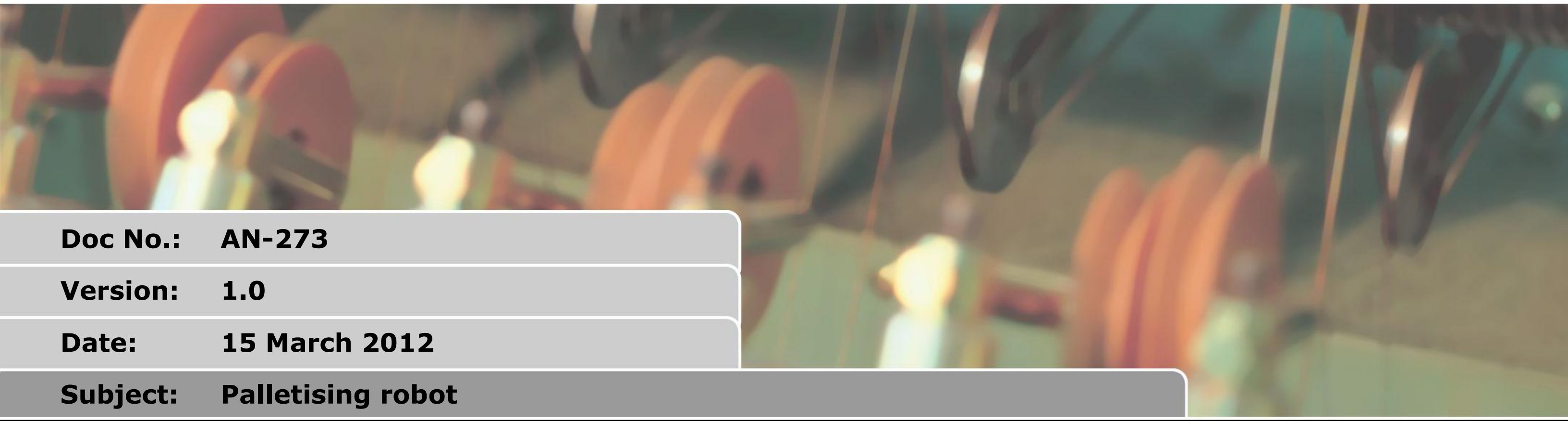
Trio Motion Technology Ltd.
Shannon Way, Tewkesbury,
Gloucestershire. GL20 8ND
United Kingdom
Tel: +44 (0)1684 292 333
Fax: +44 (0)1684 297 929

1000 Gamma Drive
Suite 206
Pittsburgh, PA 15238
United States of America
Tel: +1 412 968 9744
Fax: +1 412 968 9746

B1602 Tomson Centre
188 Zhang Yang Rd.,
Pudong New Area, Shanghai,
Postal code: 200122
CHINA
Tel/Fax: +86 21 587 97659



Doc No.: **AN-273**

Version: **1.0**

Date: **15 March 2012**

Subject: **Palletising robot**

APPLICATION NOTE

1. Introduction

This application note discusses the program needed to make a very simple palletising robot. Many different types of robot could be used but in this example it uses a scale and rotate robot. It would be very simple to change to a different robot type as detailed in the sections below.

2. Scale and rotate robot.

This robot type has 4 degrees of freedom it uses a mechanical arrangement so that the axes are as follows:

1. Arm reach
2. Arm height
3. Shoulder rotate
4. Wrist rotate

2.1. Transformation

To enable simple programming in Cartesian coordinates it is possible to make a transformation in TrioBASIC. To do this local variables are used to input the Cartesian positions and output the axis position.

5. x_position – Cartesian X position
6. y_position – Cartesian Y position
7. z_position – Cartesian Z position
8. wrist_angle_abs – Absolute angle of the wrist against the X axis (programmed in degrees in this example)

9. base_angle – angle of the shoulder (programmed in degrees in this example)
10. wrist_angle – angle of the wrist relative to the arm (programmed in degrees in this example)
11. r_position – reach of the arm

The z_position does not need any transformation as it is a 1 to 1 relationship

The actual transformation mathematics from the Cartesian to the axes positions is as follows:

```
'This is the scale and rotate robot transformation
 base_angle = (ATAN2(y_position, x_position)*360)/ (2* PI)
 wrist_angle = base_angle - wrist_angle_abs
 r_position = SQR(x_position * x_position + y_position * y_position)
```

The arm height and arm reach use a mechanical linkage to amplify the movement from the motor to the height and reach. The example program uses UNITS to scale the counts per mm of linier movement to the counts per mm of the tool tip movement as follows:

```
BASE(r_axis)
counts_x_per_mm = 10000 'counts per mm of screw feed
scale_x = 10.3 'scale from screw feed to end point movement
UNITS = counts_x_per_mm / scale_x

BASE(z_axis)
counts_z_per_mm = 10000 'counts per mm of screw feed
scale_z = 10.3 'scale from screw feed to end point movement
UNITS = counts_z_per_mm / scale_z
```

Similarly the rotate axes have been configured using UNITS so that they can be programmed in mm:

```
BASE(base_axis)
counts_base_per_rev = 10000 'counts per revolution
UNITS = counts_base_per_rev/360 'counts per degree
'Set the axis to work in +-180 degrees
REP_DIST = 180
REP_OPTION = 0

BASE(wrist_axis)

counts_wrist_per_ rev = 10000 'counts per revolution
UNITS = counts_wrist_per_rev/360 'counts per degree
'Set the axis to work in +-180 degrees
REP_DIST = 180
REP_OPTION = 0
```

2.2. Movements

To simplify the programming a sub routine is used to perform the transformation and move the axes. The routine performs the transformation then loads the output into a MOVEABS. As discussed later in this document the positions are stored in the TABLE so a local variable 'position' is set before entering the sub routine so that the correct position can be moved to.

```
'****************************************
move_robot:
'****************************************
  'load positions from the table
  x_position = (TABLE(table_start + position * 4))
  y_position = (TABLE(table_start + position * 4 + 1))
  z_position = (TABLE(table_start + position * 4 + 2))
  wrist_angle_abs = (TABLE(table_start + position * 4 + 3))

  'Calculate the wrist and base angle
  'This is the scale and rotate robot transformation
  base_angle = (ATAN2(y_position, x_position)*360)/ (2* PI)
  wrist_angle = base_angle - wrist_angle_abs
  r_position = SQR(x_position * x_position + y_position * y_position)
  'Move the robot
  BASE(r_axis, z_axis, base_axis, wrist_axis)
  MOVEABS(r_position, z_position, base_angle, wrist_angle)
  WAIT IDLE
RETURN
```

2.3. Homing

For the transformation to work the robot must be homed so that the arm and wrist are in line with the x axis. The reach should be homed so that the zero position is minimum reach. The vertical should be homed so that the zero position is the lowest position that it can reach.

In the example program this is using the DATUM command and DATUM_IN switches. The arm is homed first so that is moves to the closest then lowest position. Finally both rotate axes are homed at the same time. Then the offset is applied so that the zero position is in line with the x axis.

```
'****************************************
home_robot:
'****************************************
  BASE(r_axis)
  DATUM_IN = r_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine
  WAIT IDLE
  FS_LIMIT = 1200
  RS_LIMIT = 0

  BASE(z_axis)
  DATUM_IN = z_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine
  WAIT IDLE
  FS_LIMIT = 750
  RS_LIMIT = 0

  BASE(base_axis)
  DATUM_IN = base_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine

  BASE(wrist_axis)
  DATUM_IN = wrist_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine
  WAIT IDLE
  DEFPOS(-170)
  FS_LIMIT = 170
  RS_LIMIT = -170

  BASE(base_axis)
  WAIT IDLE
  DEFPOS(-170)
  FS_LIMIT = 170
  RS_LIMIT = -170
RETURN
```

You can see that in the above example once the positions are homed and defined then the software limits RS_LIMIT and FS_LIMIT are enabled to prevent over reaching of the arm and over rotation of the rotary axes.

3. Pick and Place application

The pick and place example here is picking up bags of rice which are coming in on a conveyor then placing them on a pallet. The pallet will hold 6 bags per layer and the layers must be alternated so that the pattern varies making the stacking more stable.

3.1. Storing positions in the TABLE

The positions are all stored in the table. This example uses fixed set of positions though it would be fairly easy to modify it to accept positions loaded from an HMI or even learnt from manually moving the robot to a position.

The load_positions sub routine loads table using the following format:

```
TABLE(table_start * position, x_position , y_position, z_position,
wrist_angle_abs)
```

So the positions are loaded as follows:

```
'Positin 0, pick position
TABLE(table_start + 0 * 4, 600 , -600, 200, 45)

'Position 1 - 6, 'horizontal layer of palletizing
TABLE(table_start + 1 * 4, 300 , 150, 700, 0)
TABLE(table_start + 2 * 4, 500 , 150, 700, 0)
TABLE(table_start + 3 * 4, 700 , 150, 700, 0)
TABLE(table_start + 4 * 4, 300 , 450, 700, 0)
TABLE(table_start + 5 * 4, 500 , 450, 700, 0)
TABLE(table_start + 6 * 4, 700 , 450, 700, 0)

'Position 7 - 12, 'horizontal layer of palletizing
TABLE(table_start + 7 * 4, 350 , 100, 700, 90)
TABLE(table_start + 8 * 4, 650 , 100, 700, 90)
TABLE(table_start + 9 * 4, 350 , 300, 700, 90)
TABLE(table_start + 10 * 4, 650 , 300, 700, 90)
TABLE(table_start + 11 * 4, 350 , 500, 700, 90)
TABLE(table_start + 12 * 4, 650 , 500, 700, 90)
```

3.2. Main program loop

The main program loop simply performs a pick, place, increments layer and bag position. If the pallet is full then it reloads the pallet. To make the program easy to read many sub routines are used.

```
bag_position=1 'first bag position on pallet
bag_layer = 0 'start with the first layer
bag_height = 100 'initial drop height for bag
pick_height = 100 'height for picking the bags

WHILE IN(machine_enabled) = ON
  GOSUB pick_bag
  IF bag_position = 1 OR bag_position = 7 THEN
    bag_layer = bag_layer + 1
  ENDIF
  place_height = bag_height * layer
  GOSUB place_bag
      bag_position = bag_position + 1
  IF bag_position = 13 THEN
    IF bag_layer = 5 THEN
      GOSUB reload_pallet
    ENDIF
    bag_position = 1
   ENDIF
WEND
```

3.3. pick_bag sub routine

This subroutine moves to the pick position, lowers the arm then waits for a bag to arrive. When the bag has been detected the jaws close and the arm rises. An output is used to close and open the jaws. An input is used to sense when the jaws are closed.

```
'****************************************
pick_bag:
```

```
'**************************************
  'Move to pick position
  position = 0
  GOSUB move_robot
  z_position = pick_height - MPOS AXIS(z_axis)
  'move down to the pick height
  MOVE(0, z_position, 0, 0)
  WAIT IDLE
  'wait for a bag to pick
  WAIT UNTIL IN(bag_loaded)=ON 'wait for bag in pick position
  OP(jaws, ON) 'close jaws to pick up bag
  'wait for the sensor to detect jaws are closed around the bag
  WAIT UNTIL IN(jaws_closed) = ON
  'move back up
  position = 0
  GOSUB move_robot
RETURN
```

3.4. place_bag sub routine

The main loop has already calculated which position on the pallet to place the bag. This routine will move to this position, lower the arm, open the jaws. Then when the sensor detects that the bag has been released the arm raises again.

```
'**************************************
place_bag:
'**************************************
  'Move to place position
  position = bag_position
  GOSUB move_robot
  BASE(r_axis, z_axis, base_axis, wrist_axis)
  z_position = place_height - MPOS AXIS(z_axis)
  'move down to the pick height
  MOVE(0, z_position, 0, 0)
  WAIT IDLE
  OP(jaws, OFF) 'open jaws to replease the bag
  'wait for the sensor to detect that the jaws are open
     OP(jaws_closed,OFF)
  WAIT UNTIL IN(jaws_closed) = OFF
  'move back up
  position = bag_position
  GOSUB move_robot
RETURN
```

4. Variables

Local variables have been used through this program to make it more readable and so that it is easy to define input, outputs etc. They are defined in a separate program which is INCLUDEd in the main program. The example VARIABLE program is as follows:

```
'**************************************
' IN
'**************************************
  machine_enabled = 8
  bag_loaded = 9
  jaws_closed = 11
  r_axis_datum = 12
  z_axis_datum = 13
  base_axis_datum = 14
  wrist_axis_datum = 15
```

```
'***************************************
' OP
'***************************************
  jaws = 10 'ON= jaws closed, OFF = jaws open

'***************************************
' TABLE
'***************************************
  'Table 100+ is used for storing the positions
  table_start = 100
```

5. Full program

The full program can be seen below. Remember this is a sample and will need customisation to run on your robot. It is also important to remember that it does not have any error handling or reset conditions and so should be used as a sample when writing your full project.

```
  INCLUDE "VARIABLES"
  GOSUB initialise_robot
  GOSUB enable_robot
  GOSUB home_robot
  GOSUB load_positions

  bag_position=1 'first bag position on pallet
  bag_layer = 0 'start with the first layer
  bag_height = 100 'initial drop height for bag
  pick_height = 100 'height for picking the bags

  WHILE IN(machine_enabled) = ON
    GOSUB pick_bag
    IF bag_position = 1 OR bag_position = 7 THEN
      bag_layer = bag_layer + 1
    ENDIF
    place_height = bag_height * layer
    GOSUB place_bag
        bag_position = bag_position + 1
    IF bag_position = 13 THEN
      IF bag_layer = 5 THEN
        GOSUB reload_pallet
      ENDIF
      bag_position = 1
     ENDIF
  WEND

  STOP

'***************************************
pick_bag:
'***************************************
  'Move to pick position
  position = 0
  GOSUB move_robot
  z_position = pick_height - MPOS AXIS(z_axis)
  'move down to the pick height
  MOVE(0, z_position, 0, 0)
  WAIT IDLE
```

```
  'wait for a bag to pick
  WAIT UNTIL IN(bag_loaded)=ON 'wait for bag in pick position
  OP(jaws, ON) 'close jaws to pick up bag
  'wait for the sensor to detect the jaws are closed around the bag
  WAIT UNTIL IN(jaws_closed) = ON
  'move back up
  position = 0
  GOSUB move_robot
RETURN

'****************************************
place_bag:
'****************************************
  'Move to place position
  position = bag_position
  GOSUB move_robot
  BASE(r_axis, z_axis, base_axis, wrist_axis)
  z_position = place_height - MPOS AXIS(z_axis)
  'move down to the pick height
  MOVE(0, z_position, 0, 0)
  WAIT IDLE
  OP(jaws, OFF) 'open jaws to release the bag
  'wait for the sensor to detect that the jaws are open
  WAIT UNTIL IN(jaws_closed) = OFF
  'move back up
  position = bag_position
  GOSUB move_robot
RETURN

'****************************************
reload_pallet:
'****************************************
  PRINT#5, "Pallet full, press any key to continue"
  GET#5,char
RETURN

'****************************************
move_robot:
'****************************************
  'load positions from the table
  x_position = (TABLE(table_start + position * 4))
  y_position = (TABLE(table_start + position * 4 + 1))
  z_position = (TABLE(table_start + position * 4 + 2))
  wrist_angle_abs = (TABLE(table_start + position * 4 + 3))

  'Calculate the wrist and base angle
  'This is the scale and rotate robot transformation
  base_angle = (ATAN2(y_position, x_position)*360)/ (2* PI)
  wrist_angle = base_angle - wrist_angle_abs
  r_position = SQR(x_position * x_position + y_position * y_position)
  'Move the robot
  BASE(r_axis, z_axis, base_axis, wrist_axis)
  MOVEABS(r_position, z_position, base_angle, wrist_angle)
  WAIT IDLE
RETURN

'****************************************
initialise_robot:
'****************************************
```

```
  BASE(r_axis)
  counts_x_per_mm = 10000 'counts per mm of screw feed
  scale_x = 10.3 'scale from screw feed to end point movement
  UNITS = counts_x_per_mm / scale_x
  SPEED = 1000
  ACCEL = SPEED * 100
  DECEL = ACCEL

  BASE(z_axis)
  counts_z_per_mm = 10000 'counts per mm of screw feed
  scale_z = 10.3 'scale from screw feed to end point movement
  UNITS = counts_z_per_mm / scale_z

  BASE(base_axis)
  counts_base_per_rev = 10000 'counts per rev
  UNITS = counts_base_per_rev/360 'counts per degree
  'set the axis to work in +-180 degrees
  REP_DIST = 180
  REP_OPTION = 0

  BASE(wrist_axis)
  counts_wrist_per_rev = 10000 'counts per rev
  UNITS = counts_wrist_per_rev/360 'counts per degree
  'set the axis to work in +-180 degrees
  REP_DIST = 180
  REP_OPTION = 0
RETURN

'****************************************
home_robot:
'****************************************
  BASE(r_axis)
  DATUM_IN = r_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine
  WAIT IDLE
  FS_LIMIT = 1200
  RS_LIMIT = 0

  BASE(z_axis)
  DATUM_IN = z_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine
  WAIT IDLE
  FS_LIMIT = 750
  RS_LIMIT = 0

  BASE(base_axis)
  DATUM_IN = base_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine

  BASE(wrist_axis)
  DATUM_IN = wrist_axis_datum 'select input to use as datum switch
  DATUM(4) 'start datum routine
  WAIT IDLE
  DEFPOS(-170)
  FS_LIMIT = 170
  RS_LIMIT = -170

  BASE(base_axis)
  WAIT IDLE
```

```
  DEFPOS(-170)
  FS_LIMIT = 170
  RS_LIMIT = -170
RETURN

'***************************************
enable_robot:
'***************************************
  IF MOTION_ERROR THEN
    DATUM(0)
  ENDIF

  BASE(r_axis)
  SERVO = ON

  BASE(z_axis)
  SERVO = ON

  BASE(base_axis)
  SERVO = ON

  BASE(wrist_axis)
  SERVO = ON
  WDOG = ON
RETURN

'***************************************
load_positions:
'***************************************
  'TABLE(table_start * position, x_position , y_position, z_position,
wrist_angle_abs)

  'Position 0, pick position
  TABLE(table_start + 0 * 4, 600 , -600, 200, 45)

  'Position 1 - 6, 'horizontal layer of palletizing
  TABLE(table_start + 1 * 4, 300 , 150, 700, 0)
  TABLE(table_start + 2 * 4, 500 , 150, 700, 0)
  TABLE(table_start + 3 * 4, 700 , 150, 700, 0)
  TABLE(table_start + 4 * 4, 300 , 450, 700, 0)
  TABLE(table_start + 5 * 4, 500 , 450, 700, 0)
  TABLE(table_start + 6 * 4, 700 , 450, 700, 0)

  'Position 7 - 12, 'horizontal layer of palletizing
  TABLE(table_start + 7 * 4, 350 , 100, 700, 90)
  TABLE(table_start + 8 * 4, 650 , 100, 700, 90)
  TABLE(table_start + 9 * 4, 350 , 300, 700, 90)
  TABLE(table_start + 10 * 4, 650 , 300, 700, 90)
  TABLE(table_start + 11 * 4, 350 , 500, 700, 90)
  TABLE(table_start + 12 * 4, 650 , 500, 700, 90)
RETURN
```