SYNTAX:
value = FRAC(expression)

DESCRIPTION:
Returns the fractional part of the expression.

PARAMETERS:

value:	The fractional part of the expression
expression:	Any valid TrioBASIC expression

EXAMPLE:
Print the fractional part of 1.234 on the command line

```
>>PRINT FRAC(1.234)
0.2340
>>
```

FRAME

TYPE:
Axis Parameter

DESCRIPTION:
A **FRAME** is a transformation which enables the user to program in one coordinate system when the machine or robot does not have a direct or one-to-one mechanical connection to this coordinate system.

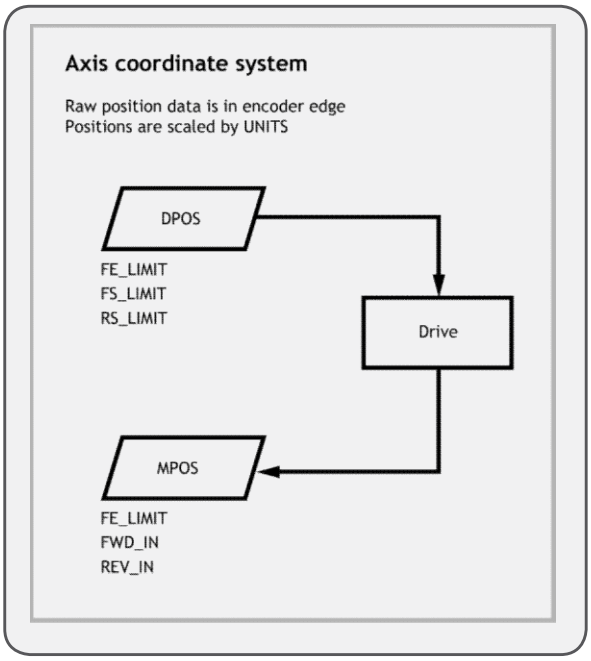
The **FRAME** command selects which transformation to use on axes in a **FRAME_GROUP**. Applying a **FRAME** to an axis in a **FRAME_GROUP** will apply that frame to all the axes in the group. To make this compatible with older firmware, if no FRAME_GROUPs have been configured then a default group is generated using the lowest axes, regardless of what axis the **FRAME** parameter was issued on.

Most transformations require configuration data to specify the lengths of mechanical links or operating modes. This is stored in the table with offsets detailed below in the parameters list. These table positions are offset by the 'table_offset' parameter in **FRAME_GROUP**. For a default **FRAME_GROUP** table_offset is 0.

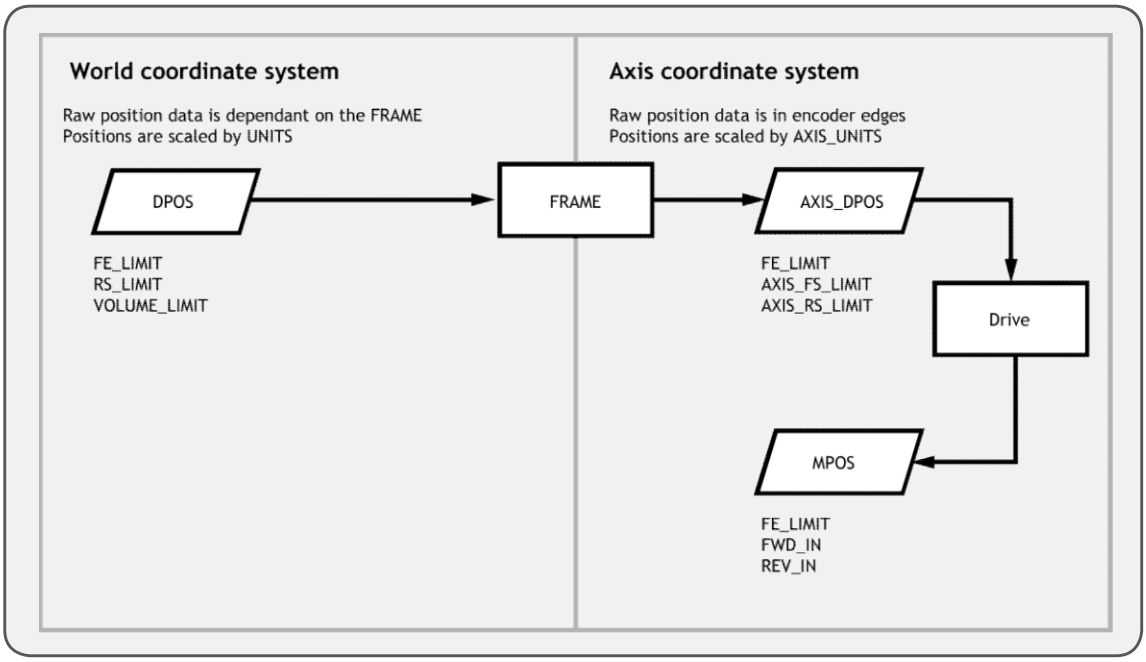

Do not change the **FRAME TABLE** parameters with the **FRAME** enabled. This can result in unpredictable movement which could cause damage or harm.


The kinematic runtime feature enable code is required to run **FRAME** 14 and higher

SYSTEM WITH FRAME=0



SYSTEM WITH FRAME<>0



AXIS SCALING

When a **FRAME** is enabled **UNITS** applies the scaling to the world coordinate system and **AXIS_UNITS** applies scaling to the axis coordinate system.

When **FRAME** is enabled **MPOS** is scaled by **AXIS_UNITS**, when frame is disabled **MPOS** is scaled by **UNITS**.

POSITION AND FOLLOWING ERRORS

When a **FRAME** is active **MPOS** is the motor position and **DPOS** is in the world coordinate system. **AXIS_DPOS** can be read to find the demand position in the motor coordinate system.

The following error is calculated between **MPOS** and **AXIS_DPOS** and so is the following error of the motor.

When using multiple frames or if you wish to group your axis you can use **DISABLE_GROUP** so that a **MOTION_ERROR** on one axis does not affect all.

HARDWARE AND SOFTWARE LIMITS

As **FS_LIMIT** and **RS_LIMIT** use **DPOS** they are both active in the world coordinate system. **VOLUME_LIMIT** also uses **DPOS** so is also in the world coordinate system. **FWD_IN** and **REV_IN**, **AXIS_FS_LIMIT** and **AXIS_RS_LIMIT** use **AXIS_DPOS** as so act on the forward and reverse limit of the motor.

When moving off **FWD_IN** and **AXIS_FS_LIMIT** the motor must move in a reverse direction. Due to the **FRAME** transformation this may not be a reverse movement in the world coordinate system. When moving off a **REV_IN** and **AXIS_RS_LIMIT** the motor must move in a forward direction. Due to the **FRAME** transformation this may not be a forward movement in the world coordinate system.

OFFSETTING POSITIONS

When a **FRAME** is enabled **OFFPOS** and **DEFPOS** must not be used as they cause a jump in both **DPOS** and **MPOS**. As the transformation separates **DPOS** and **MPOS** using these commands will cause an undesirable jump in motor position.

REP_DIST also causes a jump in **DPOS** and **MPOS** so when using a **FRAME** the position must never reach **REP_DIST**. **REP_OPTION** must be set to 0 and **REP_DIST** must be at least twice the size of the biggest possible move on the system.

When **DATUM** is complete it also causes a jump in **DPOS** and **MPOS**, so **DATUM** must never be used when **FRAME** <> 0

You can use **USER_FRAME** to define a different origin to program from.

POWER ON SEQUENCE AND HOMING

Some **FRAME** transformations require the machine to be homed and/ or moved to a position before the **FRAME** is enabled. This can be done using the **DATUM** function. If you home position is not the zero position of the **FRAME** then you can use **DEFPOS**/ **OFFPOS** to set the correct offset before enabling the **FRAME**.

When a **FRAME** is enabled **DPOS** is adjusted to the world coordinates which are calculated from the current **AXIS_DPOS**.

You should not perform a **DATUM** homing routine when the **FRAME** is enabled as this will change the **DPOS** which may result in undesirable motion. If you need to perform homing when the **FRAME** is enabled you can move to a registration position and then use **USER_FRAME** to apply the offset.

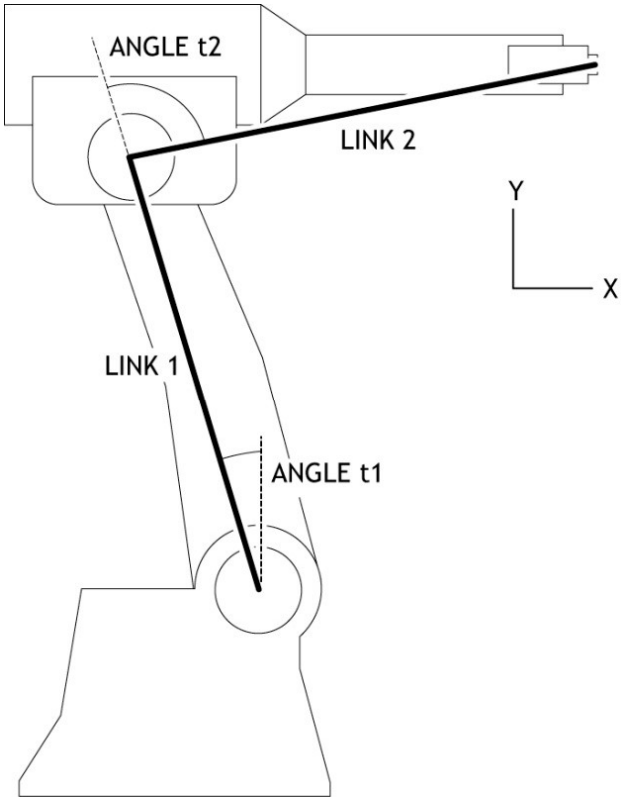
VALUE:

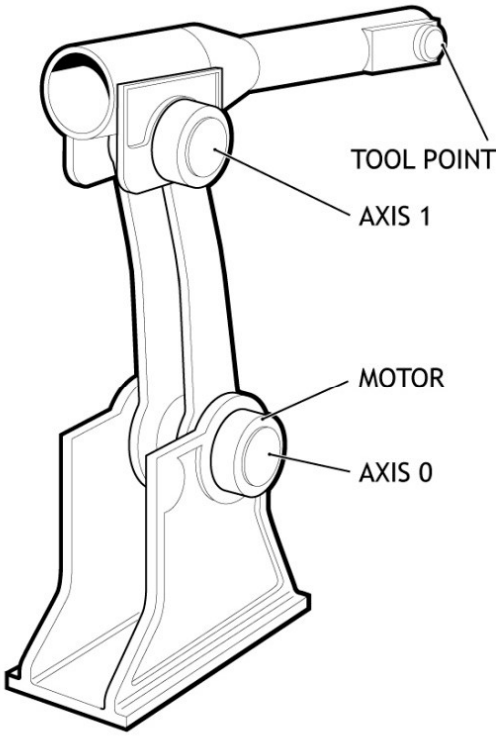
0	No transform
1	2 axis **SCARA** robot
2	XY single belt
5	2 axes rotation
6	Polar to Cartesian transformation
10	Cartesian to polar transformation
13	Dual arm robot transformation
14	3 arm delta robot.
15	4 axis **SCARA**
16	3 Axis Robot with 2 Axis Wrist
17	Wire guided camera
18	6 axis articulated arm
114	3 arm delta robot.
115	3 to 5 axis **SCARA**
116	3 Axis Robot with 2 Axis Wrist
119	3 to 5 axis cylindrical robot with 2 Axis Wrist

FRAME=1, 2 AXIS SCARA

DESCRIPTION:

Frame=1 allows the user to program in X, Y, Cartesian coordinates for a 2 axis **SCARA** arm like the example below. The frame allows for 2 configurations of a **SCARA** depending if the second axis motor is in the joint or at the base. The difference is that in angle t2 is referenced from link 1, or t2 is referenced from the base. A linkage or belt is typically used to keep t2 referenced to the base.





Second motor is carried on the end of Link 1, t2 is relative to link 1

Second motor in base with link arm to move upper part, t2 is relative to the base

Once the frame is enabled **DPOS** is measured in Micrometres, **UNITS** can then be set to a convenient scale.

HOMING

Is it required that the 2 motors' absolute positions are homed relative to the "straight up" position before the **FRAME** is enabled. In other words, the zero angle on each axis is with the arms in line and vertical. Of course it is not necessary for the motors to actually go to this position as you can offset the position using **DEFPOS** or **OFFPOS**.

JOINT CONFIGURATION

The joint configuration is determined by the position of the **SCARA** arm when you enable **FRAME** = 1

The joint is defined as Right Handed if:

(t2<t1) –both motors in base

(t2<0) –motors in the joint

Otherwise the robot is Left handed

PARAMETERS:

Table data	0	Length of arm 1 in micrometres
	1	Length of arm 2 in micrometres
	2	Edges per radian for joint 1
	3	Edges per radian for joint 2
	4	Internal value. Set to 0 to force frame re-calculation
	5	Axis configuration:
		0 – Both motors fixed in base
		1 – Motors at the joint
	6	Joint configuration (read only):
		0 – Left handed **SCARA**
		1 – Right handed **SCARA**
	7	used internally
	8	used internally

EXAMPLES:

EXAMPLE 1:

Set up the **SCARA** arm which is configured with the motors in the joints. Both motors return 16000 counts per revolution. The robot can be homed to switches which are at -80 degrees and +150degrees for the two joints. After setting **FRAME**=1 the tip of the second arm will be set with X, Y as (0,42426). This effectively makes the (0,0) XY position to be the bottom joint of the lower arm.

All the normal move types can then be run within the **FRAME**=1 setting until it is reset by setting **FRAME**=0. As the **FRAME** 1 makes the resolution of axes 0 and 1 micrometres, the **UNITS** can be set so you can program in mm.

```
FRAME=0

'Enter Configuration Parameters:
TABLE(0, 300000) '    Length of arm 1 in mm * 1000
TABLE(1, 445000) '    Length of arm 2 in mm * 1000
TABLE(2, 16000/(2*PI)) ' edges per radian for joint 1
TABLE(3, 16000/(2*PI)) ' edges per radian for joint 2
TABLE(4, 0) ' Internal value. Set to 0 to force frame re-calculation
TABLE(5, 1) ' set to 1 for second joint fixed to arm 1

'Home the robot to its mechanical limit switches
DATUM(3) AXIS(0) ' find home switch for lower part of arm
WAIT IDLE
```

```
DATUM(3) AXIS(1) ' find upper arm home position
WAIT IDLE

'The mechanical layout may make it impossible to home at (0,0)
'Define the home position values as their true angle (in edges)
DEFPOS(-3555,6667) ' say home position is -80 deg and +150 deg
WAIT UNTIL OFFPOS=0

'Move both arms to start position PI/4 radians (45 degrees)
MOVEABS(-TABLE(2)*0.7854,TABLE(3)*0.7854*2)
WAIT IDLE

FRAME=1

UNITS AXIS(0)=1000
UNITS AXIS(1)=1000
```

EXAMPLE 2:

Set up the table for **SCARA** arm which is configured with both motors in the base. Once the table is configured the rest of the initialisation is the same as the above example.

```
' Enter Configuration Parameters:
TABLE(0,400000) '         Link 1 in mm * 1000
TABLE(1,250000) '         Link 2 in mm * 1000
TABLE(2, 4096*5/(2*PI)) ' t1 in edges per radian
TABLE(3, 4096*3/(2*PI)) ' t2 in edges per radian
TABLE(4,0) ' Internal value. Set to 0 to force frame re-calculation
TABLE(5,0) ' set to 0 for second joint fixed to base
```

FRAME=2, XY SINGLE BELT

DESCRIPTION:

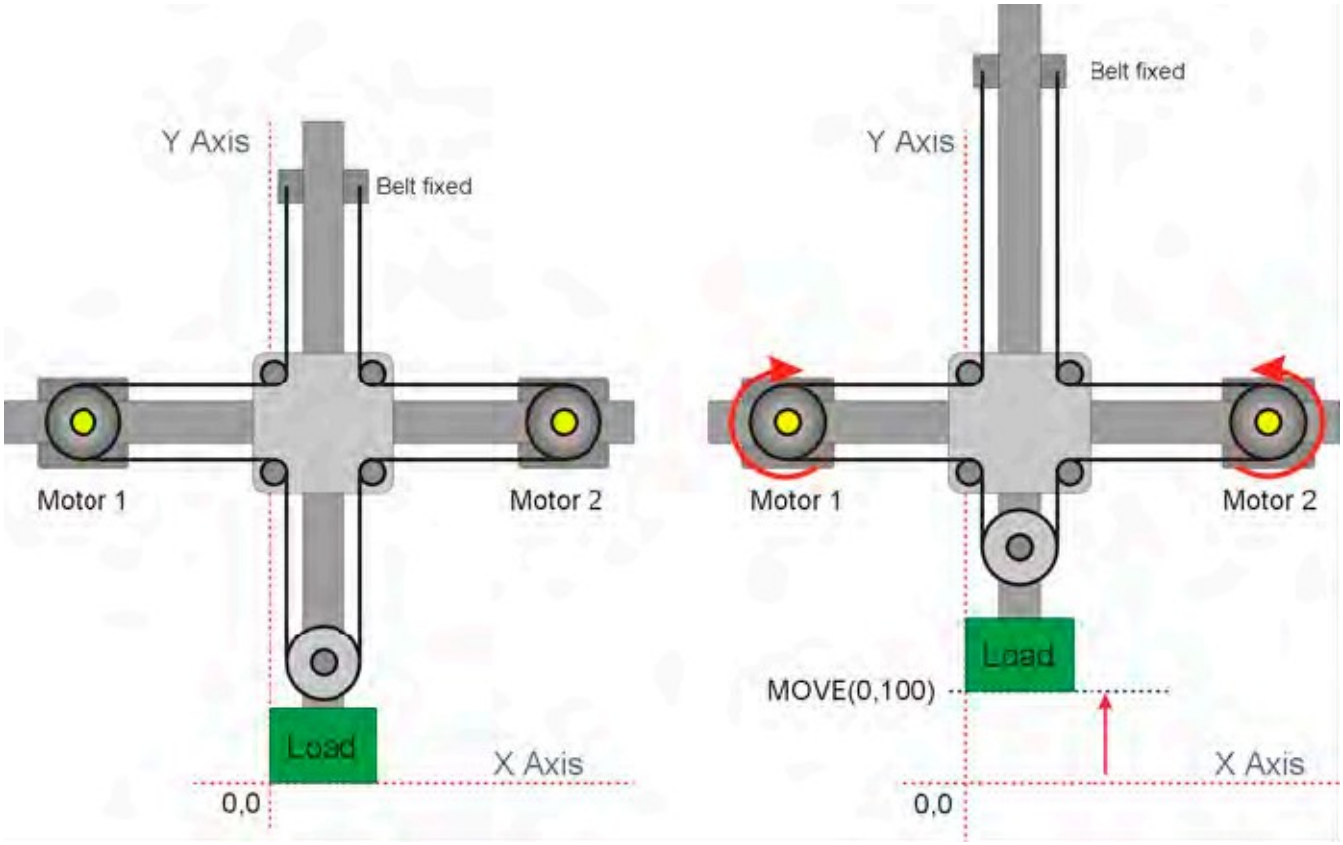
Switching to **FRAME**=2 will allow X-Y motion using a single-belt configuration. In this mode, an interpolated move of **MOVE**(0,100) produces motion on both motor 1 and motor 2 to raise the load vertically, based on the transformed position. Note that the two motors are located on the X-axis. The mass of the Y-axis can be minimized in this configuration. The equations for the transformed position of the X and Y axes are as follows:

Xtransformed = (**MPOS AXIS**(0)+ **MPOS AXIS**(1))*0.5

Ytransformed = (**MPOS AXIS**(0)- **MPOS AXIS**(1))*0.5

The transformed X-Y coordinates are derived from the measured encoder position (**MPOS**) of **AXIS**(0) and **AXIS**(1). This conversion is automatically accomplished by the *Motion Coordinator* when **FRAME**=2.

Once the frame is enabled **DPOS** is measured in encoder counts, **UNITS** can be set to enable a more convenient scale.



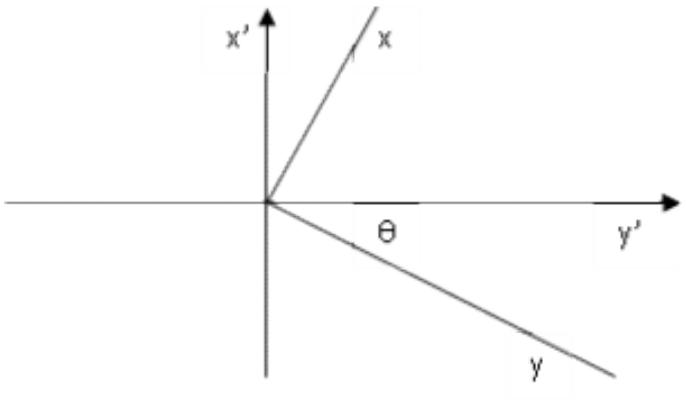
EXAMPLE:

```
ATYPE=0 'disable built in axes for MC464

FRAME=0

'Define a start position
DEFPOS(150,50)
FRAME=2
```

FRAME=5, 2 AXES ROTATION

DESCRIPTION:

This frame is designed to allow two orthogonal axes to be “turned” through an angle so that command inputs to x, y (along the required plane) are transformed to the fixed axes x’ and y’.



The transform is done by way of a 2 x 2 matrix, the coefficients of which can be easily derived from the required rotation angle of the operating plane.

CALCULATING THE MATRIX COEFFICIENTS:

For the frame to work, 2 sets of matrix coefficients must be entered, one for the forward transform and the second for the inverse. The transform calculates x and y according to the following:

```
(x', y') = (x, y) * ⎛TABLE(0), TABLE(1)ö
                    ⎝TABLE(2), TABLE(3)⌀
```

The inverse transform is calculated thus:

```
(x, y) = (x', y') * ⎛TABLE(4), TABLE(5)ö
                    ⎝TABLE(6), TABLE(7)⌀
```

HOMING:

The axes should be datumed in **FRAME**=0. Once this is done, then the frame can be set to 5 and move commands directed at either axis or at both axes together in the usual way. However the actual movement of x' and y' (the real axes) will be according to the transform.

If the axes need to be re-positioned according to the real axes, the frame can be turned off simply by setting **FRAME**=0. When this is done, the **DPOS** values will change to be the same as the **MPOS** positions, i.e. they become the positions in the x' / y' plane. The axes can then be moved to a new starting position and the frame set back to 5, perhaps with a new angle set.

PARAMETERS:

Table data	0	COS(theta)
	1	-SIN(theta)
	2	SIN(theta)
	3	COS(theta)
	4	**TABLE**(3) / det
	5	-**TABLE**(1) / det
	6	-**TABLE**(2) / det
	7	Table(0) / det

theta, the angle of rotation is in radians.

det = (**TABLE**(0) * **TABLE**(3)) - (**TABLE**(2) * **TABLE**(1))

EXAMPLE:

Configure a rotation of 45 degrees and run a move on the new X Y axes.

```
x_axis = 0
y_axis = 1

theta_degrees = 45 'Rotation angle in degrees
theta = theta_degrees * (2*PI/360) 'Convert to radians
GOSUB calc_matrix
FRAME = 5

BASE(x_axis)
MOVE(xdist, ydist)
WAIT IDLE

STOP

'=================================================
' Calculate the matrix parameters for FRAME 5
' Transform (x, y) * (TABLE(0), TABLE(1) )
'                    (TABLE(2), TABLE(3) )
'
' Inverse Transform:
'          (x', y') * (TABLE(4), TABLE(5) )
'                     (TABLE(6), TABLE(7) )
```

```
'================================================
calc_matrix:
'Forward transform
TABLE(0, COS(theta))
TABLE(1, -SIN(theta))
TABLE(2, SIN(theta))
TABLE(3, COS(theta))

'Inverse transform
det = (TABLE(0) * TABLE(3)) - (TABLE(2) * TABLE(1))
TABLE(4, TABLE(3) / det)
TABLE(5, -TABLE(1) / det)
TABLE(6, -TABLE(2) / det)
TABLE(7, TABLE(0) / det)
RETURN
```

FRAME=6, POLAR TO CARTESIAN TRANSFORMATION

DESCRIPTION:

This transformation allows the user to program in polar (radius, angle) coordinates and the actual axis to move in a Cartesian (X, Y) coordinate system.

The first axis in the frame group is the Radius, the second is the angle. .

Once the frame is enabled the raw position data (**UNITS**=1) is measured in encoder counts for the radius axis and radians*scale for the angle, **UNITS** can then be set to a convenient scale. The origin for the robot is the zero position for the Cartesian system. The zero angle position is along Axis 0.

PARAMETERS:

Table data	0	Scale (counts per radian) for the rotary axis

EXAMPLES:

EXAMPLE 1:

A gantry robot has 2 axis configured in an X, Y configuration. For ease of programming the user would like to program in Polar coordinates. Both axes return 4000 counts per revolution. The **AXIS_UNITS** are set so that the axis coordinate system is in mm, the **UNITS** are set so that the World coordinate system is in mm and degrees.

```
scale = 1000000
UNITS AXIS(0) = 4000 'To program in mm
AXIS_UNITS AXIS(0) = 4000
UNITS AXIS(1) = scale*2*PI/360 'to program in degrees
AXIS_UNITS AXIS(1) = 4000
TABLE(0, scale) 'Set resolution for the angle axis
FRAME = 6
```

EXAMPLE 2:

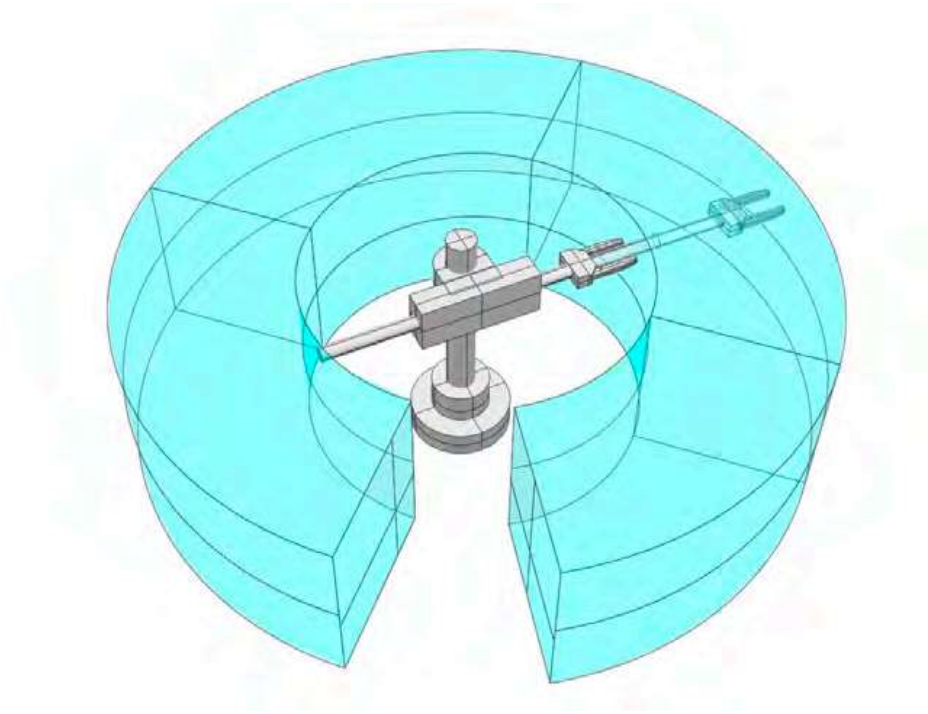
Using the robot configured in example 1 move the tool to 150mm along the X axis, then move the tool in a circle around the Polar coordinate system origin.

```
MOVEABS(150,0)
MOVE(0,360)
```

FRAME=10, CARTESIAN TO POLAR TRANSFORMATION

DESCRIPTION:

This **FRAME** transformation allows the user to program in Cartesian (X,Y) coordinates on a system that moves in a Polar (radius, angle) coordinate system. This is typically used on cylindrical robots where you need to program the arm extension (radius) and angle. The vertical Z axis can be simply added to make a 3 degree of freedom system.



Once the frame is enabled the raw position data (**UNITS**=1) is scaled the same for the X and Y axes, the resolution is set from the radius axis. **UNITS** can then be set to a convenient scale. The origin is the centre of the Polar system. .

The first axis in the group controls the radius axis and the second controls the rotary axis.

HOMING

Before enabling **FRAME**=10 the axes must be homed so that they are at a known position. When the **FRAME** is enabled the X and Y positions are calculated from the current Polar position.

Take care when executing moves that go close to the origin. Moves that travel through the origin will require infinite speed and acceleration. This is usually not possible to achieve and the axes will trip out due to excessive following error.

PARAMETERS:

Table data	0	Encoder edges/radian
	1	Number of revolutions, set by firmware
	2	Previous servo cycle's angle, set by firmware

EXAMPLE:

A cylindrical robot has 3 axis which extend the arm (radius), rotate the arm (angle) and move the up and down (Z). The radius and Z axes have 4000 counts per mm, this is used for the scale of the Cartesian axes in the **FRAME**. The rotate axis has 4000 counts per revolution, this should be divided by 2*PI to give the counts per revolution which is set in the table. The **UNITS** are set so that the Cartesian system can be programmed in mm, the **AXIS_UNITS** is set so that the axis are programmed in mm or degrees. Once the polar system has been homed the following code can be executed so that any further motion is programmed in Cartesian coordinates.

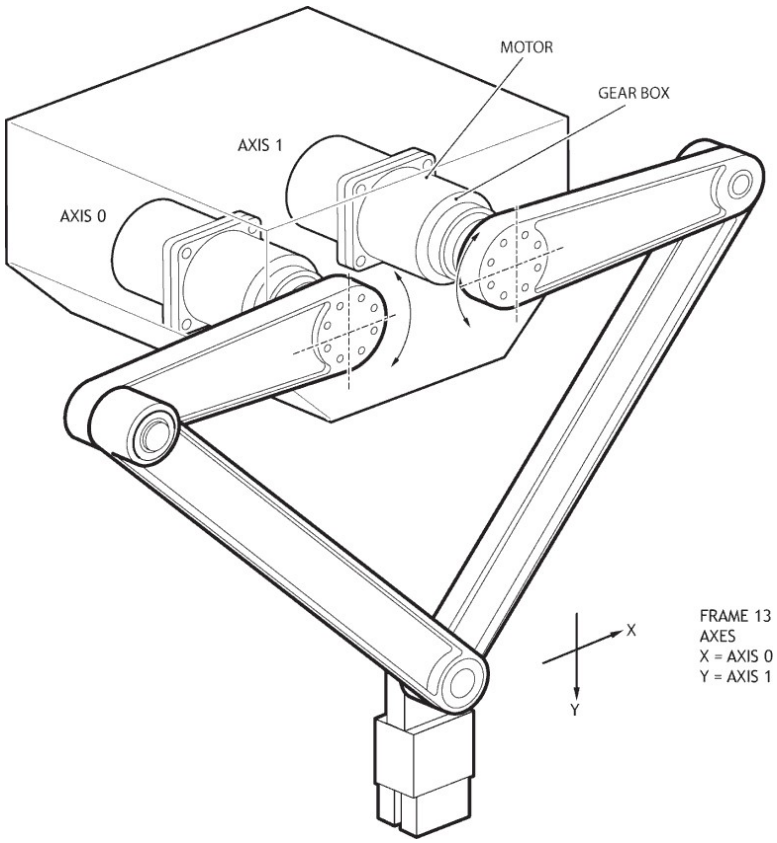
```
UNITS AXIS(0) = 4000 'To use in mm
AXIS_UNITS AXIS(0) = 4000 'To use in mm
edges_per_radian = 4000/(2*PI) 'Edges per radian for the rotary axis
UNITS AXIS(1) = 4000'To use in mm
AXIS_UNITS AXIS(1) = 4000 / 360 'To use in mm
TABLE(0,edges_per_radian)
UNITS AXIS(2) = 4000 'To use in mm
FRAME = 10
```

FRAME=13, DUAL ARM PARALLEL ROBOT

DESCRIPTION:

Frame 13 enables the transformation for a 2 arm parallel robot as shown. It is then possible to program in X Y Cartesian coordinates.

If the lower link is not directly connected as per the image but is separated, this is compensated for by decreasing the centre distance of the top link by the same amount.

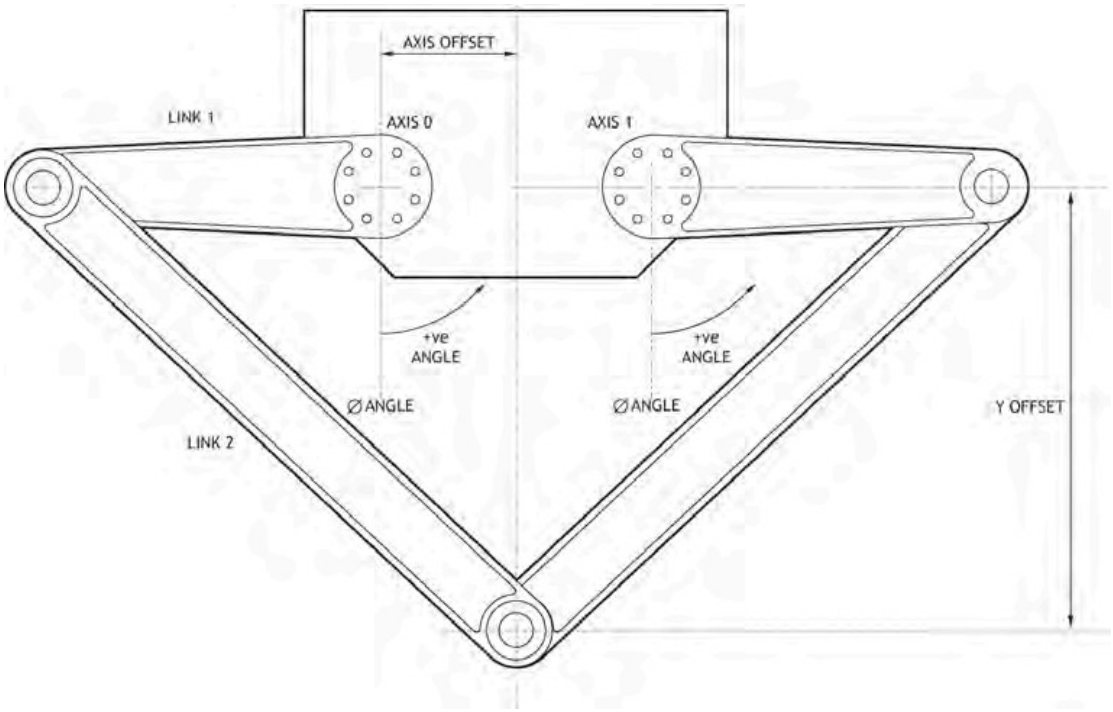


Once the frame is enabled the raw position data (**UNITS**=1) is measured in Micrometres, **UNITS** can then be set to a convenient scale.

HOMING

The 2 arm delta robot should be homed so that the two link 1's are vertical down. You do not need to enable the frame in this position, just ensure that it has been defined.

A vertical offset for the tool can be defined within the **FRAME** table data. This means that you can set the zero position vertically



PARAMETERS:

Table data	0	Link length 1 in microns
	1	Link length 2 in microns
	2	Encoder edges/radian axis 0
	3	Encoder edges/radian axis 1
	4	Horizontal offset axes from x datum
	5	Set Vertical datum with arms straight out
	6	calculated values
	7	calculated values
	8	calculated values
	12	first axis frame calculated value

EXAMPLE

The following is a typical startup program for **FRAME** 13.

```
FRAME=0
WA(10)
'-----------------------------------------------
TABLE(0,220000)'Arm
TABLE(1,600000)'Forearm
TABLE(2,(2048*4*70)/2/PI)'pulse/radian
TABLE(3,(2048*4*70)/2/PI)'pulse/radian
TABLE(4,15000)'X-offset
TABLE(5,450000)'Y-offset = 450 mm below axis 0 centre
'-----------------------------------------------

' set home position for arms at +/-90 degrees
DATUM(4) AXIS(0) 'find home switch for left arm
DATUM(3) AXIS(1) 'find home switch for right arm
WAIT IDLE AXIS(0)
WAIT IDLE AXIS(1)
home_0 = -TABLE(2)*PI/2
home_1 = TABLE(3)*PI/2
BASE(0,1)
DEFPOS(home_0,home_1)

WA(10)
FRAME=13
```

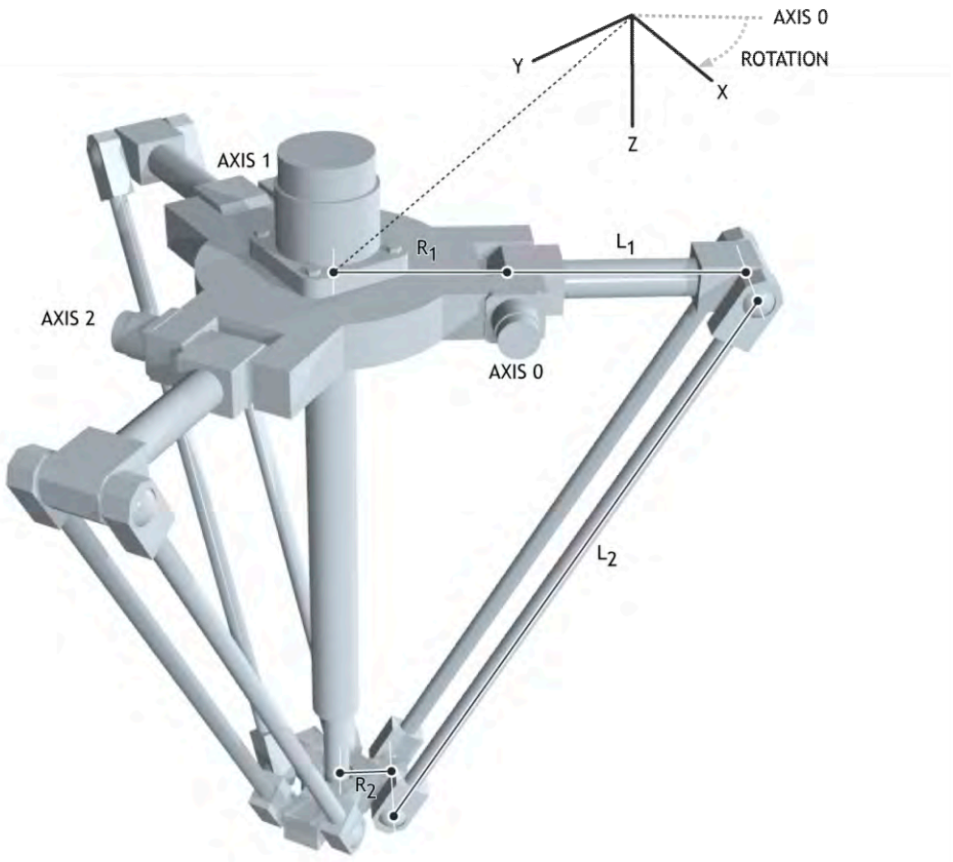
FRAME=14, DELTA ROBOT

DESCRIPTION:

FRAME=14 enables the transformation for a 3 arm 'delta' or 'parallel' robot. It transforms 3 axes from the mechanical configuration to Cartesian coordinates using the right hand rule.

For new projects **FRAME** 114 is recommended

FRAME=14 requires the kinematic runtime **FEC**



Once the frame is enabled the raw position data (**UNITS**=1) is measured in Micrometres, **UNITS** can then be set to a convenient scale. The origin for the robot is the centre of the top plate with the X direction following the first axis. This can be adjusted using the rotation parameter.

HOMING:

Before enabling **FRAME**=14 the position must be defined so that when the upper arms are horizontal the axis position is 0. You do not need to enable the frame in this position, just ensure that it has been defined.

PARAMETERS:

Table data	0	Top radius to joint in Micrometres (R1)
	1	Wrist radius to joint in Micrometres (R2)
	2	Upper arm length in Micrometres (L1)
	3	Lower arm length in Micrometres (L2)
	4	Edges per radian
	5	Angle of rotation in radians (Rotation)

EXAMPLE:

Start-up sequence for a 3 arm delta robot using the default **FRAME_GROUP**. Homing is completed using a sensor that detects when the upper arms are level.

```
' Define Link Lengths for 3 arm delta:
  TABLE(0,200000)' Top radius to joint
  TABLE(1,50000)'  Wrist radius to joint
  TABLE(2,320000)' Upper arm length
  TABLE(3,850000)' Lower arm length

' Define encoder edges/radian
  '18bit encoder and 31:1 ratio gearbox
  resolution = 262144 * 31 / (2 * PI)
  TABLE(4,resolution)

' Define rotation of robot relative to global frame
  rotation = 30 'degrees
  TABLE(5, (rotation*2*PI )/360)

' Configure axis
  FOR axis_number=0 TO 2
    BASE(axis_number)
    'World coordinate system to operate in mm
    UNITS=1000
    SERVO=ON
  NEXT axis_number

  WDOG=ON
  BASE(0)

' Home and initialise frame
  'Arms MUST be horizontal in home position
  ' before frame is initialised.
```

```
FOR axis_number=0 TO 2
  DATUM(4)
  WAIT IDLE
NEXT axis_number

'Enable Frame
FRAME=14
```

FRAME=15, 4 AXIS SCARA

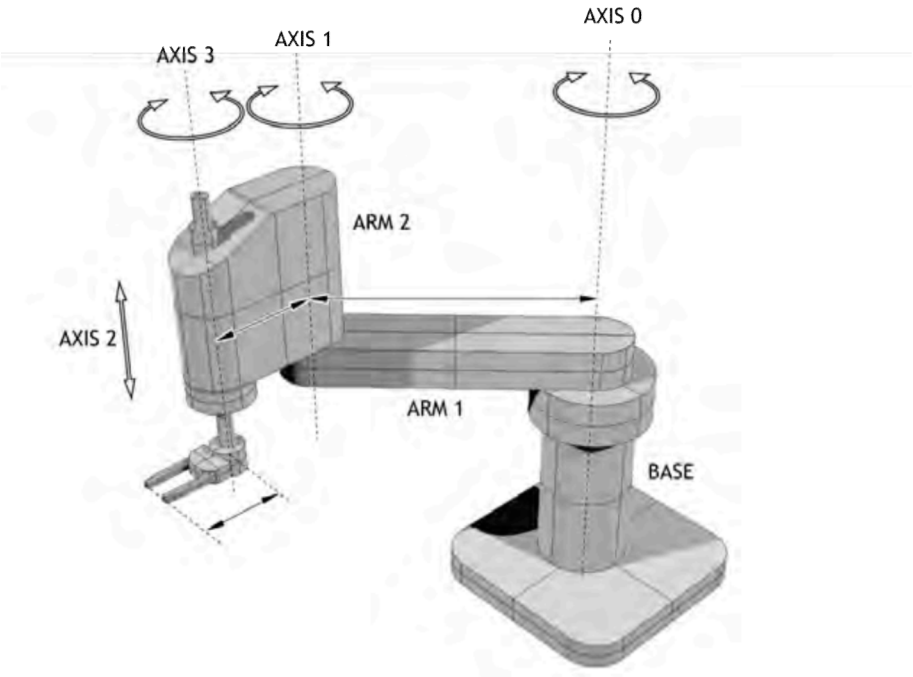
DESCRIPTION:

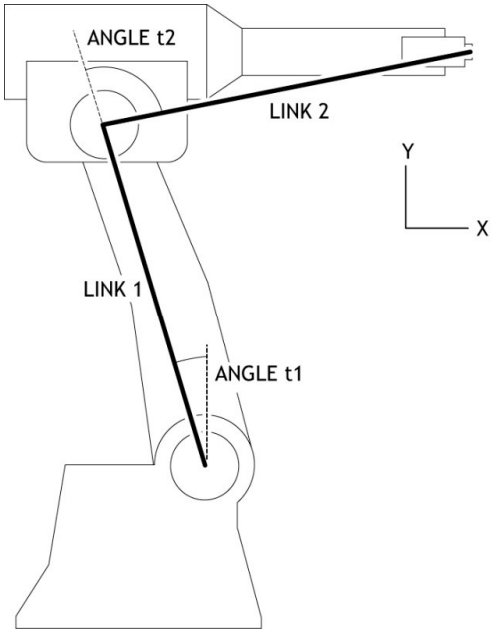
FRAME=15 enables the transformation for a 4 axis **SCARA** robot. This allows you to define the end position of the wrist in X.Y.Z and wrist angle (relative to the Y axis). The frame allows for 2 configurations of a **SCARA** depending if the second axis motor is in the joint or at the base. The difference is that the angle t2 is referenced from link 1, or the angle t2 is referenced from the base. A linkage or belt is typically used to keep t2 referenced to the base.

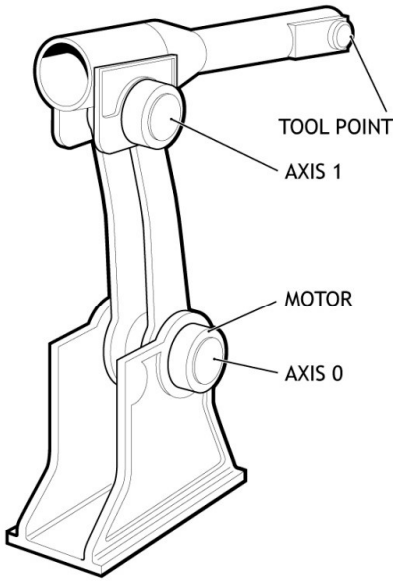
Some mechanical configurations have parasitic motion from the Z axis to the wrist angle. This can be included in the 'ratio' parameter. This is the change in encoder edges on the vertical for a change in wrist angle in encoder edges. Set this value to 0 if there is no parasitic motion.

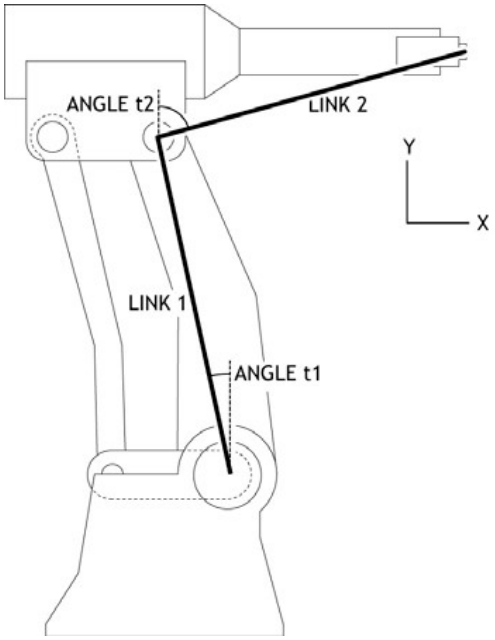
For new projects **FRAME** 115 is recommended

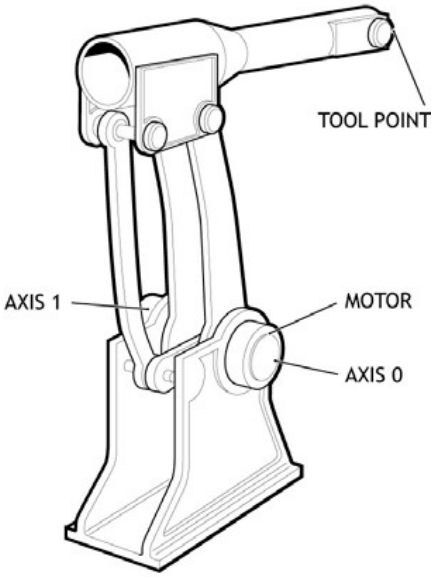
FRAME=15 requires the kinematic runtime **FEC**











Once the frame is enabled **DPOS** on the X,Y and Z axis are measured in Micrometres. The wrist axis is set to use Nanoradians. You can of course set **UNITS** for all axis to any suitable scale.

HOMING

Is it required that the X, Y and wrist absolute positions are homed relative to the “straight up” position before the **FRAME** is enabled. In other words, the zero angle on each axis is with the arms in line and vertical along the Y axis with Z=0. Of course it is not necessary for the motors to actually go to this position as you can offset the position using **DEFPOS** or **OFFPOS**.

JOINT CONFIGURATION

The joint configuration is determined by the position of the **SCARA** arm when you enable **FRAME** = 1

The joint is defined as Right Handed if:

(t2<t1) -both motors in base

(t2<0) -motors in the joint

Otherwise the robot is Left handed

PARAMETERS:

The table data values 0-8 are identical to **FRAME** 1, **SCARA**. This means you can easily switch between the 2 and 4 axis **SCARA**.

Table data	0	link1
	1	link2
	2	Encoder edges/radian axis 0
	3	Encoder edges/radian axis 1
	4	Internal value. Set to 0 to force frame re-calculation
	5	Mechanical configuration
		0 – Both motors fixed in base
		1 – Motors at the joint
	6	Joint configuration (read only)
		0 – Left handed **SCARA**
		1 – Right handed **SCARA**
	7	used internally
	8	used internally
	9	Encoder edges/radian axis 3
	10	link3
	11	Ratio of encoder edges moved on axis 2/ edge axis3
	12	Encoder edges/mm axis 2

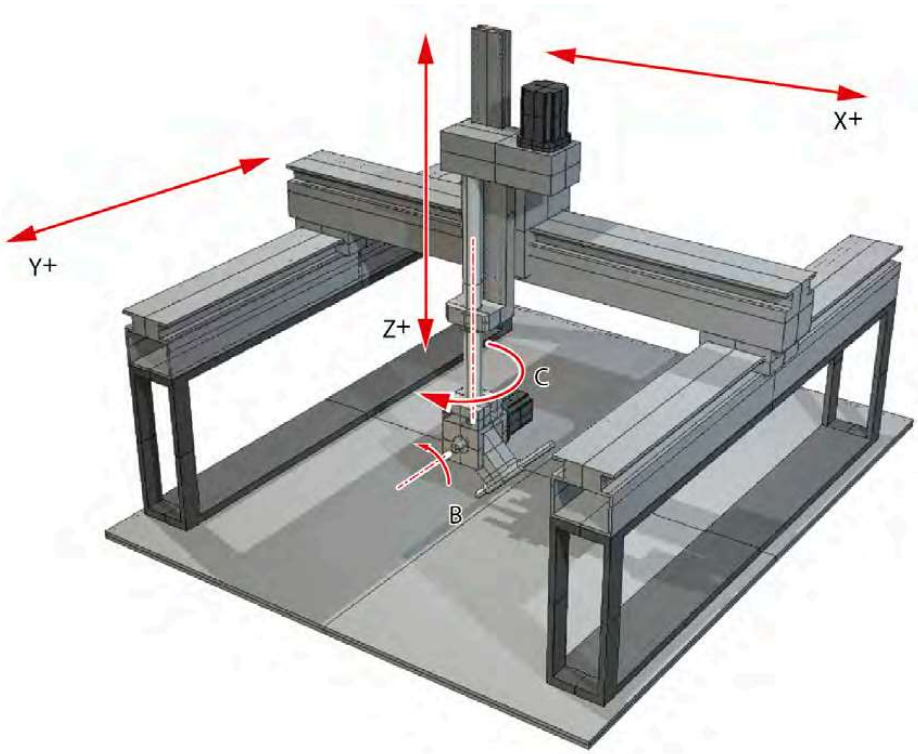
FRAME = 16, 3 AXIS ROBOT WITH 2 AXIS WRIST

DESCRIPTION:

The **FRAME** 16 transformation allows an XYZ Robot with 2 axis wrist to be easily programmed. The transformation function provides compensation in XYZ when the 2 wrist axes are rotated.

For new projects **FRAME** 116 is recommended

FRAME=16 requires the kinematic runtime **FEC**

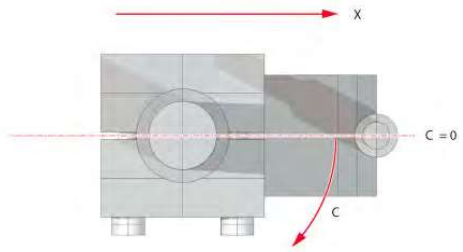


Once the frame is enabled **DPOS** on the X, Y and Z axis are measured in axis counts. The wrist axis is set to use Nanoradians. You can of course set **UNITS** for all axis to any suitable scale.

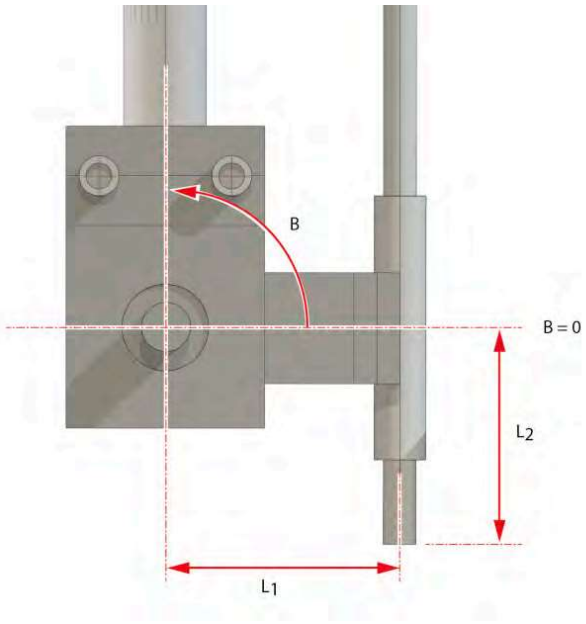
HOMING

Both wrist axes **MUST** be datumed to the correct zero position for the **FRAME** 16 transformation to operate. The zero position of the XYZ axes is not used by the transformation.

The zero position on the C axis (rotation about Z) is when the offset arm is in line with the X axis. The diagram below is drawn from above looking down on to the X-Y plane.



The zero position on the B axis(rotation about Y) is when the offset arm is the “straight down” position shown in the diagram.



The direction of motion on all 5 axes **MUST** match the diagram for the **FRAME** 16 transformation to operate.

If an axis direction of motion is inverted it can be reversed either:

Using the facility of the servo/stepper driver to invert the motion direction

On pulse direction axes using **STEP_RATIO** function inside the *Motion Coordinator*

On closed loop servo axes using **ENCODER_RATIO** / **DAC_SCALE** functions inside the *Motion Coordinator*

PARAMETERS:

Table data	0	Wrist joint to control point X offset (mm) (L1)
	1	Wrist joint to control point Z offset (mm) (L2)
	2	Wrist C axis encoder edges / radian
	3	Wrist B axis encoder edges / radian
	4	X axis encoder edges / mm
	5	Y axis encoder edges / mm
	6	Z axis encoder edges / mm

EXAMPLE:
Configure the table data for a XYZ Cartesian system with a spherical wrist.

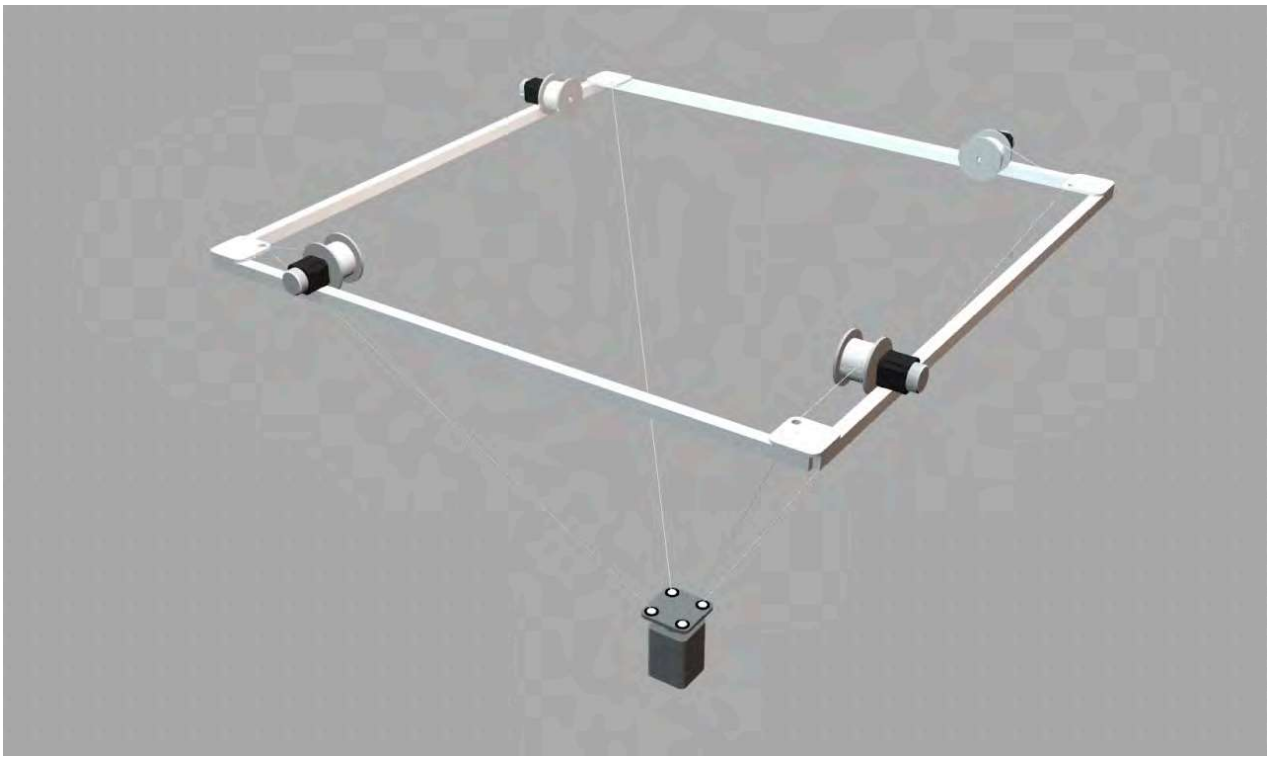
```
' Example:
' Wrist offsets: 60mm in X and 90 mm in Z
' XYZ pulses/mm 1600,1600,2560
' C and B axes pulses radian = 3200 * 16 / (2 * PI)

TABLE(100,60,90,3200 * 8 / PI, 3200 * 8 / PI,1600,1600,2560)

' Set FRAME_GROUP zero using axes 0,1,2,3,4

FRAME_GROUP(0,100,0,1,2,3,4)

FRAME=16

… program moves in XYZBC with tool angle compensation

FRAME=0

… program axes
```

FRAME=17, MULTI-WIRE CAMERA POSITIONING

DESCRIPTION:
The **FRAME** 17 transformation allows a wire mounted stadium camera to be easily programmed. The

transformation function calculates the initial XYZ position of the camera using trilateration from 3 wire mounting points. During running the **FRAME** 17 calculations will calculate the wire lengths for up to 6 support wires with reels mounted in any XYZ positions.

FRAME=114 requires the kinematic runtime **FEC**



HOMING:

The length of wire related to each motor position must be known for the **FRAME** 17 transformation to operate. This requires that the wire winding drums are fitted with absolute encoders or that the system can start from a known position effectively datuming the axes.

PARAMETERS:

0	X axis position of payout position 1	User choice units
1	Y axis position of payout position 1	User choice units
2	Z axis position of payout position 1	User choice units
3	X axis position of payout position 2	User choice units
4	Y axis position of payout position 2	User choice units

5	Z axis position of payout position 2	User choice units
6	X axis position of payout position 3	User choice units
7	Y axis position of payout position 3	User choice units
8	Z axis position of payout position 3	User choice units
9	X axis position of payout position 4 (optional)	User choice units
10	Y axis position of payout position 4 (optional)	User choice units
11	Z axis position of payout position 4 (optional)	User choice units
12	X axis position of payout position 5 (optional)	User choice units
13	Y axis position of payout position 5 (optional)	User choice units
14	Z axis position of payout position 5 (optional)	User choice units
15	X axis position of payout position 6 (optional)	User choice units
16	Y axis position of payout position 6 (optional)	User choice units
17	Z axis position of payout position 6 (optional)	User choice units
18	Edges per user unit payout reel 1	Ratio (E.G. edges/mm)
19	Edges per user unit payout reel 2	Ratio (E.G. edges/mm)
20	Edges per user unit payout reel 3	Ratio (E.G. edges/mm)
21	Edges per user unit payout reel 4 (optional)	Ratio (E.G. edges/mm)
22	Edges per user unit payout reel 5 (optional)	Ratio (E.G. edges/mm)
23	Edges per user unit payout reel 6 (optional)	Ratio (E.G. edges/mm)
24	Option	0 or 1
25	Axes	3..6
26	Scale	Scale User units (see below)
27	Calculation Error	Output 0 (Error) 1 (Solution)

Payout positions: The positions (X,Y,Z) of between 3 and 6 payout positions must be specified to the calculation. These can be in the users choice of units. For example mm

Edges per user unit payout reel: These factors specify the number of encoder edges/user unit for each of the wire payout reels. The user units must be consistent with the payout positions so if the payout positions are specified in metres the edges number specified here must be edges/metre.

Option: The calculation for the camera position from 3 given lengths has 2 potential solutions. (The alternative solution normally requires negative gravity !) The Option parameter should be set to zero or 1 to give the correct solution.

Axes: A minimum of 3 wires are required. The **FRAME** 17 function will calculate the required wire lengths for between 3 and 6 payout drums. Note that the first 3 payouts only are used for calculating the starting position in **XYZ** from the 3 lengths. Where 4 or more wires are used the first 3 specified should be the most critical for the camera position.

Scale: When the **FRAME** 17 is running it calculates **INTEGER** positions in the **XYZ** space for the motion generator program inside the MC4XX. Since the user units (for example metres) are quite large distances a scale factor is required to ensure the integer positions are of fine resolution. The value should give fine resolution but the exact value is not critical. For example if the user units are metres the scale factor should be 100,000 or higher.

Calculation Error: In certain conditions (for example if the length of 1 or more wires is too short) the **FRAME** 17 calculation cannot be performed during the initial trilateration. In this case **TABLE** offset (27) is set to 0. 1 indicates a solution can be calculated.

EXAMPLE:

Test program using the **FRAME_TRANS** function to check correct operation:

```
ATYPE AXIS(0)=0
ATYPE AXIS(1)=0
ATYPE AXIS(2)=0
ATYPE AXIS(3)=0

FRAME_GROUP(1,100,0,1,2,3)

' These positions are in user units (mm for example)

TABLE(100,0,0,0)
TABLE(103,70,0,0)
TABLE(106,70,-40,0)

' 4th axis is not used to calculate starting position

TABLE(109,0,0,0)
TABLE(112,0,0,0)
TABLE(115,0,0,0)

' ratios:

ratio1=1000
ratio2=1000
ratio3=1000
ratio4=1000
```

```
TABLE(118,ratio1,ratio2,ratio3)
TABLE(121,ratio4,ratio5,ratio6)

' option:

scale = 1000

TABLE(124,1)'    solution option (1 or 0)
TABLE(125,4)'    axes 3..6
TABLE(126,1000)' scale factor

' These distances simulate axis positions so should be in edges:
TABLE(200,92.195*ratio1,60*ratio2,72.111*ratio3)

FRAME_TRANS(17,200,300,1,100)' convert wire lengths to XYZ

PRINT TABLE(300),TABLE(301),TABLE(302)

FRAME_TRANS(17,300,400,0,100)' convert XYZ to wire lengths

PRINT TABLE(400)/ratio1,TABLE(401)/ratio2,TABLE(402)/ratio3,TABLE(403)/
ratio4
```

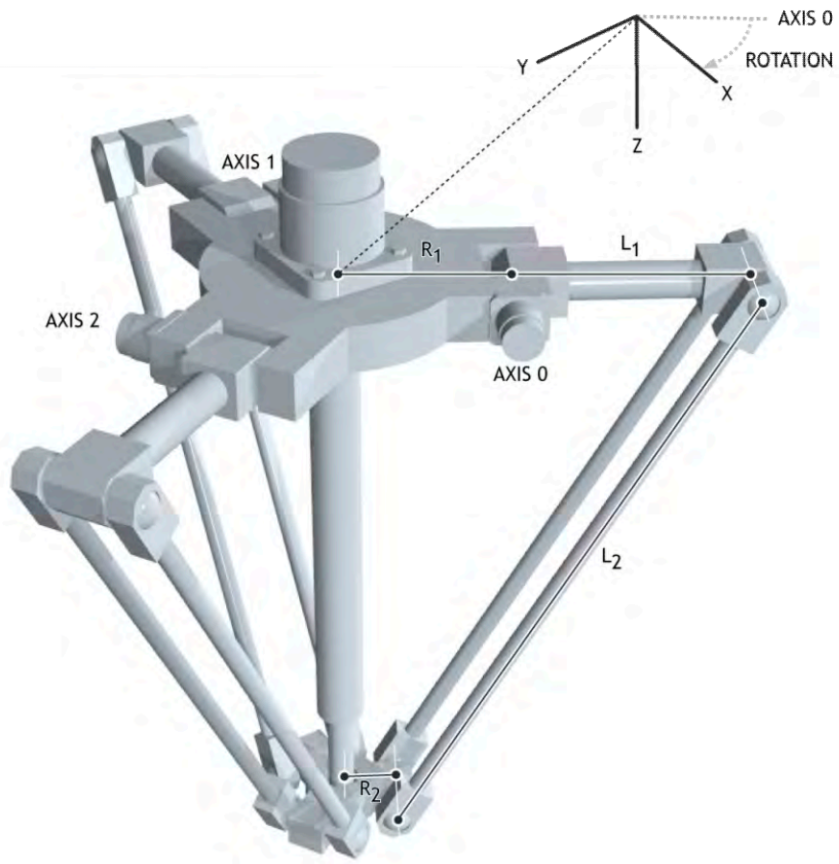
FRAME=18, 6 AXIS ARTICULATED ARM

DESCRIPTION:
Please contact Trio for details.

FRAME=114, DELTA ROBOT

DESCRIPTION:
FRAME=114 enables the high accuracy transformation for a 3 arm ‘delta’ or ‘parallel’ robot. It transforms 3 axes from the mechanical configuration to Cartesian coordinates using the right hand rule.

FRAME=114 requires the kinematic runtime **FEC**



Once the **FRAME** is enabled set the **UNITS** to **FRAME_ANGLE_SCALE** so that the Cartesian movements use the same scale as that used in the table data. So if the **TABLE** data is programmed in mm then when **UNITS** is set to **FRAME_ANGLE_SCALE** then the robot can be programmed in mm.

The origin for the robot is the centre of the top plate with the X direction following the first axis. This can be adjusted using the rotation parameter.

HOMING:

Before enabling **FRAME**=114 the position must be defined so that when the upper arms are horizontal the axis position is 0. You do not need to enable the frame in this position or even move to it, just ensure that it has been defined.

Limits:

-70 to 90 degree

PARAMETERS:

Table data	0	Top radius to joint (R1)
	1	Wrist radius to joint (R2)
	2	Upper arm length (L1)
	3	Lower arm length (L2)
	4	Edges per radian
	5	Angle of rotation in radians (Rotation)
	6	Linkx (optional with 4 or 5 axis)
	7	Linky (optional with 4 or 5 axis)
	8	Linkz (optional with 4 or 5 axis)
	9	Encoder edges/radian (optional Z rotation)
	10	Encoder edges/radian (optional Y rotation)

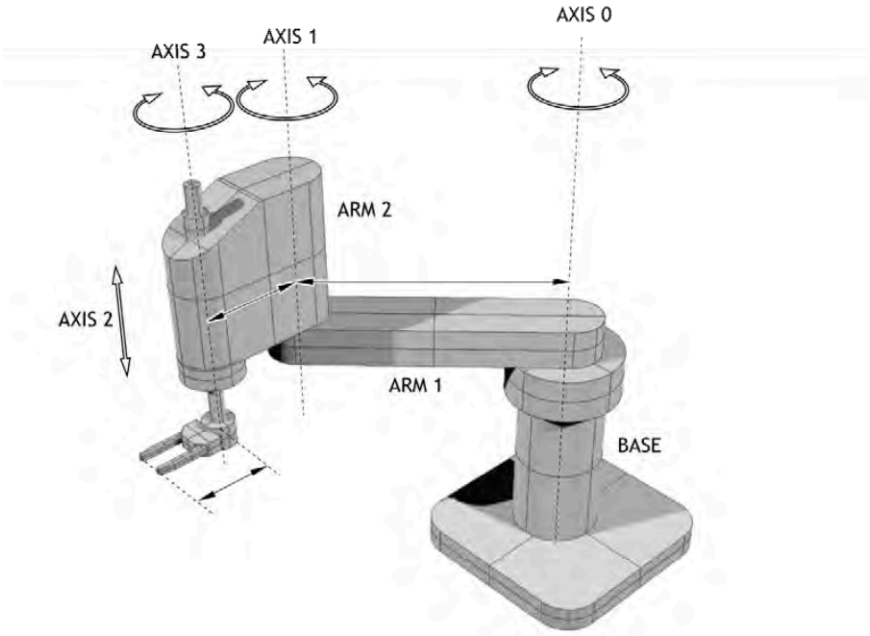
FRAME=115, 3 TO 5 AXIS SCARA

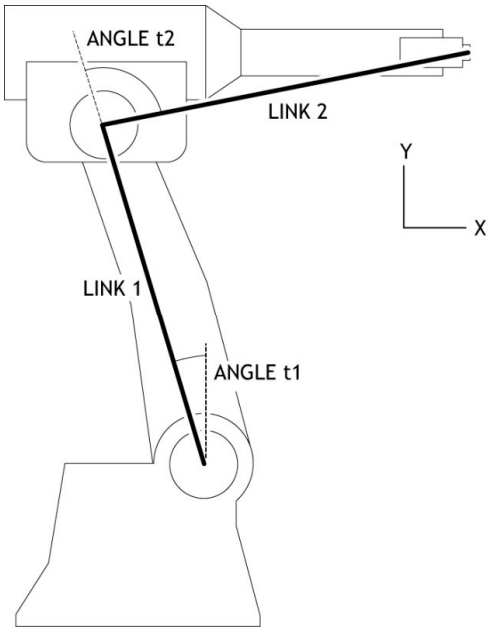
DESCRIPTION:

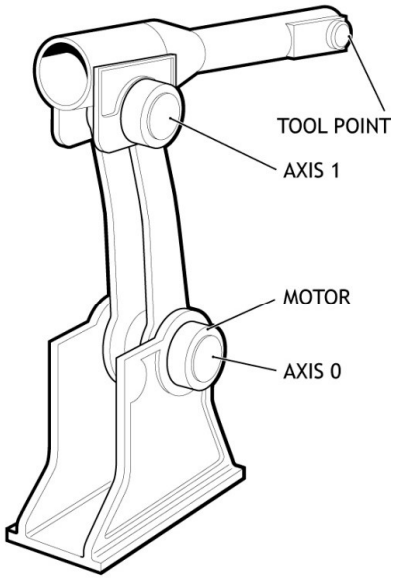
FRAME=115 enables the transformation for a 4 axis **SCARA** robot. This allows you to define the end position of the wrist in X,Y,Z and wrist angle (relative to the Y axis). The frame allows for 2 configurations of a **SCARA** depending if the second axis motor is in the joint or at the base. The difference is that the angle t2 is referenced from link 1, or the angle t2 is referenced from the base. A linkage or belt is typically used to keep t2 referenced to the base.

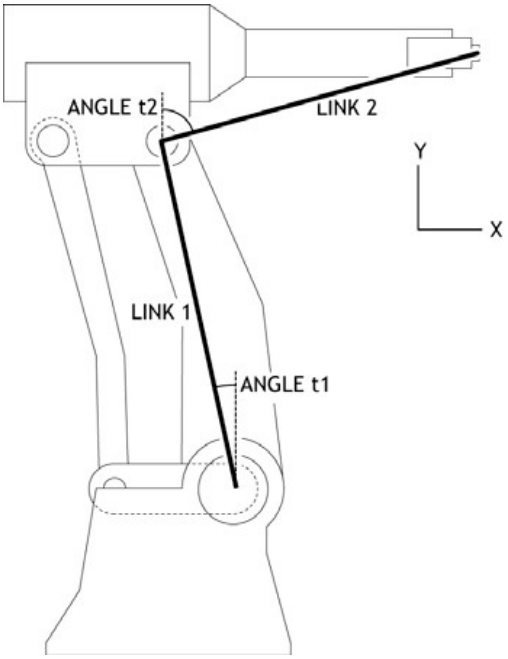
Some mechanical configurations have parasitic motion from the Z axis to the wrist angle. This can be included in the 'ratio' parameter. This is the change in encoder edges on the vertical for a change in wrist angle in encoder edges. Set this value to 0 if there is no parasitic motion.

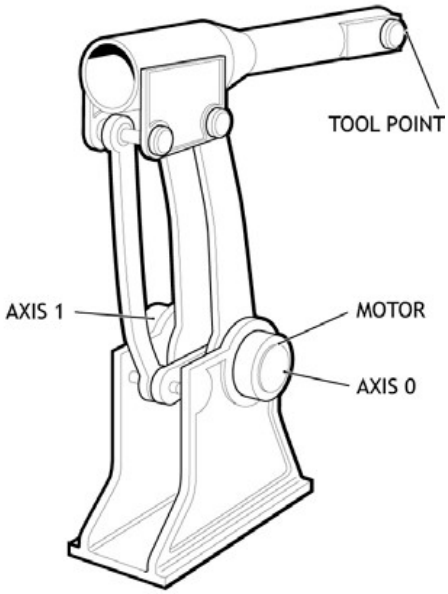
FRAME=115 requires the kinematic runtime **FEC**


AXIS 0
AXIS 1
AXIS 3
ARM 2
AXIS 2
ARM 1
BASE


ANGLE t2
LINK 2
Y
X
LINK 1
ANGLE t1


TOOL POINT
AXIS 1
MOTOR
AXIS 0





Once the **FRAME** is enabled set the **UNITS** to **FRAME_ANGLE_SCALE** so that the Cartesian movements use the same scale as that used in the table data. So if the **TABLE** data is programmed in mm then when **UNITS** is set to **FRAME_ANGLE_SCALE** then the robot can be programmed in mm.

Set the **UNITS** on the rotational (wrist) axes to **FRAME_ANGLE_SCALE** so that they are programmed in radians. You can of course set **UNITS** for all axis to any suitable scale.

HOMING

Is it required that the X, Y and wrist absolute positions are homed relative to the “straight up” position before the **FRAME** is enabled. In other words, the zero angle on each axis is with the arms in line and vertical along the Y axis with Z=0. Of course it is not necessary for the motors to actually go to this position as you can offset the position using **DEFPOS** or **OFFPOS**.

JOINT CONFIGURATION

The joint configuration is determined by the position of the **SCARA** arm when you enable **FRAME** = 1

The joint is defined as Right Handed if:

(t2<t1) –both motors in base

(t2<0) –motors in the joint

Otherwise the robot is Left handed

PARAMETERS:

The table data values 0-8 are identical to **FRAME** 1, **SCARA**. This means you can easily switch between the 2 and 5 axis **SCARA**.

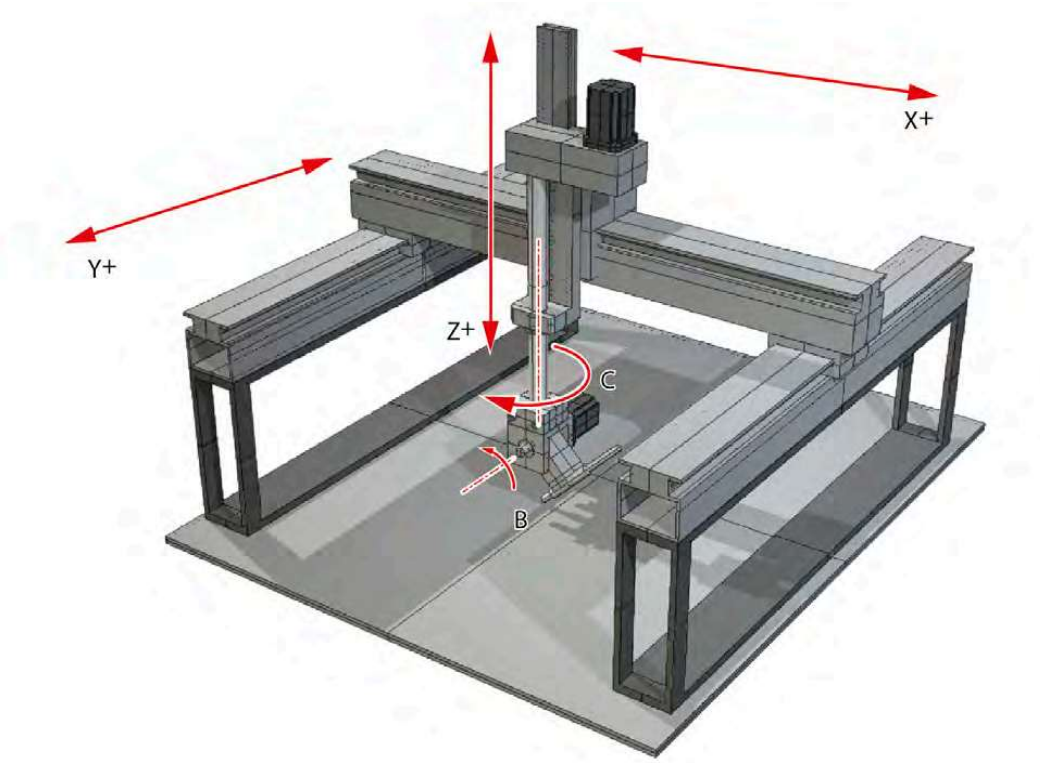
Table data	0	link1
	1	link2
	2	Encoder edges/radian axis 0
	3	Encoder edges/radian axis 1
	4	Mechanical configuration
		0 – Both motors fixed in base
		1 – Motors at the joint
	5	Joint configuration (read only)
		0 – Left handed **SCARA**
		1 – Right handed **SCARA**
	6	Encoder edges/mm axis 2
	7	Ratio of encoder edges moved on axis 2/ edge axis3
	8	Linkx (optional with 4 or 5 axis)
	9	Linky (optional with 4 or 5 axis)
	10	Linkz (optional with 4 or 5 axis)
	11	Encoder edges/radian (optional Z rotation)
	12	Encoder edges/radian (optional Y rotation)

FRAME = 116, 3 AXIS ROBOT WITH 2 AXIS WRIST

DESCRIPTION:

The **FRAME** 116 transformation allows an XYZ Robot with 2 axis wrist to be easily programmed. The transformation function provides compensation in XYZ when the 2 wrist axes are rotated.

FRAME=116 requires the kinematic runtime **FEC**

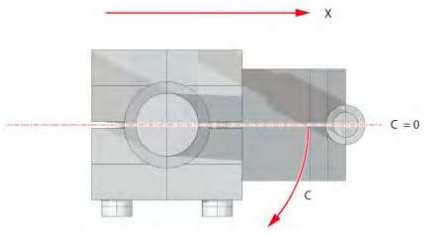


Once the **FRAME** is enabled set the **UNITS** to **FRAME_ANGLE_SCALE** so that the Cartesian movements use the same scale as that used in the table data. So if the **TABLE** data is programmed in mm then when **UNITS** is set to **FRAME_ANGLE_SCALE** then the robot can be programmed in mm.

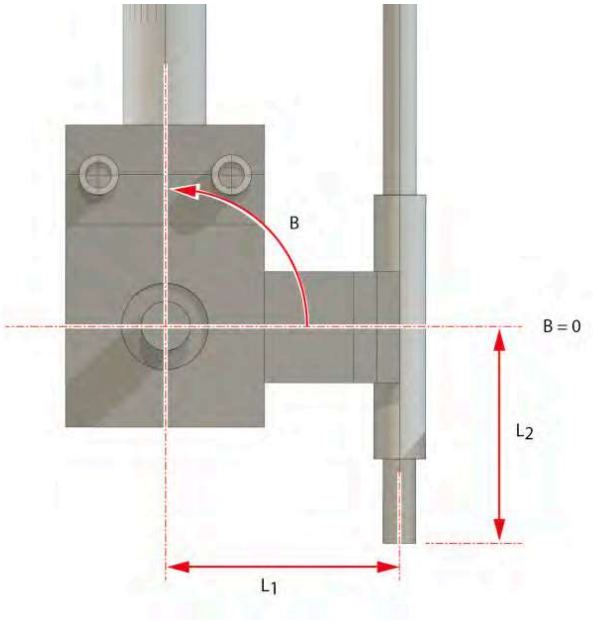
Set the **UNITS** on the rotational (wrist) axes to **FRAME_ANGLE_SCALE** so that they are programmed in radians. You can of course set **UNITS** for all axis to any suitable scale. Homing

Both wrist axes **MUST** be datumed to the correct zero position for the **FRAME** 116 transformation to operate. The zero position of the XYZ axes is not used by the transformation.

The zero position on the C axis (rotation about Z) is when the offset arm is in line with the X axis. The diagram below is drawn from above looking down on to the X-Y plane.



The zero position on the B axis(rotation about Y) is when the offset arm is the “straight down” position shown in the diagram.



The direction of motion on all 5 axes **MUST** match the diagram for the **FRAME** 116 transformation to operate.

- If an axis direction of motion is inverted it can be reversed either:
- Using the facility of the servo/stepper driver to invert the motion direction
- On pulse direction axes using **STEP_RATIO** function inside the *Motion Coordinator*
- On closed loop servo axes using **ENCODER_RATIO** / **DAC_SCALE** functions inside the *Motion Coordinator*

PARAMETERS:

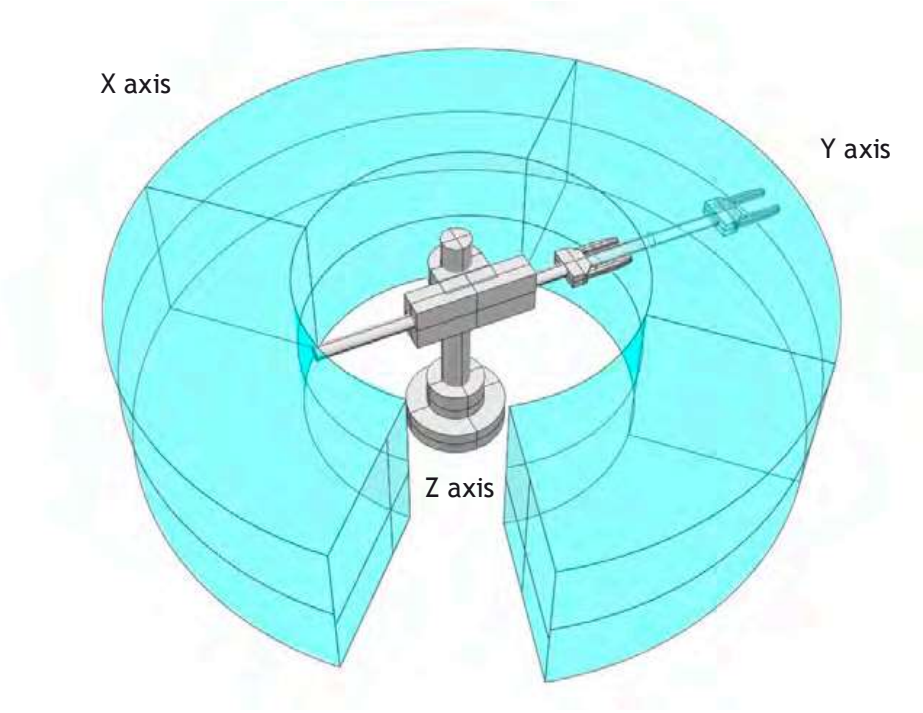
Table data	0	X axis encoder edges / mm
	1	Y axis encoder edges / mm
	2	Z axis encoder edges / mm
	3	Linkx (optional with 4 or 5 axis)
	4	Linky (optional with 4 or 5 axis)
	5	Linkz (optional with 4 or 5 axis)
	6	Encoder edges/radian (optional Z rotation)
	7	Encoder edges/radian (optional Y rotation)

FRAME 119

DESCRIPTION:

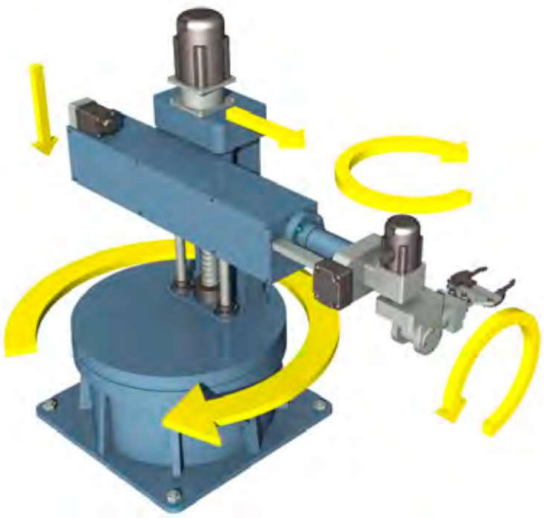
FRAME=119 enables the high accuracy transformation for a 3 axis cylindrical robot with a 2 axis wrist. It has optionally 3 to 5 axes which can be set by **FRAME_GROUP**.

FRAME=119 requires the kinematic runtime **FEC**



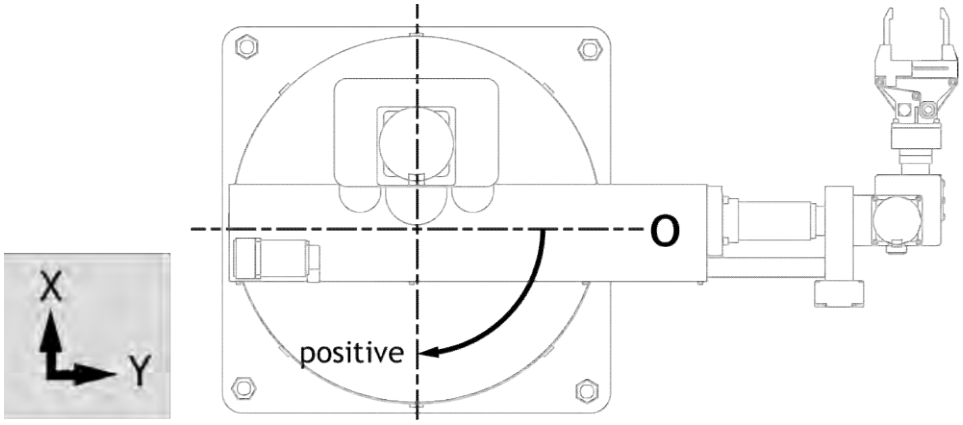
Once the **FRAME** is enabled set the **UNITS** to **FRAME_ANGLE_SCALE** so that the Cartesian movements use the same scale as that used in the table data. So if the **TABLE** data is programmed in mm then when **UNITS** is set to **FRAME_ANGLE_SCALE** then the robot can be programmed in mm.

The origin for the robot is the centre of the rotation axes.



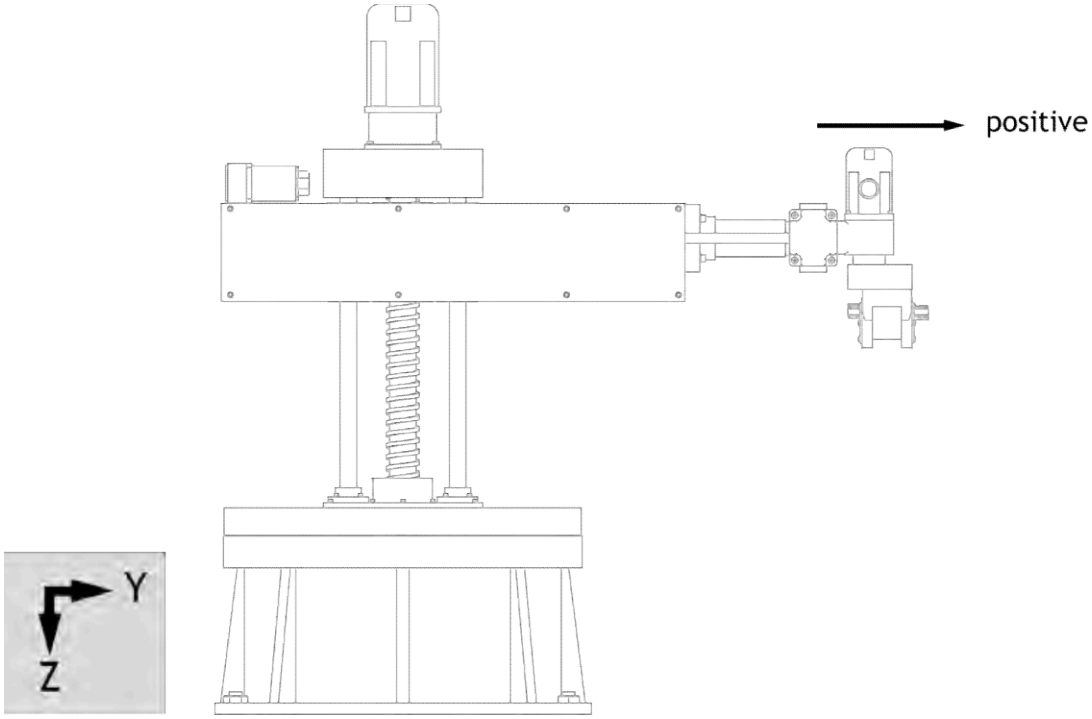
HOMING:

AXIS(0) – BASE ROTATION



Home so that the zero position is along the y axis
Positive direction is clockwise looking from above

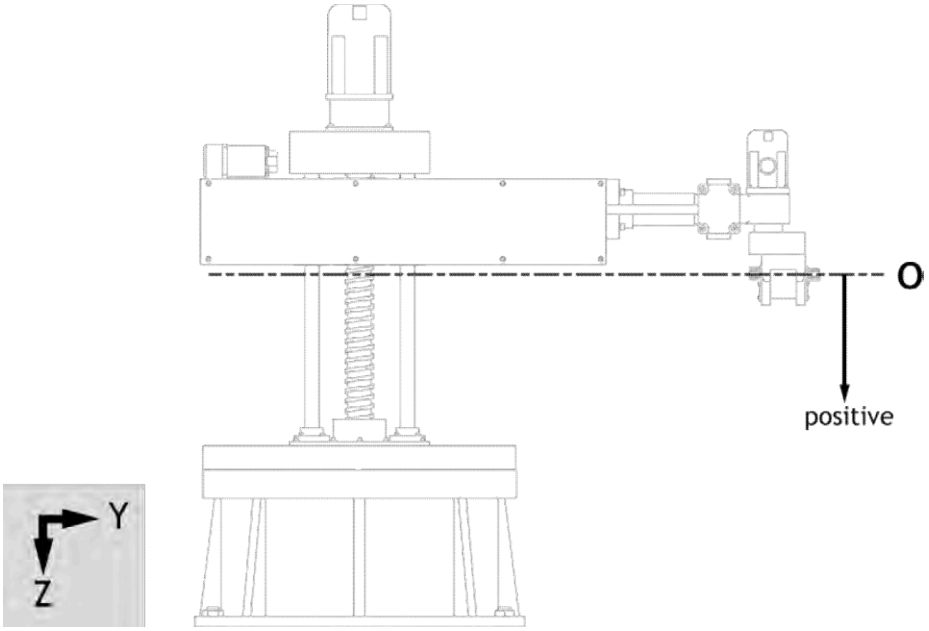
AXIS(1) – ARM EXTENSION



Home with arm at shortest position. Use **DEFPOS** to define the offset from the centre of rotation to the wrist
Positive direction is moving away from centre
Range: greater than zero

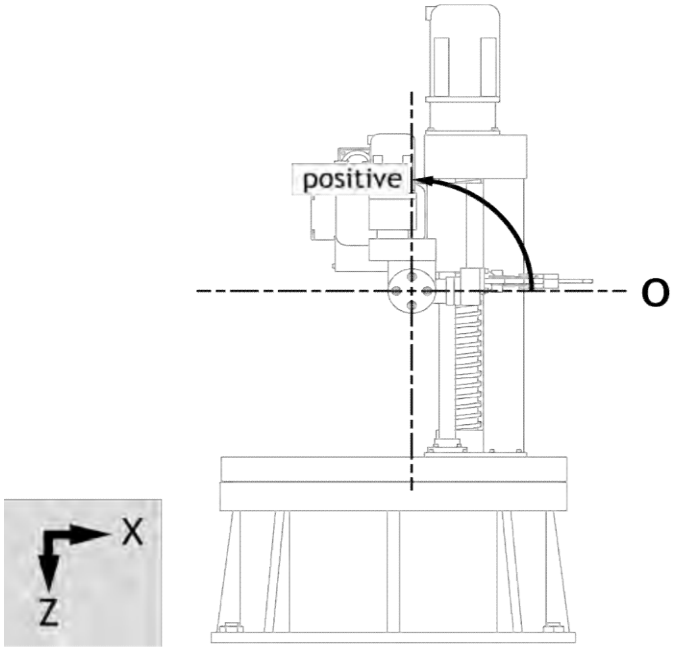
The arm extension must never be allowed to become zero or negative as this will result in a jump in motion. You can set your **RS_LIMIT** to prevent this situation.

AXIS(2) – VERTICAL AXIS



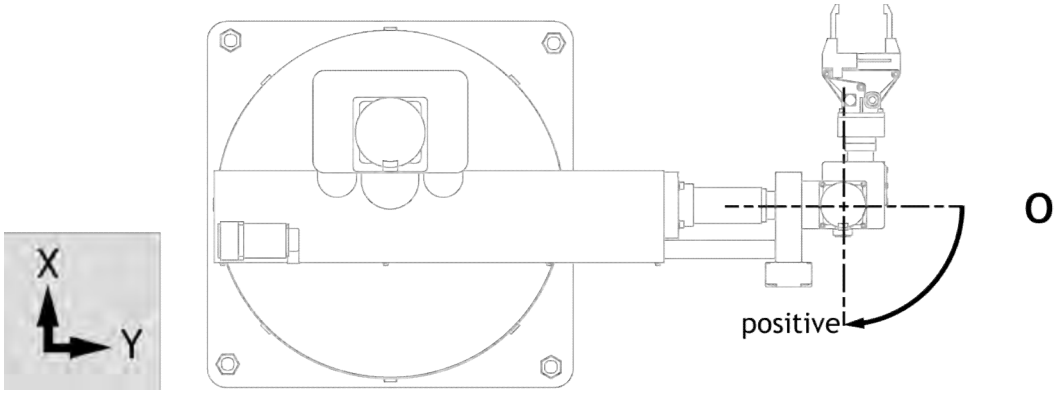
Home with zero at highest position
Positive direction is moving down
Range: 0 to infinite

AXIS(3) – WRIST ROTATE ABOUT Y



Home so that the wrist is horizontal
Range: - infinite to infinite

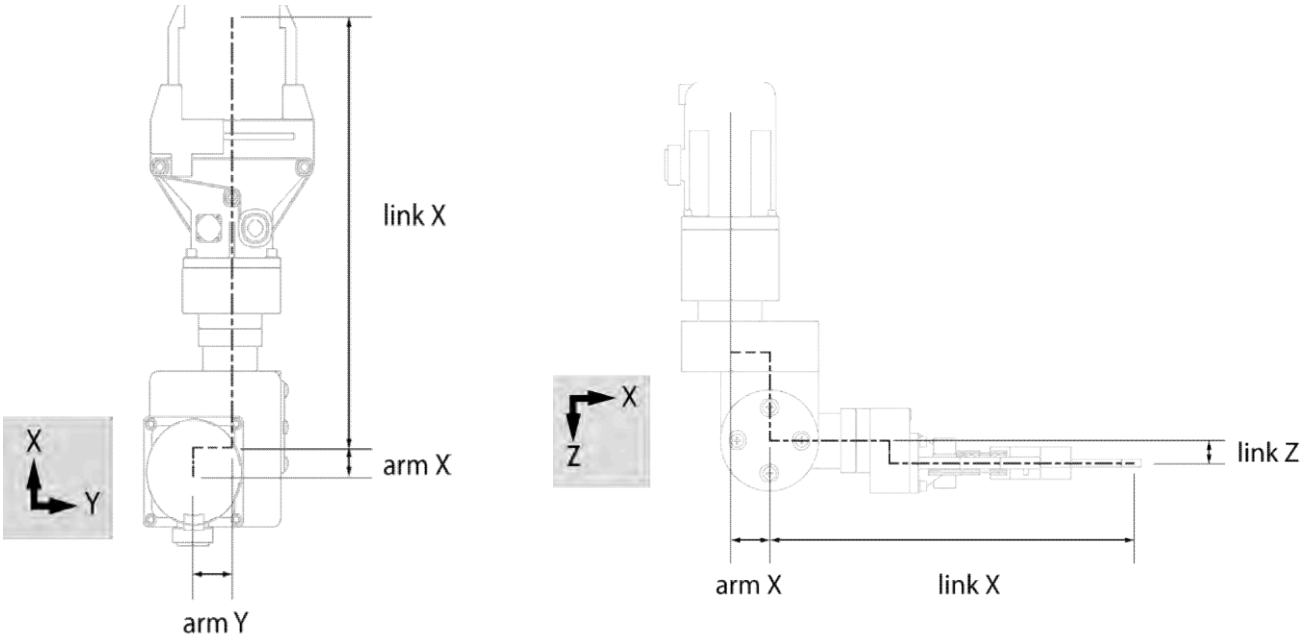
AXIS(4) – WRIST ROTATE ABOUT Z



Home so that the zero position is along the y axis
Positive direction is clockwise looking from above
Range: - infinite to infinite

PARAMETERS:

Table data	0	Edges per radian (base rotation)
	1	Edges per mm (arm extension)
	2	Edges per mm (vertical axis)
	3	Revolutions – set to 0
	4	Previous position – set to 0
	5	Linkx (optional with 4 or 5 axis)
	6	Linky (optional with 4 or 5 axis)
	7	Linkz (optional with 4 or 5 axis)
	8	Encoder edges/radian (optional Z rotation)
	9	Encoder edges/radian (optional Y rotation)



EXAMPLES:

EXAMPLE 1:

This example sets up a 5 axis system

```
linkx = 50'mm
linky = 50'mm
linkz = 50'mm

t1_encoder = 4*17000 'Encoder counts per revolution
t1_gearbox = 50
t1_edges_per_radian = t1_encoder * t1_gearbox / (2 * PI)
t1_edges_per_degree = t1_encoder * t1_gearbox / (360)

t2_encoder = 4*250 'Encoder counts per revolution
t2_gearbox = 1
t2_mm_per_rev = 1
t2_edges_per_mm = t2_encoder * t2_gearbox / t2_mm_per_rev

t3_encoder = 4*250 'Encoder counts per revolution
t3_gearbox = 1
t3_mm_per_rev = 1
t3_edges_per_mm = t3_encoder * t3_gearbox / t3_mm_per_rev

t4_encoder = 4*16000 'Encoder counts per revolution
t4_gearbox = 50
t4_edges_per_radian = t4_encoder * t4_gearbox / (2 * PI)
t4_edges_per_degree = t4_encoder * t4_gearbox / (360)

t5_encoder = 4*16000 'Encoder counts per revolution
t5_gearbox = 50
t5_edges_per_radian = t4_encoder * t4_gearbox / (2 * PI)
t5_edges_per_degree = t4_encoder * t4_gearbox / (360)

revolutions = 0
prev_pos = 0

group_size = 5

TABLE(0, t1_edges_per_radian, t2_edges_per_mm, t3_edges_per_mm,
revolutions, prev_pos)
TABLE(5, linkx, linky, linkz, t4_edges_per_radian, t5_edges_per_
radian)
FRAME_GROUP(0,0,0,1,2,3,4)
BASE(0)
UNITS =FRAME_SCALE 'mm
BASE(1)
UNITS =FRAME_SCALE 'mm
BASE(2)
UNITS =FRAME_SCALE 'mm
```

```
BASE(3)
UNITS =(FRAME_SCALE * 2 * PI) / (360)'degrees
BASE(4)
UNITS =(FRAME_SCALE * 2 * PI) / (360)'degrees

BASE(0,1,2)
MOVE(-100,-100,100)
MOVE(200,0)
MOVE(-100,100,-100)
BASE(0,1,zrot)
MHELICAL(0,0,0,-50,0,360,1)
MOVE(0,25,0)
MOVECIRC(0,0,0,-75,0)
```

EXAMPLE 1:

This example sets up a 4 axis system

```
linkx = 50'mm
linky = 50'mm
linkz = 50'mm

t1_encoder = 4*17000 'Encoder counts per revolution
t1_gearbox = 50
t1_edges_per_radian = t1_encoder * t1_gearbox / (2 * PI)
t1_edges_per_degree = t1_encoder * t1_gearbox / (360)

t2_encoder = 4*250 'Encoder counts per revolution
t2_gearbox = 1
t2_mm_per_rev = 1
t2_edges_per_mm = t2_encoder * t2_gearbox / t2_mm_per_rev

t3_encoder = 4*250 'Encoder counts per revolution
t3_gearbox = 1
t3_mm_per_rev = 1
t3_edges_per_mm = t3_encoder * t3_gearbox / t3_mm_per_rev

t4_encoder = 4*16000 'Encoder counts per revolution
t4_gearbox = 50
t4_edges_per_radian = t4_encoder * t4_gearbox / (2 * PI)
t4_edges_per_degree = t4_encoder * t4_gearbox / (360)

revolutions = 0
prev_pos = 0

group_size = 4

TABLE(0, t1_edges_per_radian, t2_edges_per_mm, t3_edges_per_mm,
revolutions, prev_pos)
TABLE(5, linkx, linky, linkz, t4_edges_per_radian)
```

```
FRAME_GROUP(0,0,0,1,2,3)
BASE(0)
  UNITS =FRAME_SCALE 'mm
  BASE(1)
  UNITS =FRAME_SCALE 'mm
  BASE(2)
  UNITS =FRAME_SCALE 'mm
  BASE(3)
  UNITS =(FRAME_SCALE * 2 * PI) / (360)'degrees
```

FRAME_GROUP

TYPE:
System Command

SYNTAX:
FRAME_GROUP(group, [table_offset, [axis0, axis1 ...axisn]])

DESCRIPTION:
FRAME_GROUP is used to define the group of axes and the table offset which are used in a **FRAME** or **USER_FRAME** transformation. There are 8 groups available meaning that you can run a maximum of 8 FRAMEs on the controller.

FRAME_GROUP requires the kinematic runtime **FEC**

Although 8 **FRAME**s 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 **FRAME** is selected, drive connection method, if **USER_FRAME** and **TOOL_OFFSET** are enabled and additional factory communications.

The number of axes in the group must match the number of axes used by the **FRAME**. The axes must also be ascending order though they do not have to be contiguous. If a group is deleted **FRAME** and **USER_FRAME** are set to 0 for those axes.

To maintain backward compatibility if the **FRAME** command is used on an axis that is not in a group, or no groups are configured then a default group is created using the lowest axes and table_offset=0. In this situation if **FRAME_GROUP**(0) is already configured it is overwritten.

When the group is deleted **FRAME** is set to 0, **USER_FRAME**(0) is activated, **TOOL_OFFSET**(0) is activated and **VOLUME_LIMIT**(0) is activated. This means you can delete the **FRAME_GROUP** to reset all of these commands.

PARAMETERS:

group:	The group number, 0-7. When used as the only parameter **FRAME_GROUP** prints the **FRAME_GROUP**, the active **USER_FRAME** and **TOOL_OFFSET** information to the currently selected output channel (default channel 0)
table_offset:	-1 = Delete group data
	0+ = The start position in the table to store the **FRAME** configuration.
axis0:	The first axis in the group
axis1:	The second axis in the group
axisn:	The last axis in the group

The text returned when only printing **FRAME_GROUP** is in the following format:

group [table_offset] : axes {**USER_FRAME**: **USER_FRAME** parameters} TO={**TOOL_OFFSET** : **TOOL_OFFSET** parameters} VL={**VOLUME_LIMIT** parameters}

EXAMPLES:

EXAMPLE 1:

Configure a **FRAME_GROUP** for axes 1,2 and 5 using table offset 100.

```
'Initialise the FRAME_GROUP
FRAME_GROUP(0,100, 1,2,5)

'Configure the axes, FRAME table data and home the robot
GOSUB configure_frame

'PRINT the FRAME_GROUP information to the command line
FRAME_GROUP(0)

'Enable the frame
FRAME AXIS(1)=14
```

EXAMPLE 2:

Reset the **FRAME_GROUP** to set: **USER_FRAME**(0), **TOOL_OFFSET**(0), **FRAME** = 0 and **VOLUME_LIMIT**(0)

```
BASE(0) 'Select an axis in the FRAME_GROUP
FRAME_GROUP(0,-1)
```

EXAMPLE 3:

Print the **FRAME_GROUP** in the terminal.

```
>>FRAME_GROUP(0,1,2,3,4,5)
>>PRINT FRAME_GROUP(0)
0 [1] : 2, 3, 4, 5 {0:0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
0.00000} TO={0
```

```
:0.00000, 0.00000, 0.00000} VL={0, 0}
0
```

FRAME_REP_DIST

TYPE:
Axis Parameter

DESCRIPTION:
Orientation axes on a **FRAME** or **USER_FRAME** must be programmed between ± half a revolution (**UNITS** can be used to set radians, degrees etc). This cannot be done using **REP_DIST** and has to be done using **FRAME_REP_DIST** and **REP_OPTION** bit 3.

When this is configured the **DPOS** will wrap to ± half a revolution and **AXIS_DPOS** will not be wrapped so that the absolute axis position is maintained.

Wrapping will only occur when **FRAME** <> 0 or **USER_FRAME** <> 0. While both are set to zero the wrapping will be inhibited so that the absolute axis position is maintained.

With **REP_OPTION** bit 3 set and **DPOS** exceeding **FRAME_REP_DIST** it will wrap to -**FRAME_REP_DIST**. The same applies in reverse so when **DPOS** exceeds -**FRAME_REP_DIST** it will wrap to **FRAME_REP_DIST**.

VALUE:
The position in user **UNITS** where the axis position wraps.

FRAME_REP_DIST uses *UNITS*. You must remember to set **FRAME_REP_DIST** while the correct *UNITS* are active.

EXAMPLES:
A 4 axis delta robot has one orientation axis which is the angle of rotation about the Z axis. The user is programming in degrees so the **DPOS** must be limited to ±180 degrees.

```
BASE(axis_w)
UNITS = (FRAME_SCALE*2*PI) / 360 'degrees
FRAME_REP_DIST = 180
REP_OPTION = 8
```

SEE ALSO:
REP_OPTION

FRAME_ SCALE

TYPE:
Axis Parameter

DESCRIPTION:
FRAME_ SCALE is used to adjust the resolution of the high accuracy FRAMEs (100+). The default value is very large and so the accuracy is sufficient for most applications.

VALUE:
Default value 1000000000

FRAME_TRANS

TYPE:
Mathematical Function

SYNTAX:
FRAME_TRANS(frame, table_in, table_out, direction [,table_offset])

DESCRIPTION:
This function enables you to perform both the forward and inverse transformation calculations of a **FRAME**. One particular use is to check following errors in user units or to calculate positions outside of the **FRAME** working area.

FRAME_TRANS requires the kinematic runtime **FEC** to use a **FRAME** 14 and higher.

The **FRAME** calculations are performed on raw position data. When using a **FRAME** typically the raw position data for **DPOS** is micrometres and the raw position data for **MPOS** is encoder counts but this can vary depending on which **FRAME** you select.

PARAMETERS:

frame:	The **FRAME** number to run

table_in	The start position in the **TABLE** of the input positions
table_out	The start position in the **TABLE** of the generated positions
direction	1 = **AXIS_DPOS** to **DPOS** (Forward Kinematics)
	0 = **DPOS** to **AXIS_DPOS** (Inverse Kinematics)
table_offset	The first position in the table where the frame configuration is found (default 0)

EXAMPLES:

EXAMPLE 1:

Using **MPOS** calculate the Cartesian values so you can compare them to **DPOS**. This can be used to check the following error in the world coordinate system. The frame configuration is stored in the table starting at position 100.


```
'Load positions into the table
FOR x=0 TO 3
BASE(x)
TABLE(1000+x,MPOS AXIS(x)*UNITS AXIS(x))
NEXT x
'Calculate forward transform to see MPOS is Cartesian coordinates
FRAME_TRANS(15, 1000,2000,1,100)

TABLE(3000, TABLE(2000)/ UNITS AXIS(0))
TABLE(3001, TABLE(2001)/ UNITS AXIS(1))
TABLE(3002, TABLE(2002)/ UNITS AXIS(2))
PRINT «DPOS IN ENCODER COUNTS»,TABLE(2000),TABLE(2001),TABLE(2002)
PRINT «DPOS IN MM»,TABLE(3000),TABLE(3001),TABLE(3002)
PRINT «FE in world x = «, TABLE(3000) – DPOS AXIS(0)
PRINT «FE in world y = «, TABLE(3001) – DPOS AXIS(1)
PRINT «FE in world z = «, TABLE(3002) – DPOS AXIS(2)
```

EXAMPLE 2:

Use the inverse kinematics to confirm that a demand position will result in an axis position that the motors can achieve.


```
'Load positions into the table
TABLE(5000,100*UNITS AXIS(0),200*UNITS AXIS(1),400*UNITS AXIS(2))
```

```
'Calculate reverse transform to see
FRAME_TRANS(14, 5000,6000,0)

'Divide the result by the AXIS_UNITS to get
'the MPOS in degrees
TABLE(7000, TABLE(6000)/ AXIS_UNITS)
TABLE(7001, TABLE(6001)/ AXIS_UNITS)
TABLE(7002, TABLE(6002)/ AXIS_UNITS)

PRINT "MPOS RAW ENCODER COUNTS", TABLE(6000),TABLE(6001),TABLE(6002)
PRINT "MPOS degrees", TABLE(7000),TABLE(7001),TABLE(7002)
```

FREE

TYPE:
System Parameter (Read Only)

DESCRIPTION:
Returns the amount of program memory available for user programs.

Each line takes a minimum of 4 characters (bytes) in memory. This is for the length of this line, the length of the previous line, number of spaces at the beginning of the line and a single command token. Additional commands need one byte per token, most other data is held as **ASCII**.

The *Motion Coordinator* compiles programs before they are run, this means that a little under twice the memory is required to be able to run a program.

VALUE:
The amount of available user memory in bytes.

EXAMPLE:
Check the available memory on the command line

```
>>PRINT FREE
47104.0000
>>
```

SEE ALSO:
DIR