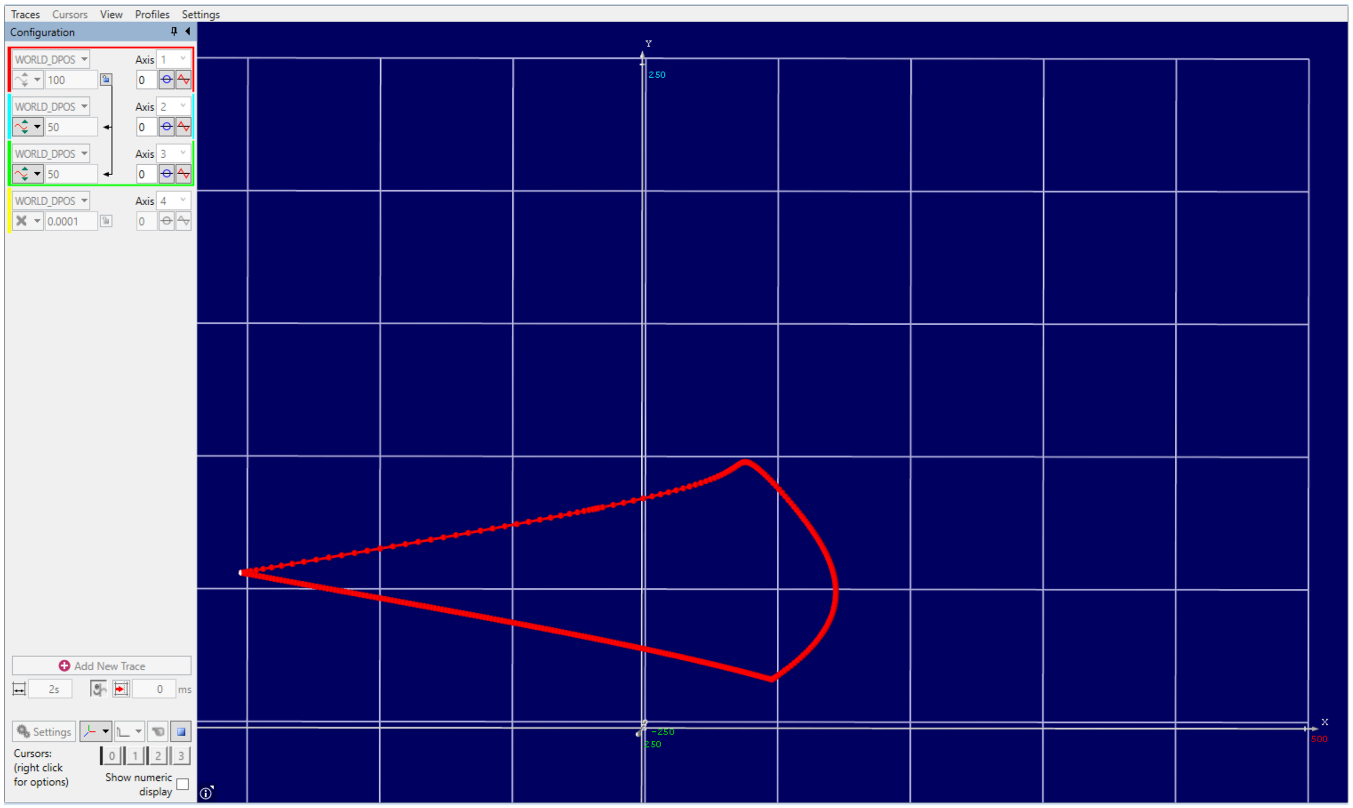
The robot will go towards target point 0 and 67 before the reaching the target point, it will blend the next move and go to target point 1.

```
MOVEJ GTA("p1") S:=150 O:="of1" T:="tool1" Z:=42 C:=0
MOVEJ GTA("p2") S:= 100 C:=1
```

Type	Parameter for MOVE GTA(0)	Parameter for MOVE GTA(1)
Object Frame	Object Frame 1	Currently active object frame
Tool Offset	Tool Offset 1	Currently active tool offset
Speed	150*LOOKAHEAD_FACTOR	100*LOOKAHEAD_FACTOR
Acceleration	WORLD_POS_MAX_ACC*JOG_ACC_FACTOR	WORLD_POS_MAX_ACC*JOG_ACC_FACTOR
Jerk	WORLD_POS_MAX_JERK	WORLD_POS_MAX_JERK
Blend radius	42mm	0
Configuration	0	1

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)

ROBOT(0)
MOVEJ GTA("p1") S:=80 O:=1 T:=1 Z:=159 C:=1
MOVEJ GTA("p2") S:= 60 C:=0
MOVEJ GTA("p3")
STOP
```



WORLD_DPOS	100	Axis 1	0
WORLD_DPOS	50	Axis 2	0
WORLD_DPOS	50	Axis 3	0
WORLD_DPOS	0.0001	Axis 4	0

2s 0 ms

Figure 6-2: Trace of Example Program

6.4. MOVEL

Definition

MOVEL is used to move the robot from one point to another along a linear path. The end effector moves along the straight line between current point and target point in cartesian space. All axes reach the destination position at the same time.

Syntax

MOVEL target [S:= T:= O:= ZT:= ZF:= A:= D:=]

Examples

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)

ROBOT(0)
MOVEL GTA("p1")
MOVEL GTA("p2")
MOVEL GTA("p3")
STOP
```

This example commands Robot 0 to Move linearly to target point 0. Note that there are no other embedded instructions in the MOVE. This instruction will MOVE the robot to target point 1 using the following parameters.

Type	Parameter
Object Frame	Currently active object frame
Tool Offset	Currently active tool offset
Speed [mm/s]	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR
Acceleration [mm/s^2]	WORLD_ORI_MAX_ACC*JOG_ACC_FACTOR
Jerk [mm/s^3]	WORLD_ORI_MAX_JERK
Blend radius [mm]	0

MOVEL GTA("p1") S:=100

Type	Parameter
Speed[mm/s]	100*LOOKAHEAD_FACTOR

MOVEL GTA("p1") O:="of1"

Type	Parameter
Object Frame	Object Frame "of1"
Speed[mm/s]	WORLD_ORI_SPEED *LOOKAHEAD_FACTOR

Frame 1 is not set as the active object frame. This instruction only takes Frame 1 into consideration for this move

MOVEL GTA("p1") T:="tool1"

Type	Parameter
Object Frame	Currently active object frame
Tool Offset	Tool Frame "tool1"

Frame 1 is not set as the active tool frame. This instruction only takes Frame 1 into consideration for this move.

MOVEL GTA("p1") Z:=67
MOVEL GTA("p1")

Type	Parameter
Tool Offset	Currently active tool offset
Blend radius[mm]	67mm

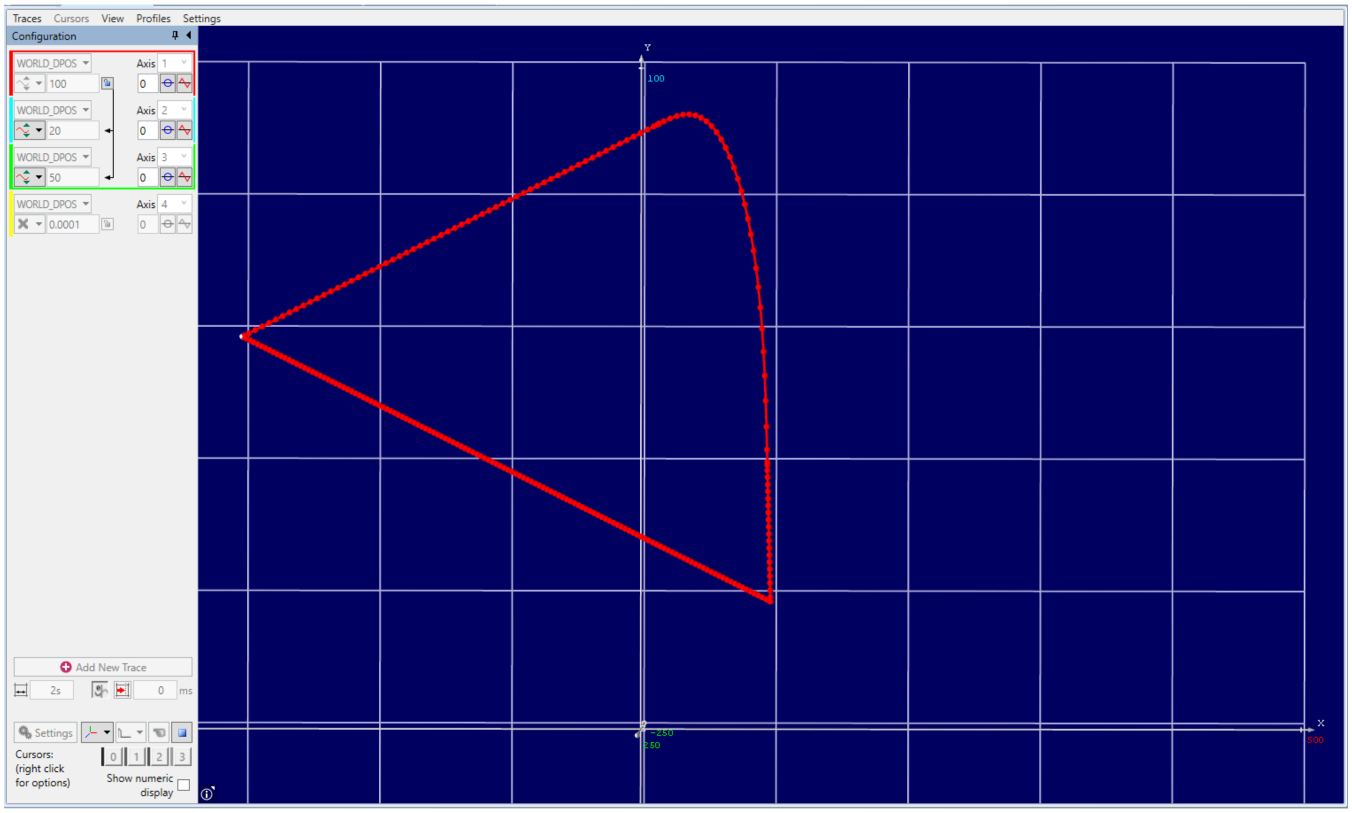
The robot will go towards target point 0 and 67 before the reaching the target point, it will blend the next move and go to target point 1.

MOVEL GTA("p1") S:=150 O:="of1" T:="tool1" Z:=42
MOVEL GTA("p1") S:= 100

Type	Parameter for MOVE GTA(0)	Parameter for MOVE GTA(1)

Object Frame	Object Frame 1	Currently active object frame
Tool Offset	Tool Offset 1	Currently active tool offset
Speed	150*LOOKAHEAD_FACTOR	100*LOOKAHEAD_FACTOR
Acceleration	WORLD_POS_MAX_ACC*JOG_ACC_FACTOR	WORLD_POS_MAX_ACC*JOG_ACC_FACTOR
Jerk	WORLD_POS_MAX_JERK	WORLD_POS_MAX_JERK
Blend radius	42mm	0

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1, -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1, 0, 0, 50, 0, 0, 0)
ROBOT(0)
MOVEL GTA("p1") S:=280 O:=1 T:=1 Z:=80
MOVEL GTA("p2") S:=360
MOVEL GTA("p3")
STOP
```



WORLD_DPOS Axis 1 100 0
WORLD_DPOS Axis 2 20 0
WORLD_DPOS Axis 3 50 0
WORLD_DPOS Axis 4 0.0001 0
2s 0 ms

Figure 6-3: Trace of Example Program

6.5. MOVEC

Definition

MOVEC is used to move the robot from one point to another along a circular path. All axes reach the destination position at the same time.

Syntax

MOVEC target (middle point) target (end point) [S:= T:= O:= A:= D:=]

Examples

```
GTA(0) = 350, 0, 50, 180, 0, 0
GTA(1) = 400, 100, 20, 180, 0, 0
GTA(2) = 450, 0, 60, 180, 0, 0
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)

ROBOT(0)
MOVEC GTA("p1") GTA("p2")
MOVEL GTA("p3")
STOP
```

This example commands Robot 0 to Move linearly to target point 0. Note that there are no other embedded instructions in the MOVE. This instruction will MOVE the robot to target point 1 using the following parameters.

Type	Parameter
Object Frame	Currently active object frame
Tool Offset	Currently active tool offset
Speed [mm/s]	WORLD_ORI_SPEED*LOOKAHEAD_FACTOR
Acceleration [mm/s^2]	WORLD_ORI_MAX_ACC*JOG_ACC_FACTOR
Jerk [mm/s^3]	WORLD_ORI_MAX_JERK
Blend radius [mm]	0

```
MOVEC GTA("p1") GTA("p2") S:=100
```

Type	Parameter
Speed[mm/s]	100*LOOKAHEAD_FACTOR

MOVEC GTA("p1") GTA("p2") O:="of1"

Type	Parameter
Object Frame	Object Frame 1
Speed[mm/s]	WORLD_ORI_SPEED *LOOKAHEAD_FACTOR

Frame 1 is not set as the active object frame. This instruction only takes Frame 1 into consideration for this move

MOVEC GTA("p1") GTA("p2") T:="tool1

Type	Parameter
Object Frame	Currently active object frame
Tool Offset	Tool Frame 1

Frame 1 is not set as the active tool frame. This instruction only takes Frame 1 into consideration for this move.

MOVEC GTA("p1") GTA("p2") Z:=67
MOVEL GTA("p3")

Type	Parameter
Tool Offset	Currently active tool offset
Blend radius[mm]	67mm

The robot will go towards target point 0 and 67 before the reaching the target point, it will blend the next move and go to target point 1.

MOVEC GTA("p1") GTA("p2") S:=150 O:="of1" T:="tool1" Z:=42
MOVEL GTA("p3") S:= 100

Type	Parameter for MOVE GTA(0)	Parameter for MOVE GTA(1)
Object Frame	Object Frame 1	Currently active object frame
Tool Offset	Tool Offset 1	Currently active tool offset
Speed	150*LOOKAHEAD_FACTOR	100*LOOKAHEAD_FACTOR
Acceleration	WORLD_POS_MAX_ACC*JOG_ACC_FACTOR	WORLD_POS_MAX_ACC*JOG_ACC_FACTOR
Jerk	WORLD_POS_MAX_JERK	WORLD_POS_MAX_JERK
Blend radius	42mm	0

```
GTA(0) = 350, 0, 50, 180, 0, 0, "p1"
GTA(1) = 400, 100, 20, 180, 0, 0, "p2"
GTA(2) = 450, -300, 60, 180, 0, 0, "p3"
OBJECT_FRAME(1,"of1", -10, 0, 0, 0, 0, 15)
TOOL_OFFSET(1,"tool1", 0, 0, 50, 0, 0, 0)

ROBOT(0)
MOVEC GTA("p1") GTA("p2") S:=150 O:="of1" T:="tool1"
MOVEL GTA("p3") S:= 100
STOP
```

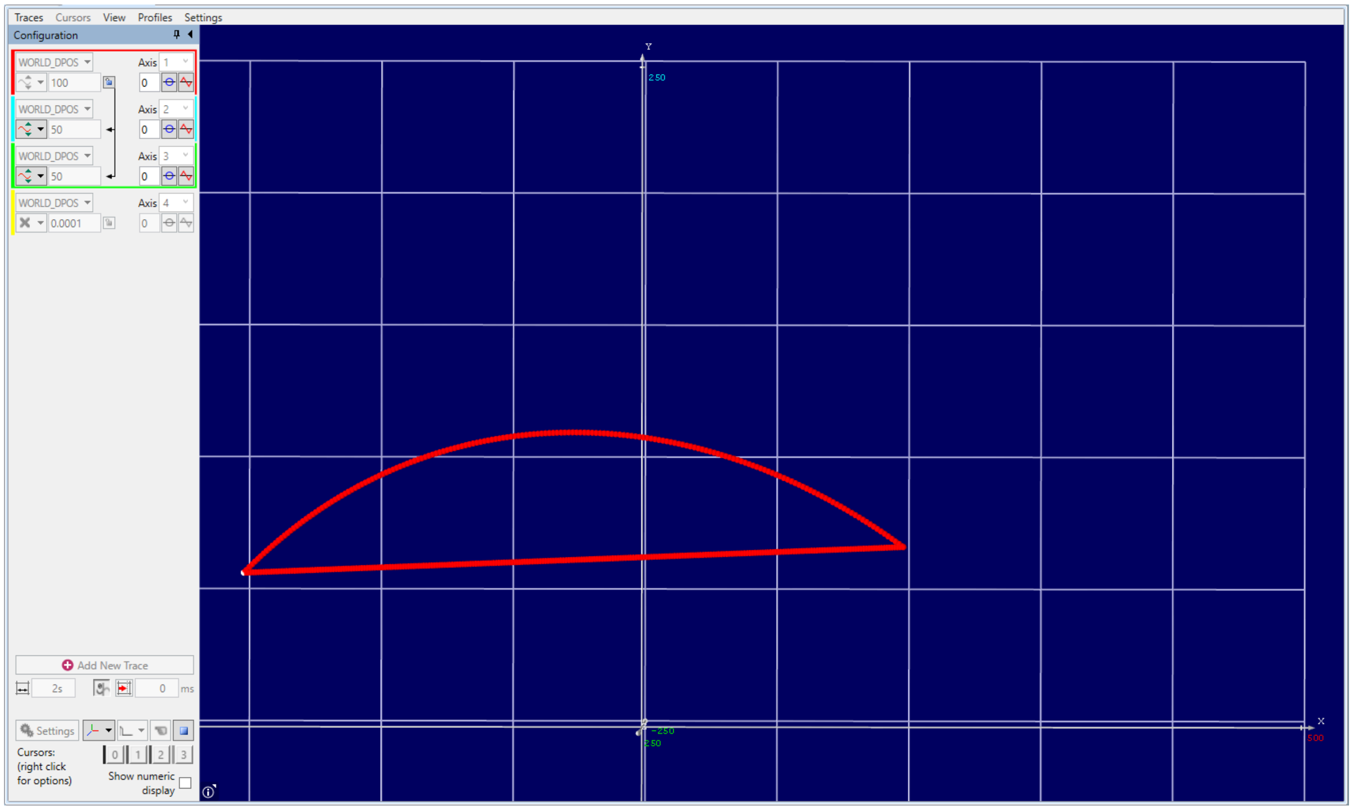


Figure 6-4: Trace of Example Program

6.6. MOVEJREL

Definition

MOVEJREL is used to move the robot from one point to another, relative to the starting position, along a non-linear path. All axes reach the destination position at the same time. It is the quickest type of movement due to the axes move the exact amount of degrees needed to reach the desired position.

The speed of the move is an interpolated speed and NOT the speed of any individual axis or the end effector.

A single joint can be moved by simply set all values of the target as 0 except the desired joint to move.

Syntax

MOVEJREL target [S:= T:= O:= C:= Z:= A:= D:=]

Examples

```
GTA(0) = 0,-20,0,0,0,0

ROBOT(0)
MOVELREL GTA(0) 'The robot will move 20 degrees of the second joint in negative direction from its
current position
STOP
```

6.7. MOVELREL

Definition

MOVELREL is used to move the robot from one point to another, relative to the starting position, along a linear path. The end effector moves along the straight line between current point and target point in cartesian space. All axes reach the destination position at the same time.

A single vector can be moved by simply set all values of the target as 0 except the desired vector to move.

Syntax

MOVELREL target [S:= T:= O:= ZT:= ZF:= A:= D:=]

Examples

```
GTA(0) = 10,0,0,0,0,0

ROBOT(0)
MOVELREL GTA(0) 'The robot will move 10mm in X from its current position

STOP
```

7. Jogging

7.1. Definition

Jog is the manual process of move the axes of the robot in positive or negative direction, or moves the TCP along the specified Cartesian direction.

7.2. Syntax

JOG_OPERATION(mode)

Default jog mode is 0, it means disable.

7.3. Behaviour

To jog the robot is needed to set a FWD_IN and REV_IN for each robot axis.

Inputs used for FWD_IN and REV_IN are active low.

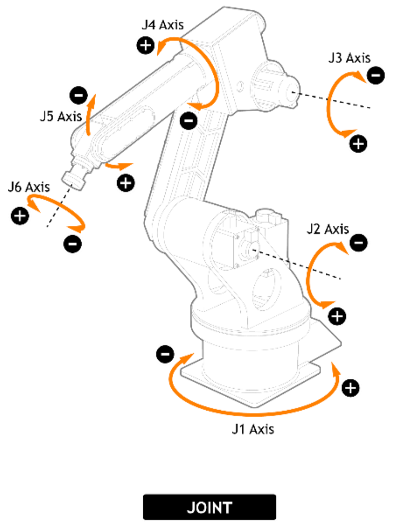
After setting all the inputs, jog mode has to be set using JOG_OPERATION command. Change the jog mode to activate the process and use FWD_IN and REV_IN inputs to jog the robot. Set jog mode back to 0 to release jog process for the execution of move instructions.

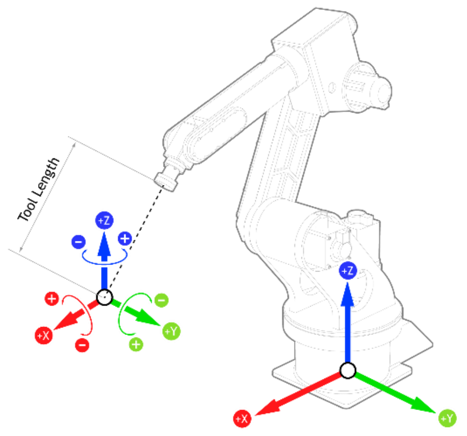
Jog cannot be performed until axes are idle.

JOG_OPERATION command has to be executed in the same base array of the robot.

There are five modes to jog the robot:

- JOINT (mode = 1): each axis moves independently.

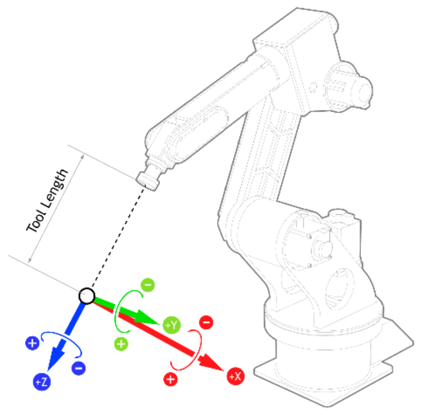




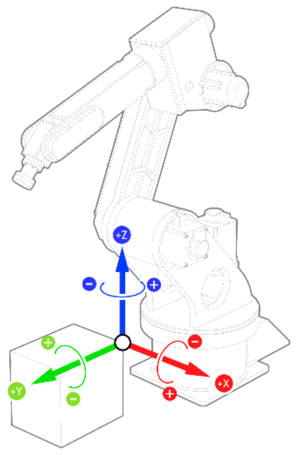
BASE / WORLD

- WORLD (mode = 2): the end effector moves straight along the world coordinate system. The orientation use extrinsic rotations.
- BASE (mode = 3): the end effector moves straight along the base coordinate system. The orientation use extrinsic rotations.

- TOOL jog (mode = 4): the end effector moves straight along the tool coordinate system. The orientation use intrinsic rotation.



TOOL



OBJECT

- OBJECT (mode = 5): the end effector moves straight along the active object frame. The orientation use extrinsic rotations.

Extrinsic rotations are elemental rotations that occur about the axes of the fixed coordinate system XYZ. The XYZ system rotates, while XYZ is fixed. Intrinsic rotations are elemental rotations that occur about the axes of the rotating coordinate system XYZ, which changes its orientation after each elemental rotation.

8. Singularities

In a singular position, the end effector cannot move in certain direction, thus the manipulator loses one or more degrees of freedom (DOF). Furthermore, near singular configuration, some axes can move to a very large speed in order to maintain the end effector speed constant.

In a 6 DOF robot arm, the most common singular positions are described below:

- Wrist singularity occurs when the wrist and the first orientation axis are collinear.

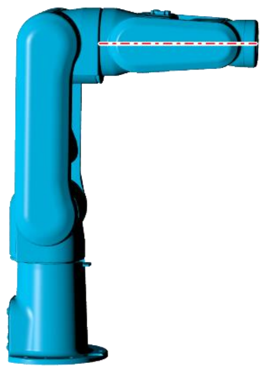


Figure 8-1: Wrist singularity.

- Elbow singularity occurs when the arm is fully extended.

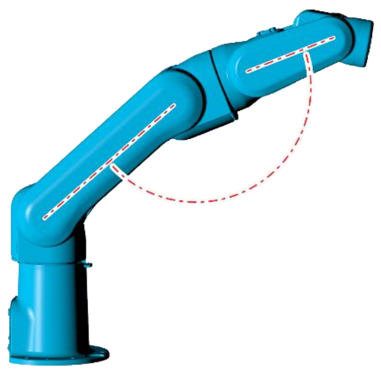


Figure 8-2: Elbow singularity.

- Alignment singularity occurs when the wrist and the base are aligned.

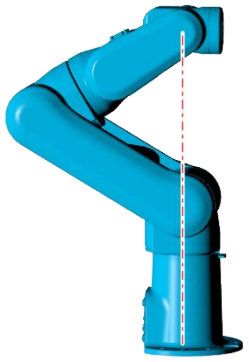


Figure 8-3: Alignment singularity.

If a singular position is reached or is close enough and the system is not in joint mode, a ROBOTSTATUS error will be triggered at the involved axis and the robot will stop at the configured WORLD_FASTDEC.

To leave the singular position the system has to be in joint mode.

9. Robot configurations

The system can solve the kinematics for a particular position and orientation of the TCP in up to eight different axis configurations. The different configurations can be specified by the embedded motion parameter "C".

C parameter can be only used in MOVEJ instruction with values from 0 to 7.

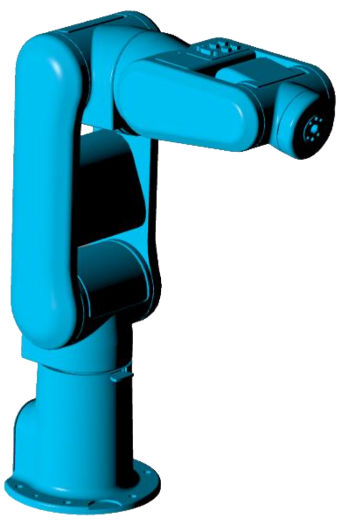


Figure 9-1: C := 0, Wrist no flip, elbow above, base forward.

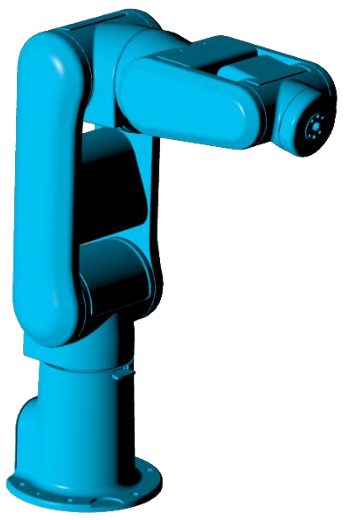


Figure 9-2: C := 1, Wrist flip, elbow above, base forward.

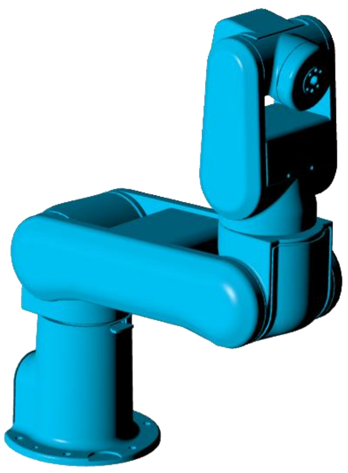


Figure 9-3: C := 2, Wrist no flip, elbow below, base forward.

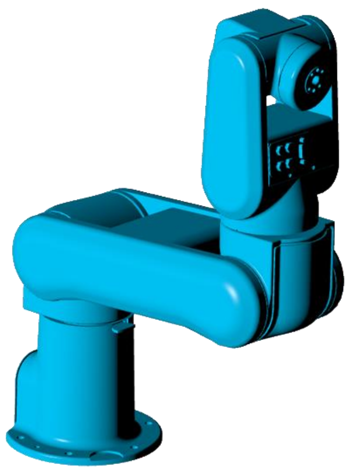


Figure 9-4: C := 3, Wrist flip, elbow below, base forward.

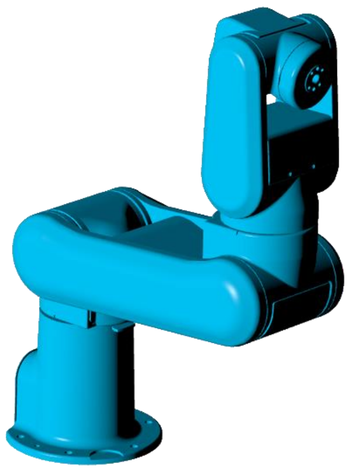


Figure 9-5: C := 4, Wrist no flip, elbow below, base backward.

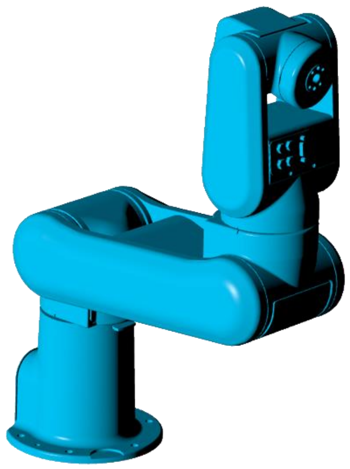


Figure 9-6: C := 5, Wrist flip, elbow below, base backward.

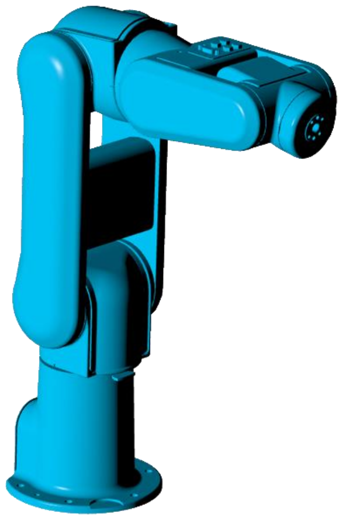


Figure 9-7: C := 6, Wrist no flip, elbow above, base backward.

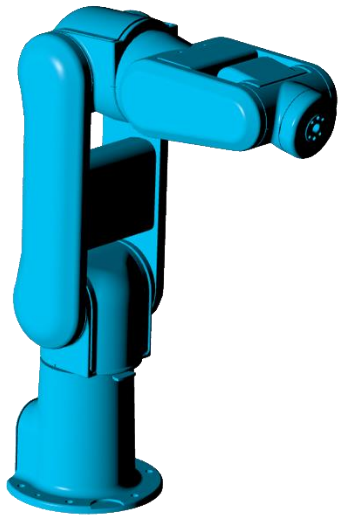


Figure 9-8:C := 7, Wrist flip, elbow above, base forward.

10. Limits

10.1. Definition

All the next limits can be scaled by its respective UNITS.

Limits in axis joints:

AXIS_FS_LIMIT: AXIS_DPOS is greater than AXIS_FS_LIMIT. This refers to joint angle limit.

AXIS_RS_LIMIT: AXIS_DPOS is greater than AXIS_RS_LIMIT. This refers to joint angle limit.

AXIS_MAX_SPEED: Maximum allowed axis speed. If the actual speed exceeds this value, error 'max speed limit exceeded' will be triggered.

AXIS_MAX_ACC: Maximum allowed axis acceleration/deceleration.

AXIS_MAX_JERK: Maximum allowed axis jerk.

AXIS_MAX_JOG_SPEED: Maximum allowed jog speed.

AXIS_MAX_JOG_ACC: Maximum allowed jog acceleration.

Limits in world mode:

WORLD_POS_MAX_SPEED: Maximum linear speed.

WORLD_POS_MAX_ACC: Maximum linear acceleration.

WORLD_POS_MAX_JERK: Maximum linear jerk.

WORLD_ORI_MAX_SPEED: Maximum orientation speed.

WORLD_ORI_MAX_ACC: Maximum orientation acceleration.

WORLD_ORI_MAX_JERK: Maximum orientation jerk.

11. Parasitic Compensation

Some robots have a mechanical link between 2 axes. This causes one axis to move when the other linked axis moves, causing parasitic motion. To compensate this type of motion, COMP_RATIO can be used.

11.1. Syntax

COMP_RATIO(driving_axis, ratio_numerator, ratio_denominator)

11.2. Description

This function links the DEMAND_EDGES of base axis to the measured movements of the driving axes to compensate mechanical parasitic motion. It will add the compensation pulses to demand edges. When this function is called, an initial compensation amount will be added to AXIS_DPOS to account for previous compensation. No motion will occur during this.

COMP_RATIO(-1) will delete any existing COMP_RATIO applied to the base axis.

11.3. Parameters

Standalone parameters	Embedded parameters
`Axis`	This parameter specifies the axis to link to.
`Ratio numerator`	This parameter holds the number of edges the base axis is required to move per increment of the driving axis. The ratio numerator value can be either positive or negative. The ratio numerator is always specified as an encoder edge ratio.
`Ratio denominator`	This parameter holds the number of edges the base axis is required to move per increment of the driving axis. The ratio denominator value can be either positive or negative. The ratio denominator is always specified as an encoder edge ratio.

Example

```
master_axis = 3
slave_axis = 2
compensation_value = -18
num = UNITS AXIS(slave_axis) * compensation_value
denom = UNITS AXIS(master_axis) * 360

COMP_RATIO(master_axis,num,denom) AXIS(slave_axis)
```

12. Errors

12.1. AXISSTATUS bit definitions

Keyword	Bit
AS_LOOKAHEAD_OVERRIDE	0
AS_FE_WARNING	1
AS_COMMS_ERROR	2
AS_REMOTE_DRIVE_ERROR	3
AS_IN_FORWARD_LIMIT	4
AS_IN_REVERSE_LIMIT	5
AS_DATUMING	6
AS_FEEDHOLD	7
AS_FE_EXCEEDS_LIMIT	8
AS_FORWARD_SOFTWARE_LIMIT	9
AS_REVERSE_SOFTWARE_LIMIT	10
AS_CANCELLING_MOVE	11
AS_PULSE_OVER_SPEED	12
AS_OVERRIDE_TO_ZERO	13
AS_MOVE_CANCEL_DONE	14
AS_VOLUME_LIMIT	15
AS_AXIS_FS_LIMIT	16
AS_AXIS_RS_LIMIT	17
AS_ENCODER_OVER_I	18
AS_PSWITCH_FIFO_NOT_EMPTY	19
AS_PSWITCH_FIFO_FULL	20
AS_KINEMATICS_1HOUR	21
AS_BISS_CLEAN_WARNING	22
AS_WORLD_FS_LIMIT	23

AS_WORLD_RS_LIMIT	24
AS_CANCEL_MODE_3	25
AS_REMOTE_NODE_STATUS_ERR	26

12.2. ROBOTSTATUS bit definitions

Keyword	Bit
RS_WORLD_FS_LIMIT	0
RS_WORLD_RS_LIMIT	1
RS_ROBOT_FS_LIMIT	2
RS_ROBOT_RS_LIMIT	3
RS_TCP_FS_LIMIT	4
RS_TCP_RS_LIMIT	5
RS_FE_WARNING	6
RS_FE_EXCEEDS_LIMIT	7
RS_WRIST_SINGULARITY	8
RS_ALIGNMENT_SINGULARITY	9
RS_ELBOW_SINGULARITY	10
RS_MAX_SPEED_LIMIT	11
RS_COLLIDED	12

12.3. Robot log

A log system is provided to store in flash events that can happen at any time. Up to 2048 entries are stored as a cyclic buffer.

The stored data is compound by: entry index, date and time of the event, controller up time and message.

Syntax

ROBOT_LOG([Message] | [Range of messages to show])

The messages can be shown as follows:

- No parameters: the whole buffer.

Example:

```
Terminal
robot_log
   1 23/Jan/2018 13:15:10 [    4618.272] Wrist Singularity axis [4]
   2 23/Jan/2018 13:15:12 [    4620.961] Blend max radius reached: 89
Units
   3 23/Jan/2018 13:15:13 [    4620.963] Blend max radius reached: 121
Units
   4 23/Jan/2018 13:15:13 [    4621.130] Encompassed point:
350,150,550,-120,16,-100
   5 23/Jan/2018 13:15:14 [    4622.739] Wrist Singularity axis [4]
   6 23/Jan/2018 13:15:14 [    4622.739] Max speed reached: 356.445313
Units/s in axis 5
<END OF LOG>
```

- One parameter: from that entry until the end.

```
Terminal
robot_log(3)
   3 23/Jan/2018 13:15:13 [    4620.963] Blend max radius reached: 121
Units
   4 23/Jan/2018 13:15:13 [    4621.130] Encompassed point:
350,150,550,-120,16,-100
   5 23/Jan/2018 13:15:14 [    4622.739] Wrist Singularity axis [4]
   6 23/Jan/2018 13:15:14 [    4622.739] Max speed reached: 356.445313
Units/s in axis 5
<END OF LOG>
```

- Two parameters: from the fist to the last parameter.
  Terminal

```
robot_log(3,5)
   3 23/Jan/2018 13:15:13 [    4620.963] Blend max radius reached: 121
Units
   4 23/Jan/2018 13:15:13 [    4621.130] Encompassed point:
350,150,550,-120,16,-100
   5 23/Jan/2018 13:15:14 [    4622.739] Wrist Singularity axis [4]
```

It is possible to remove all entries using parameter -1.

ROBOT_LOG(-1)

Store user messages can be done by entering a string parameter:

ROBOT_LOG(“My message”)

13. Collision detection

13.1. Definition

A collision between an Oriented Bounding Box (OBB specified by COLLISION_OBJECT) around a real object and the OBB specified in TOOL_COLLISION can be detected and controlled by the collision detection algorithm.

The distance between the TOOL_COLLISION and the active COLLISION_OBJECT(s) is checked every servo period and the robot will decelerate at FAST_DEC in the case of the collision distance is smaller than the deceleration distance.

It is recommended to specify the COLLISION_OBJECT and the TOOL_COLLISION slightly bigger than the real object to have a margin where the tool will stop.

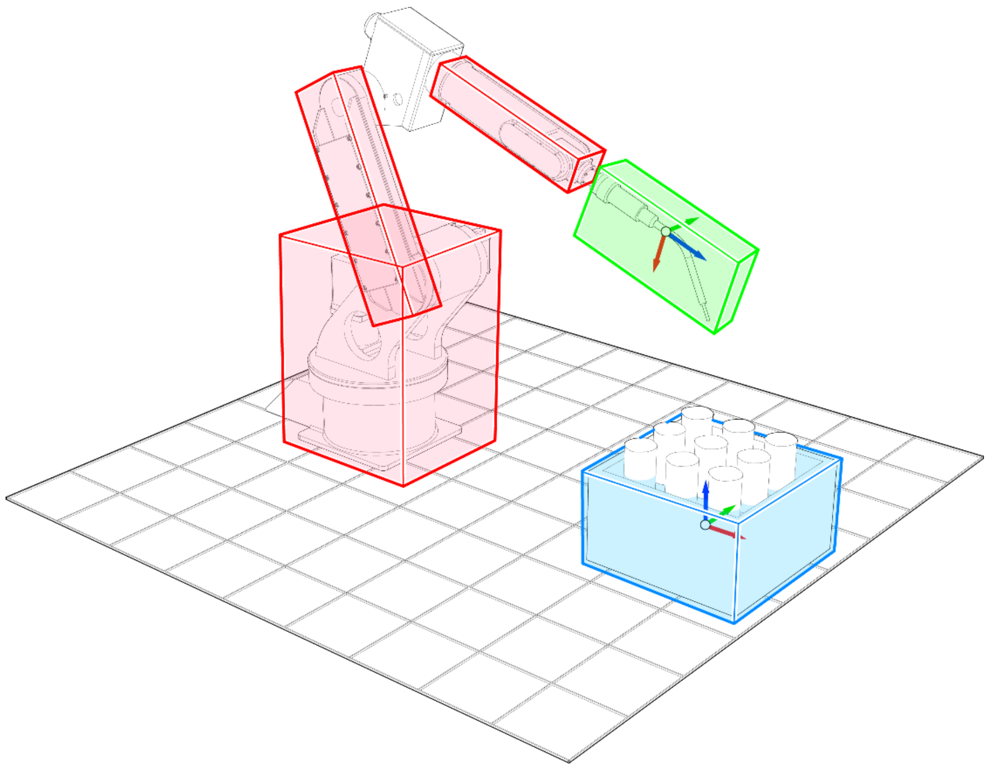


Figure 13-1: Collision detection scenario.

Tool collision object is depicted in green. Collision object is depicted in blue and OBBs bounding robot links are in red.

The collision algorithm measures the distance between the tool collision object and collision objects and OBBs of the robot links. If the distance is less than the fast deceleration distance then, the robot will stop securely to avoid the collision. Be aware that the algorithm uses the tool collision against other objects, the OBBs of the robot links can collide with collision objects.

13.2. Syntax

COLLISION_OBJECT(identity, name, x centre, y centre, z centre, x rotation, y rotation, z rotation, x half distance, y half distance, z half distance)

X centre, y centre and z centre have to be specified in mm.

X rotation, y rotation and z rotation have to be specified in degrees.

X half distance, y half distance and z half distance have to be specified in mm.

Keyword	Description
`x, y and z centres`	Centre position of the OBB on the object.
`x, y and z rotations`	Orientation of the OBB on the object.
`x, y and z half distances`	Half distances measured in every vector of the OBB.

This draw depicts the parameters of a Collision Object. The frame arrows are in the centre of the object representing the position and orientation of the object.

Half distances measured in the different vectors are shown with the same colours of their vectors.

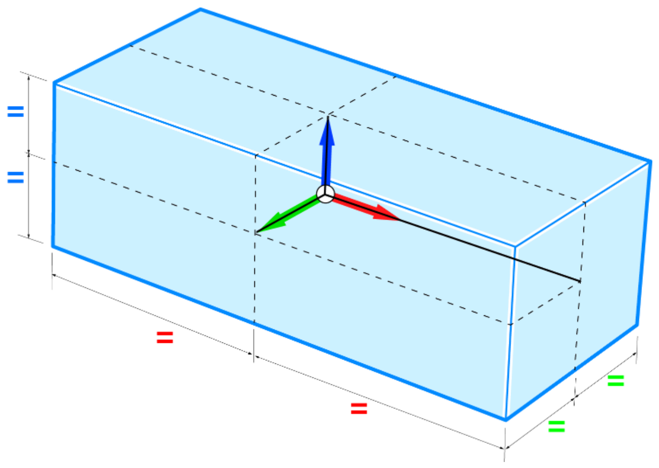


Figure 13-2: Collision object definition

A collision object can be activated or deactivated as follows:

- Activate it: COLLISION_OBJECT(identity, ON)
- Deactivate it: COLLISION_OBJECT(identity, OFF)

The same COLLISION_OBJECT can be activated for some kinematic groups and deactivate for some others.

Maximum number of COLLISION_OBJECTs active is 10.

To know what objects are active just simply print KINEMATIC_GROUP.

It cannot be edited if it is active in any kinematic group.

Remove an entry is possible using the parameters -2 and index or just one parameter (-2) to remove all of them.

Example:

```
COLLISION_OBJECT(-2)
```

Request for existing entries is possible with parameter -1. If the parameters are (-1, index) it returns only the requested index.

Example:

```
COLLISION_OBJECT(-1,2)
Terminal:
2 [object2] : 350.00000, 0.00000, 50.00000, 0.00000, 0.00000, 150.00000,
50.00000, 50.00000, 50.00000
```

14. Synchronisation

It is possible to synchronise the robot TCP with a moving object or with another machine or robot. This means the robot TCP will move or rotate in the same directions than the synchronized object and absolute movements on top of it.

There are two methods to synchronise the robot TCP: synchronisation with an object frame and synchronization with a GTA.

In order to synchronize with an object frame, it has to be attached to some axis first.

As synchronisation and attachment do not get loaded in to the move buffer they are not cancelled by CANCEL or RAPIDSTOP , a desynchronization or detachment has to be performed. When a software or hardware limit is reached synchronisation and attachment are immediately stopped with no deceleration.

14.1. Attach Object Frame

Attach Object Frame is a method to link the vectors of an Object Frame with up to six axes and place it in a specific position. The object frame will move then according to those axes. There are two different modes to attach an object frame; one specially designs to work with a conveyor and another one to work with a multipurpose machine.

Before attach it, the Object Frame has to be created.

Attach Object Frame with a conveyor

As the conveyor can be in different orientations, the object frame can be set in multiples orientations to match the conveyor. The object frame vector selected will follow the conveyor vector direction.

Syntax

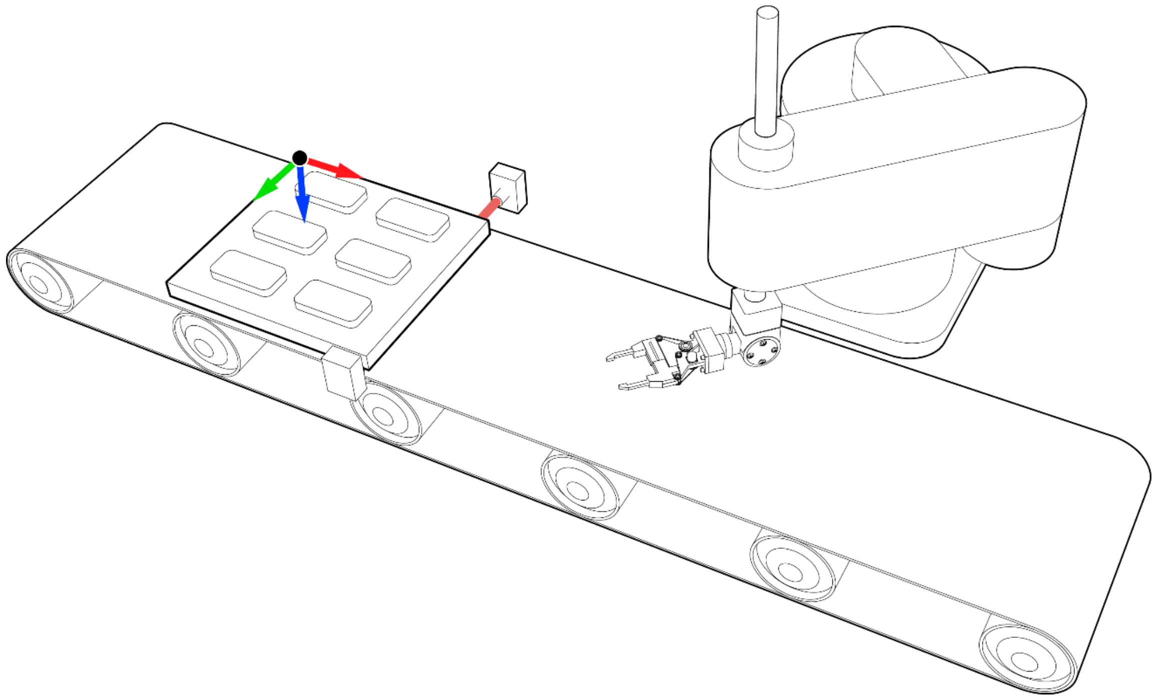
ATTACH_OBJECT_FRAME(control, object frame, syncVector, syncPos, syncAxis)

Parameter	Description
`Control`	Select the mode ATTACH_OBJECT_FRAME will work. Value = 1.
`Object Frame`	Object Frame that will be attached. It could be an index or name.
`syncVector`	Object Frame vector that will be attached to syncAxis. There are 3 possible values: 1: X, 2: Y, 3: Z
`syncPos`	Captured position on syncAxis.
`syncAxis`	Axis to synchronise with.

Example:
A common industry example is pick objects from a pallet and place them onto another station or in a package. The synchronization could be simplified by using Attach Object Frame technique. An Object Frame will be placed on the pallet in a specific position. All the objects will be taught related to the Object Frame "pallet". The Object Frame "pallet" will be attached to the conveyor using one of the three possible options: vector X, vector Y or vector Z. Bear in mind that the conveyor could not be perfectly aligned with the robot, so orientation data of the Object Frame could be used to adjust it.

```
ATTACH_OBJECT_FRAME(1,"pallet",1,registed_position,11)
```

After the command is executed, the Object Frame "pallet" will follow the conveyor precisely. At this state, the robot should be synchronised with the Object Frame "pallet". Please, go to the next section to continue with the example.



Attach Object Frame with a multipurpose machine

For machines with more than one axis, it is possible to attach the positions and orientation vectors with up to 6 axes changing the control parameter. If, for instance, an object frame is attached to a 6 DOF robot arm, the position and orientation of the object frame will follow the robot TCP.

Syntax

ATTACH_OBJECT_FRAME(control, object frame, axisX, axisY, axisZ, axisU, axisV, axisW, xOffset, yOffset, zOffset, uOffset, vOffset, wOffset)

Parameter	Description
`Control`	Select the mode ATTACH_OBJECT_FRAME will work. Value = 2.
`Object Frame`	Object Frame that will be attached. It could be an index or name.
`axisX - axisW`	Machine axis number that correspond with the Object Frame vector that will be attached. Set -1 for non-attached vectors.
`xPos - wPos`	Object Frame position and orientation related to the origin of the Robot.

Example:

```
ATTACH_OBJECT_FRAME(2,3,-1,11,-1,-1,-1,-1,400,0,400,0,90,0)
```

Detach Object Frame

Syntax

ATTACH_OBJECT_FRAME(control, object frame)

Parameter	Description
`Control`	Select the mode ATTACH_OBJECT_FRAME will work. Value = 4.
`Object Frame`	Object Frame that will be attached. It could be an index or name.

Example:

```
ATTACH_OBJECT_FRAME(4,"of2")
```

General behaviour

Request for existing entries is possible by using just index as a parameter in the command. It returns true or false depending if the entry is already attached or not.

Example:

```
ATTACH_OBJECT_FRAME("of2")
```

14.2. Sync to object frame

Once an object frame has been attached, it is possible to synchronize the robot TCP to it with the command SYNC_TO_OBJECT_FRAME.

Syntax

SYNC_TO_OBJECT_FRAME(control, time, object frame, GTA)

Parameter	Description
`Control`	Select the mode SYNC_TO_OBJECT_FRAME will work. 1 = Sync, 4 = stop sync, 10 = re-sync to another attached object frame
`Time`	Time to complete the synchronisation in milliseconds.
`Object frame`	Object frame already attached. It could be an index or name.
`GTA`	GTA related to the object frame synchronised. It could be an index or name. Set it as -1 for no GTA

In the case of "Stop sync" the only parameters available are Control and Time
SYNC_TO_OBJECT_FRAME(control, time)

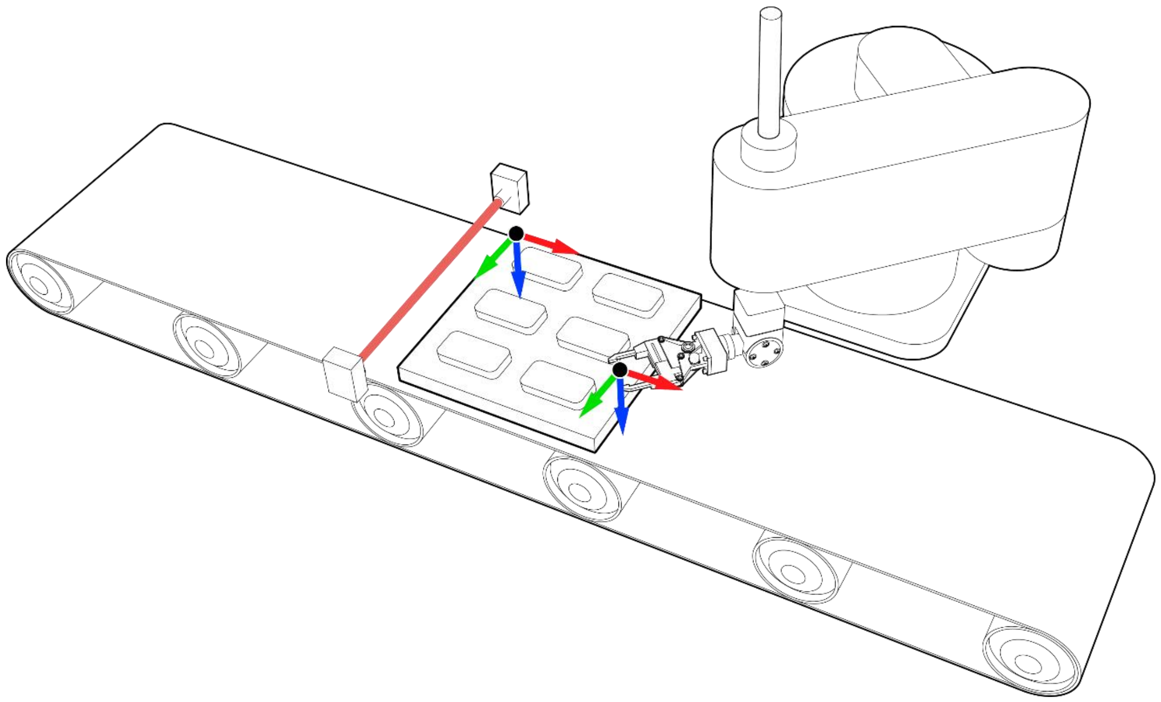
It is possible to check if the robot is already synced by interrogating the command SYNC_CONTROL and the axis of the robot that has been synchronised.

Parameter	Description

SYNC_CONTROL	Desync = 0
	Synced = 3

Example:

```
WAIT UNTIL SYNC_CONTROL axis(2) = 3
```



Example:

After have attached the Object Frame onto the pallet, the robot has to be synchronised with the Object Frame and, in this particular case, placed over a specific object (GTA related to the Object Frame "pallet" previously taught).

```
SYNC_TO_OBJECT_FRAME(1,1000,"pallet","object_1")
```

14.3. Sync to GTA

Sync to GTA has been designed for more simple synchronizations. It synchronizes the robot TCP with one axis in X direction. If the robot is not aligned perfectly with X direction, Object Frame can always be previously set and selected.

Syntax

SYNC_TO_GTA(control, time, syncPos, syncAxis, object frame, GTA)

Parameter	Description
`Control`	Select the mode SYNC_TO_GTA will work. 1 = Sync, 4 = stop sync, 10 = re-sync to another GTA.
`Time`	Time to complete the synchronisation in milliseconds.
`SyncPos`	Captured position on syncAxis.

SyncAxis	Axis to sync with.
Object frame	Object frame related to selected GTA. Set to 0 (default) for no object frame. It could be an index or name.
GTA	GTA to sync with. It could be an index or name. Set it as -1 for no GTA

In the case of "Stop sync" the only parameters available are Control and Time
SYNC_TO_OBJECT_FRAME(control, time)

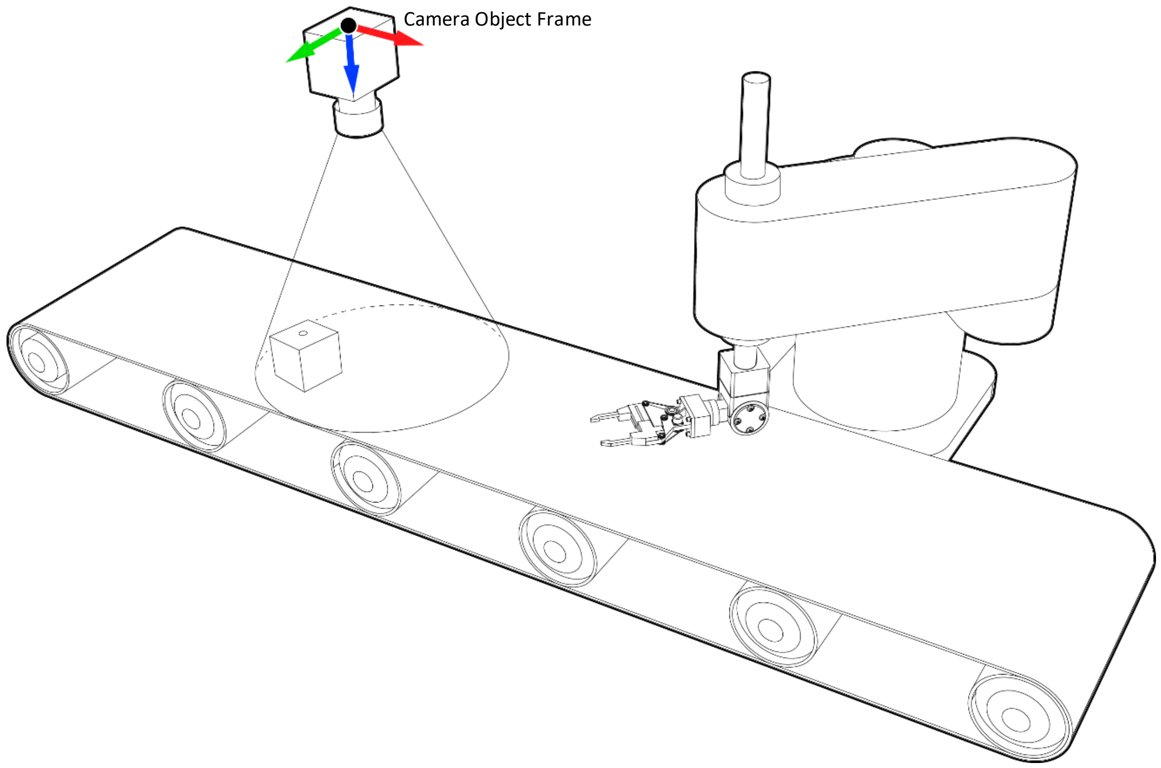
It is possible to check if the robot is already synced by interrogating the command SYNC_CONTROL. In this case, because it is synchronised to axis X, it is not necessary to specify the robot axis.

Parameter	Description

SYNC_CONTROL	Desync = 0
	Synced = 3

Example:

```
WAIT UNTIL SYNC_CONTROL = 3
```

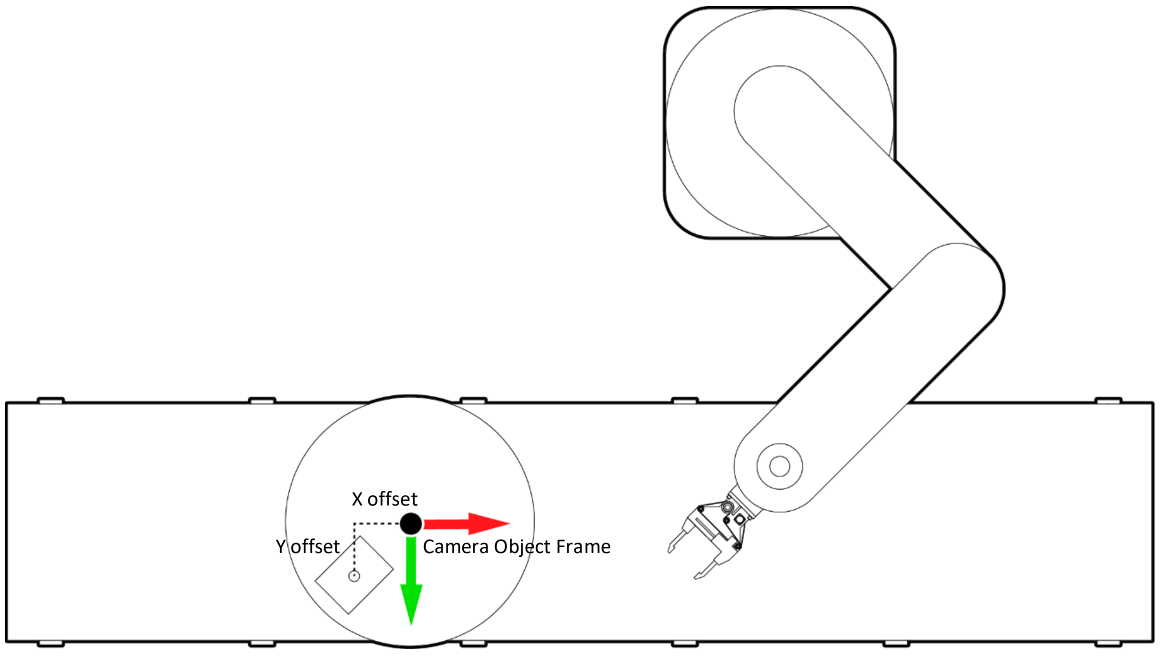


Example:

Products are placed in a conveyor belt randomly and have to be picked and placed to a specific position. A camera vision system will process the position and orientation of the products, sending this information to the controller. Depending on the camera technology used, the information could be sent through an Ethernet socket or using Trio ActiveX. The offsets measured for the camera are related to the camera origin of coordinate so we need to tell the system where that origin is by the use of OBJECT_FRAME.

Bear in mind that the robot will be synced with X direction, so the X vector of the OBJECT_FRAME should be aligned with the direction of the conveyor. After set the OBJECT_FRAME in the camera origin, a GTA should be set with the offset detected. In this example, the camera will send 5 parameters to a table data:

Table 0: X offset, Table 1: Y offset, Table 2: Orientation, Table 3: Conveyor position, Table 4: Detected



Code:

```
fixed_z = 50 ' Z will always be the same due to conveyor is parallel to the
ground

' Open gripper
OP(gripper,OFF)

SYNC_TO_GTA(4,500) ' Desync in 500ms
WAIT UNTIL SYNC_CONTROL = 0 ' Wait until fully desync

WAIT UNTIL TABLE(4) = 1 ' Wait until product is detected
TABLE(4) = 0 ' Reset product detected

' Set GTA value with camera measured offsets
GTA(0) = TABLE(0), TABLE(1),fixed_z, 180,0, TABLE(2)

' Register conveyor position when camera detects the product
registed_position = TABLE(3)

' Sync to axis 10, GTA(0) in Object Frame "camera" (1) in 2000ms
SYNC_TO_GTA(1,2000,registed_position,10,1,0)
WAIT UNTIL SYNC_CONTROL = 3 ' Wait until fully synced

' Close gripper
OP(gripper,ON)
WA(200) ' Wait for gripper close

SYNC_TO_GTA(4,500) ' Desync in 500ms
WAIT UNTIL SYNC_CONTROL = 0 ' Wait until fully desync
```

STOP