Machine and Robot in Harmony

RPS: Language Extension

Trio Motion Technology

Trio Robotics Series

First Edition • 2018

Revision: 1.2

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. 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 extensions is a dedicated language for robotics base on Trio basic, which improves the efficiency by reducing the number of lines in your programs. It match 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 has to 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 are 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 robot 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. Behavior

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 program types

There are two different robot program types: robot programs and basic robot programs.

Robot programs can contain just the instructions that TPS handled while Robot Basic Programs can contain all the existing Trio instructions.

Also, Robot Basic Program can only be edited by Motion Perfect.

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 specify 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.

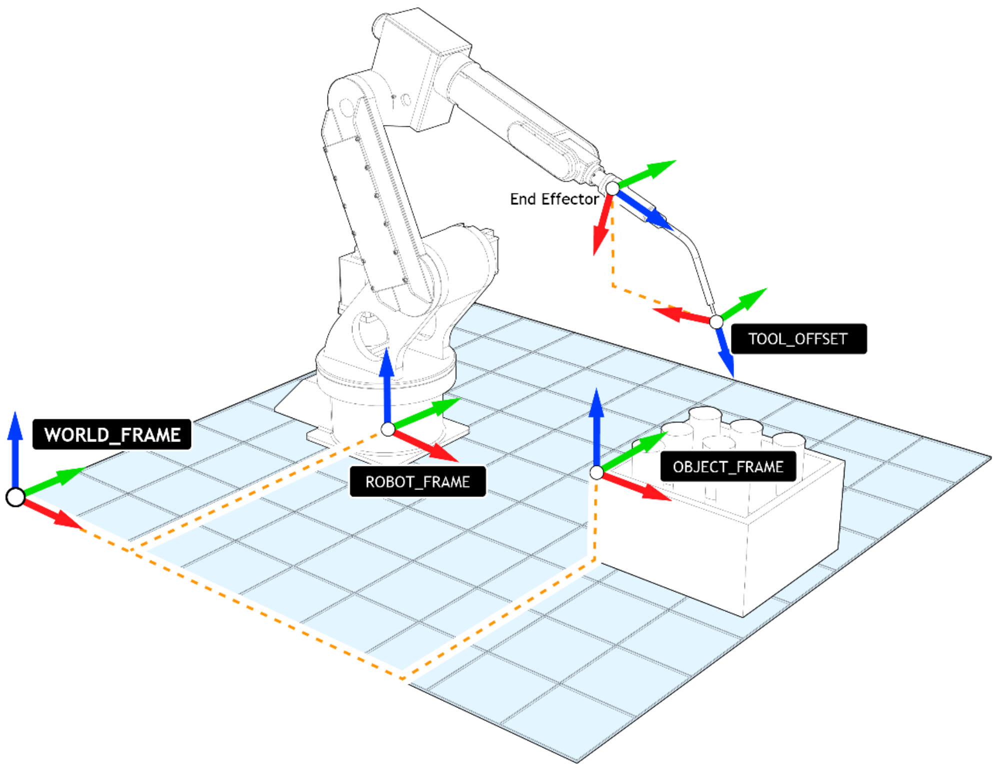


Figure 4-1: Coordinate system scenario.

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 specify in mm.

X rotation, y rotation and z rotation have to be specify 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.

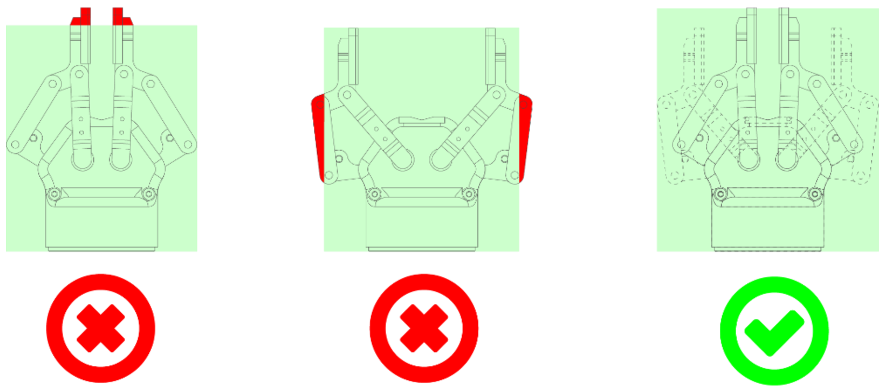


Figure 4-2: Tool collision dimensions.

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 modify or deleted, throwing the runtime error “TOOL cannot be modify 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.

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. Syntax

- Global target:

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

Example:

```
GTA(10) = 100,0,50,180,0,180,"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
```

5.3. 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 declare 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)
```

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 parameters	Embedded parameters
`AXIS_SPEED: joint speed deg/s`	S
`WORLD_SPEED: linear speed mm/s`	S
`TOOL_OFFSET`	T
`OBJECT_FRAME`	O
`ROBOT CONFIGURATION`	C
`PRECISION_ZONE`	ZF: orientation changes from start until half of blend curve. ZT: orientation changes along blend curve.

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.

6.2. Precision zone

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

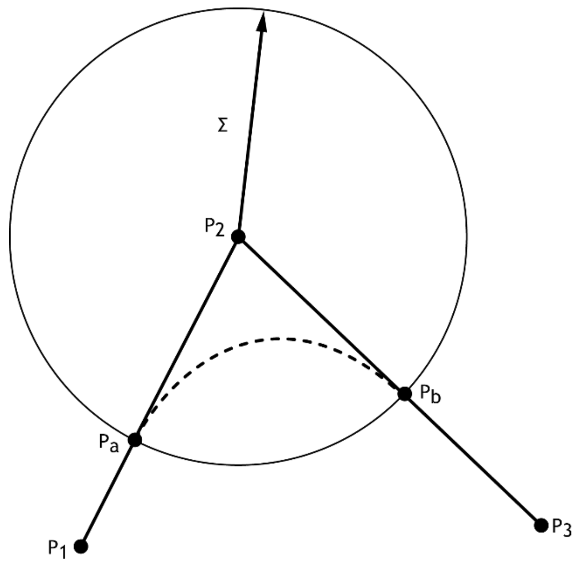


Figure 6-1: precision zone curve

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 half 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 quickest type of movement due to the axes move the exact amount of degrees needed to reach the desired position.

Syntax

MOVEJ target [S:= T:= O:= C:=]

6.4. MOVEL

Definition

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

Syntax

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

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:=]

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.

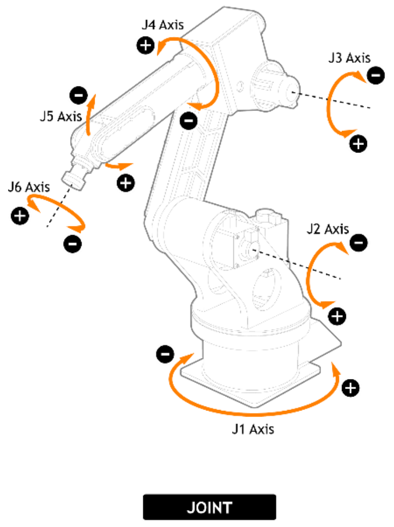
After setting all the inputs, jog mode has to be set using JOG_OPERATION command. Change the jog mode to activate the process and use FWD_IN and REV_IN inputs to jog the robot. Set jog mode back to 0 to release jog process for the execution of move instructions.

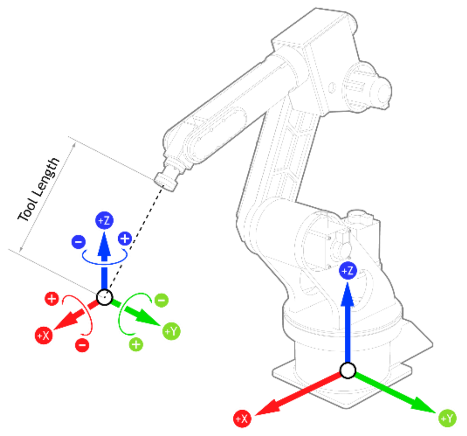
Jog cannot be performed until axes are idle.

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.

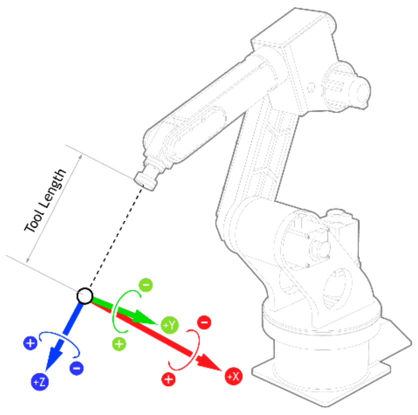




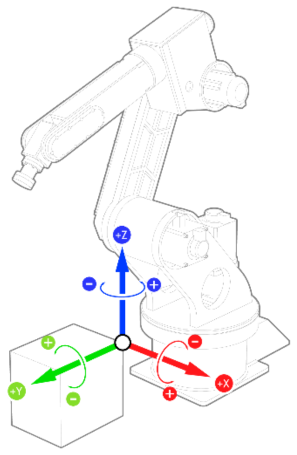
BASE / WORLD

- 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.

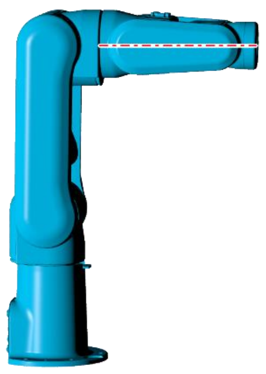


Figure 8-1: Wrist singularity.

- Elbow singularity occurs when the arm is fully extended.

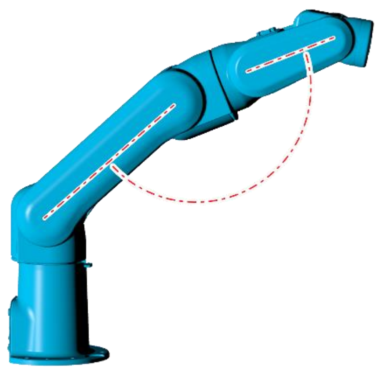


Figure 8-2: Elbow singularity.

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

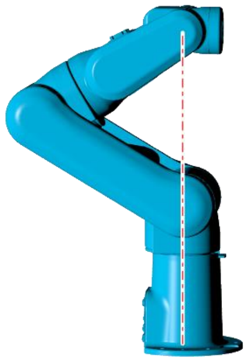


Figure 8-3: Alignment singularity.

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 configuration. 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.

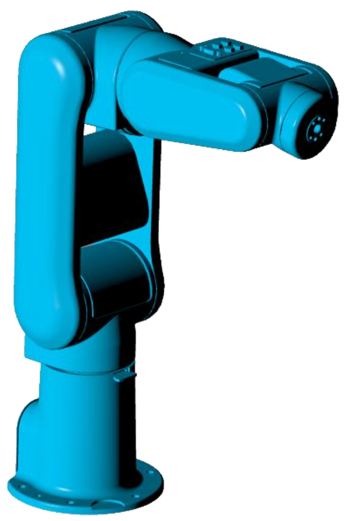


Figure 9-1: C := 0, Wrist no flip, elbow above, base forward.

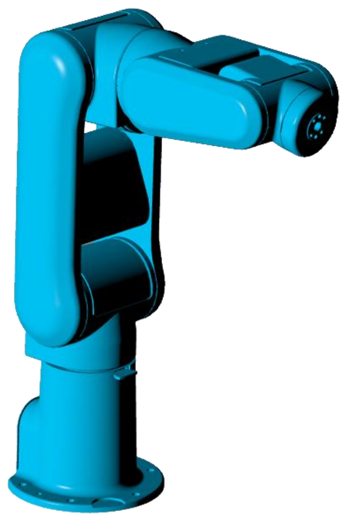


Figure 9-2: C := 1, Wrist flip, elbow above, base forward.

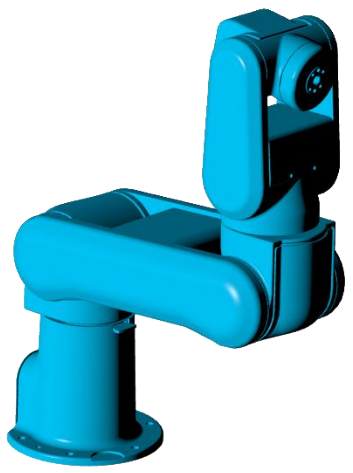


Figure 9-3: C := 2, Wrist no flip, elbow below, base forward.

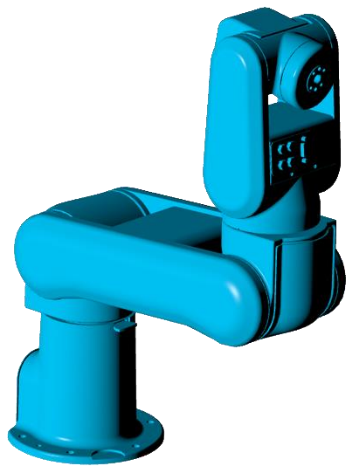


Figure 9-4: C := 3, Wrist flip, elbow below, base forward.

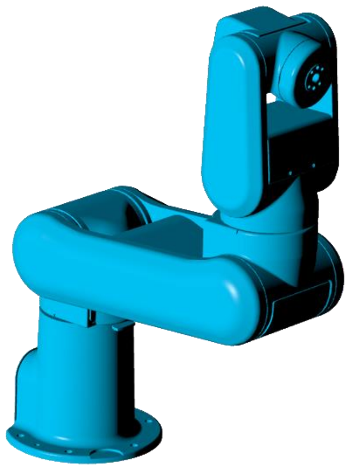


Figure 9-5: C := 4, Wrist no flip, elbow below, base backward.

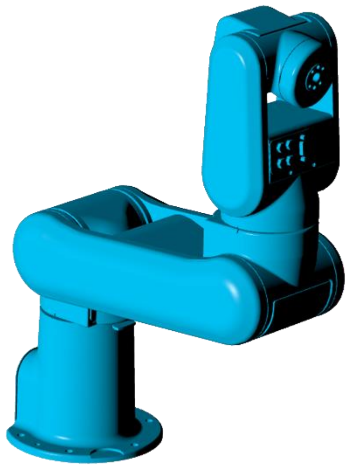


Figure 9-6: C := 5, Wrist flip, elbow below, base backward.

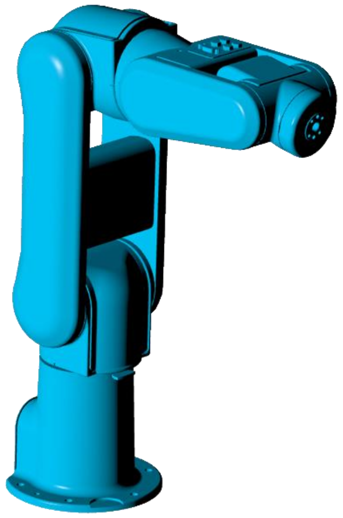


Figure 9-7: C := 6, Wrist no flip, elbow above, base backward.

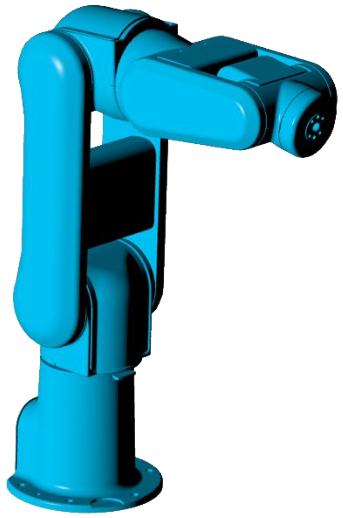


Figure 9-8:C := 7, Wrist flip, elbow above, base forward.

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.

Limits relative to world coordinate system. Object frame and robot frame applies here:

WORLD_FS_LIMIT: WORLD_DPOS is greater than WORLD_FS_LIMIT. This refers to Cartesian limit.

WORLD_RS_LIMIT: WORLD_DPOS is greater than WORLD_RS_LIMIT. This refers to Cartesian limit.

Limits relative to the TCP:

TCP_FS_LIMIT: TCP_DPOS is greater than TCP_FS_LIMIT. This refers to Cartesian limit on the TCP.

TCP_RS_LIMIT: TCP_DPOS is greater than TCP_FS_LIMIT. This refers to Cartesian limit on the TCP.

Limits relative to the end effector:

ROBOT_FS_LIMIT: ROBOT_DPOS is greater than ROBOT_FS_LIMIT. This refers to Cartesian limit on the end effector.

ROBOT_RS_LIMIT: ROBOT_DPOS is greater than ROBOT_FS_LIMIT. This refers to Cartesian limit on the end effector.

11. Errors

11.1. 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

11.2. 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

11.3. 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”)

12. Collision detection

12.1. 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.

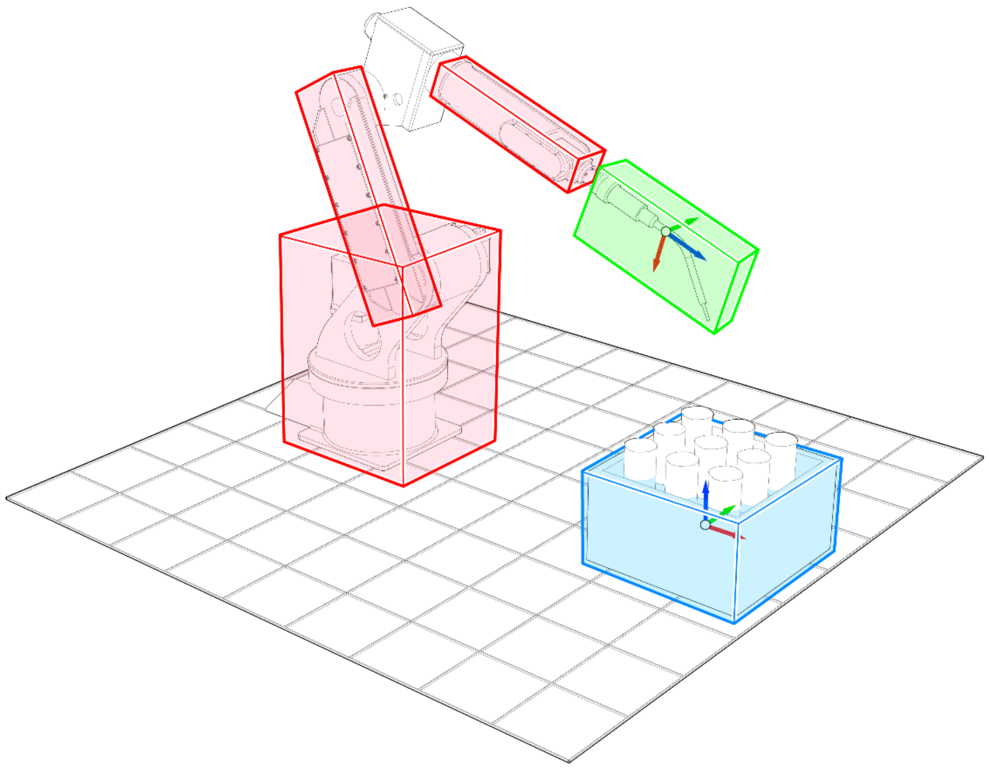


Figure 12-1: Collision detection scenario.

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

The collision algorithm measure 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.

12.2. 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 specify in mm.

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

X half distance, y half distance and z half distance have to be specify 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 depict 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.

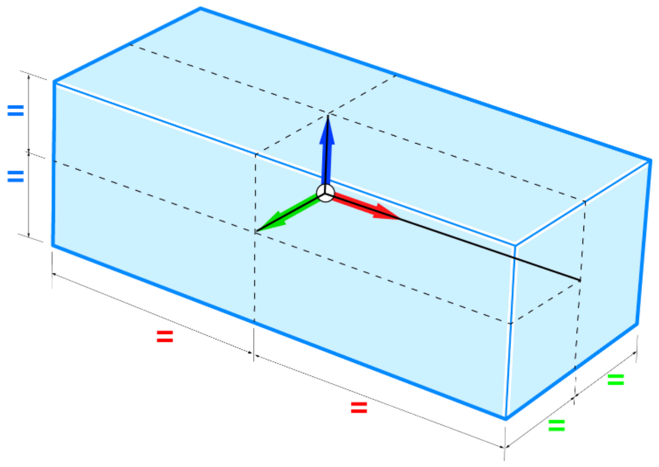


Figure 12-2: Collision object definition

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
```

13. 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.

13.1. 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 design 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

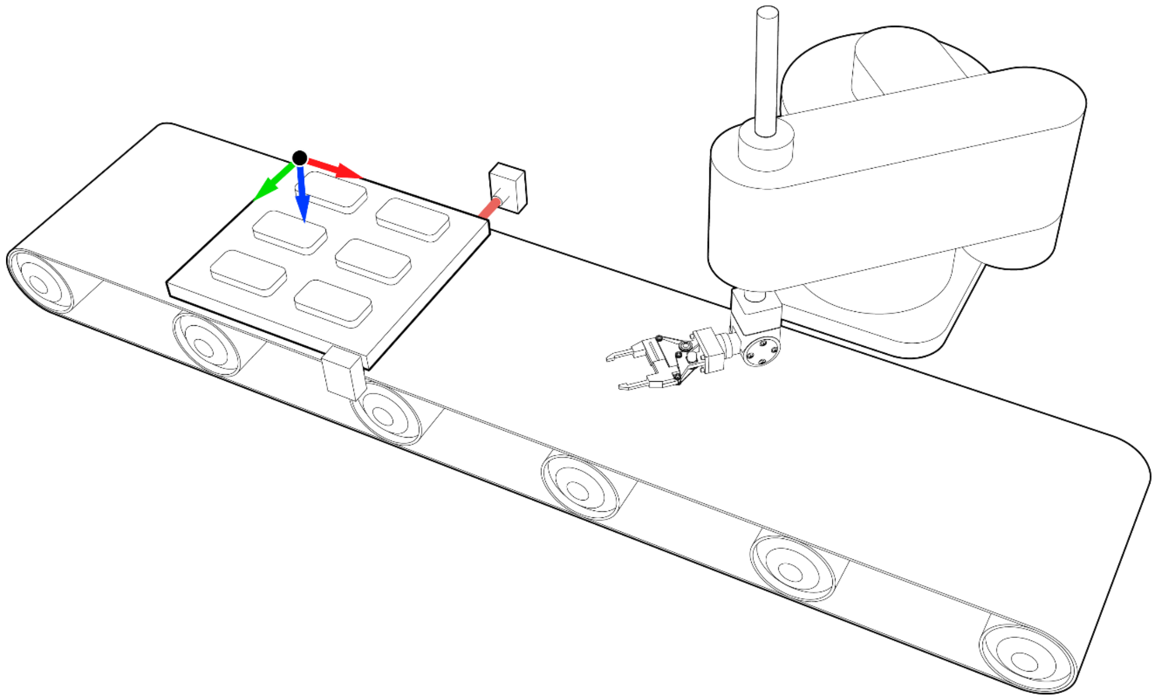
ATTACH_OBJECT_FRAME(control, object frame, syncVector, syncPos, syncAxis)

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.
`xOffset - wOffset`	Object Frame position and orientation related to the origin.

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 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:

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

13.2. 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

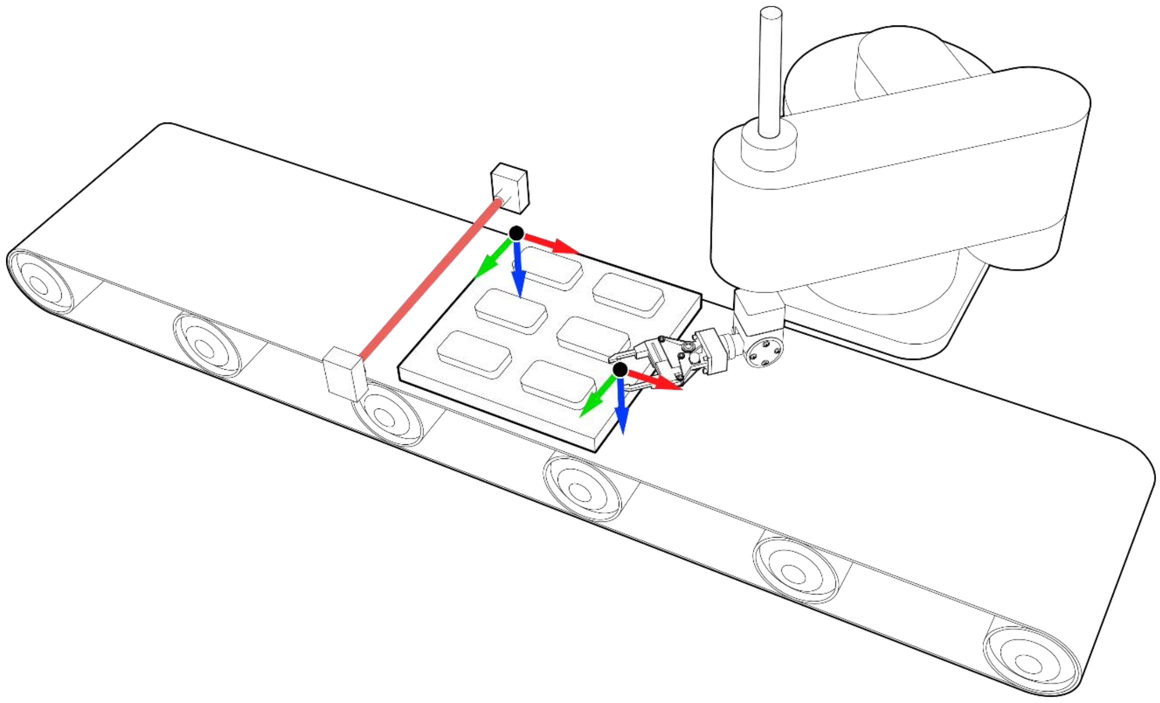
SYNC_TO_OBJECT_FRAME(control, time, object frame, GTA)

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.

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")
```



13.3. Sync to GTA

Sync to GTA has been designed for more simple synchronizations. It synchronize the robot TPC 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.

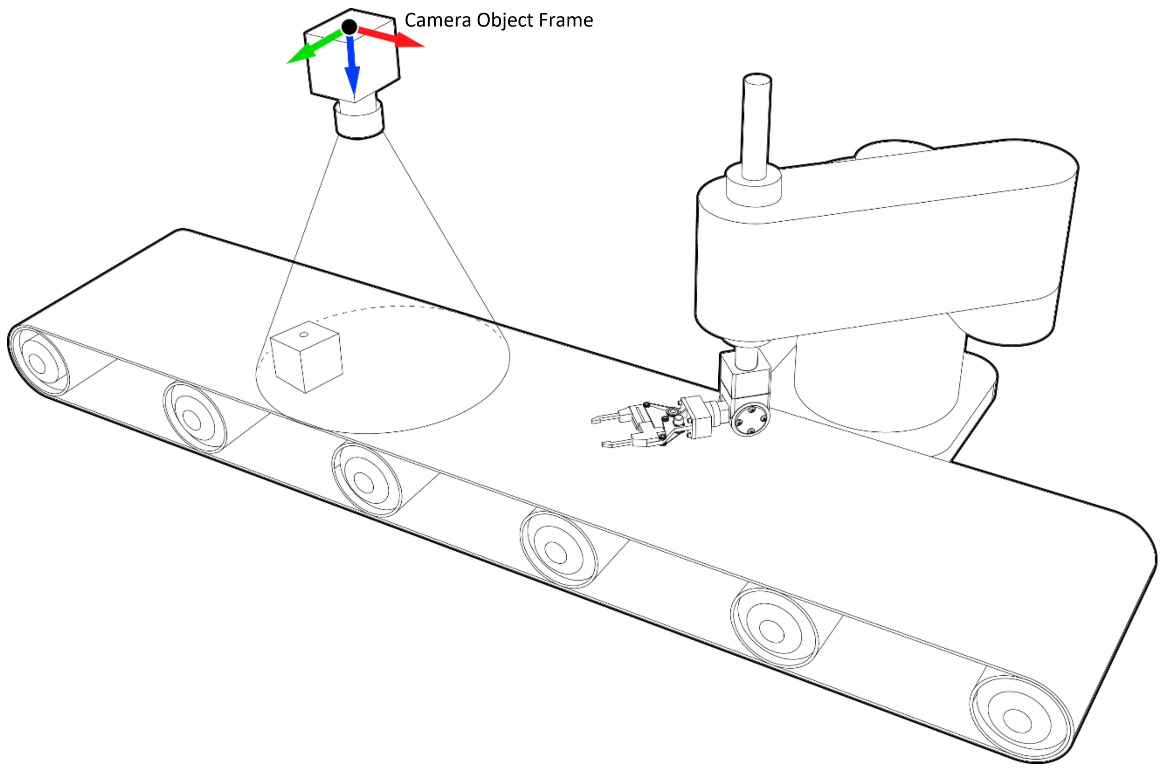
Syntax

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.

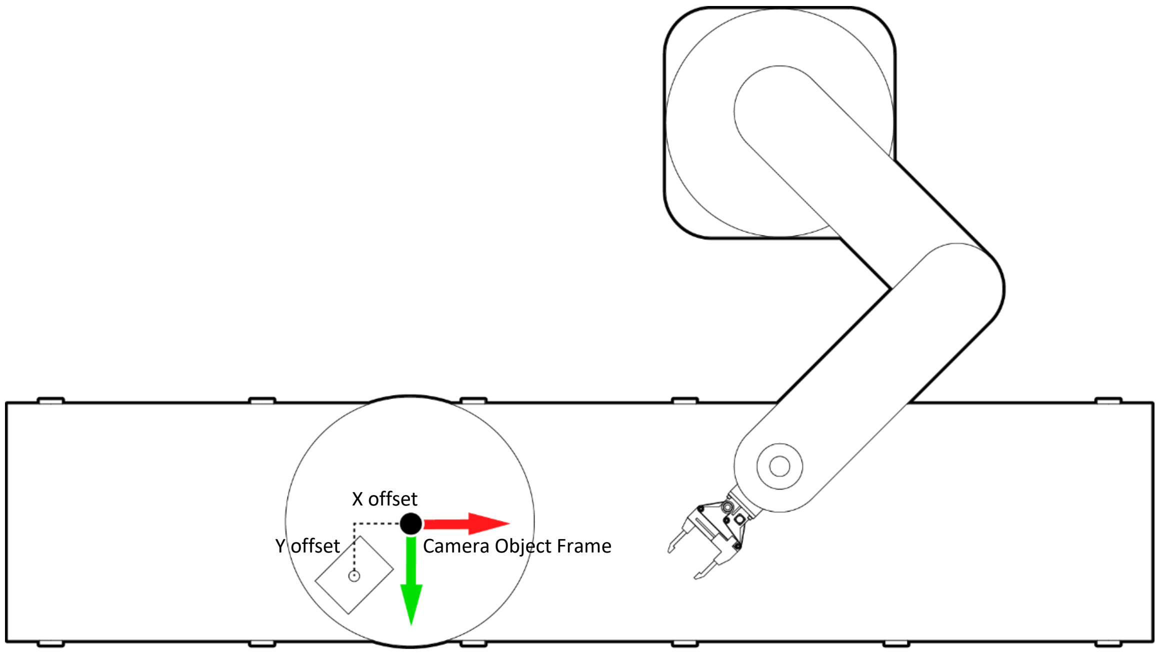
Example:

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 detect 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