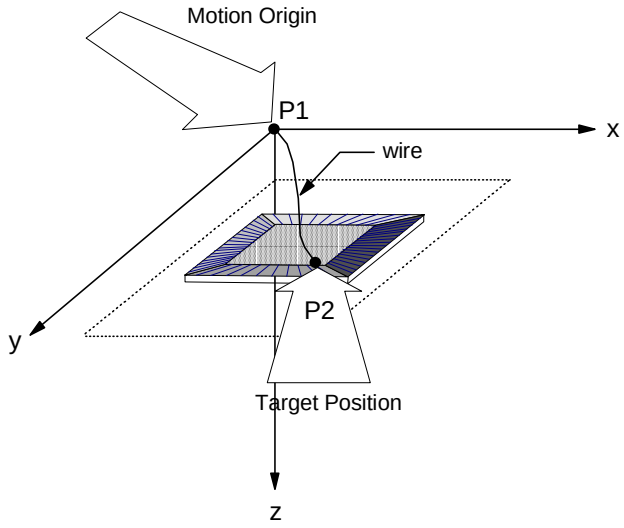
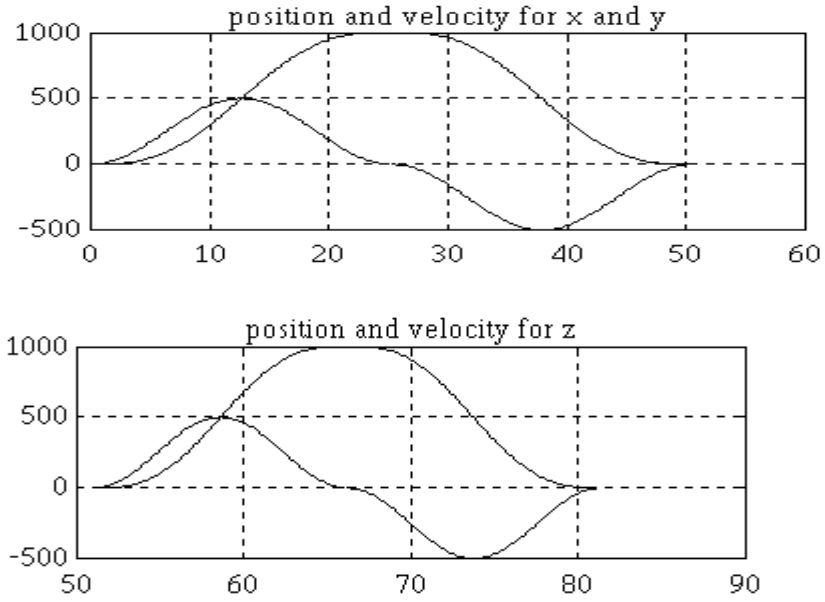
High Speed Moves with User Defined Trajectories

This application coordinates x, y, z (and w in a later example) axes to perform series of high speed (10-50 ms travel time) contouring moves. An example of such application is semiconductor wire bonding. We describe the DSPL programs that achieve the target points for x, y, and z along the user-defined trajectory. In the following examples the user defines a shape of the traveling trajectories such as the one illustrated below.



where P1 and P2 are characterized by their x, y, and z components. In this example, the user has defined the moves from P1 to P2 along a (1-cos(ωt)) velocity trajectory. The user has also specified that x and y complete their moves, simultaneously, in 50 ms. As you will see in the first DSPL application program listing (wirebond.hll), the motion trajectory period for both x-y and z are independently programmed. The DSPL routines *xy_traj.hll* and *z_traj.hll* generate the corresponding trajectories.

In the later example, the program automatically adjusts the move time to the length of target points.



In the first example, axes x and y reach their targets simultaneously. The z axis starts its move upon the completion of x and y motion. We've separated z trajectory from x and y to point out that z can have its own independent shape.

Supplying the Mx4 Target Positions for x, y and z

The end points for x, y and z trajectories can be downloaded in one of the following ways:

1) Host downloads the entire target points to the Mx4 memory using the download utility *Table|Points Data Table* in Windows 95/NT.

Since the DSPL allows internal computation, it is also possible for the Mx4 to obtain its own move parameters, on the fly and independent of the host.

2) Host provides the Mx4, the end points one set of x, y and z at a time.

The first DSPL program describes the first method. In this example, the data points for 16 pins of a semiconductor are downloaded. Each pin's x, y and z is characterized in a row as follows:

pin	x(count)	y(count)	z(count)
1	200	50	200
2	300	150	200
3	400	250	200
4	500	350	200
5	600	450	200
6	700	550	200
7	800	650	200
8	900	750	200
9	0	250	200
10	100	350	200
11	200	450	200
12	300	550	200
13	400	650	200
14	500	750	200
15	600	850	200
16	700	950	200

We start with creating a data file that contains the above end-points, saved in ASCII file "*points.dat*" (do not include the pin number in the data file).

Next, download the end-points to the Mx4 controller using the Mx4 table download utility as follows: click on *Tables*|*Points Data Table*|*File*. Select the file "points.dat". Next, select *Position Points* and *Offset* 800 in the Table Setup area of the box. Finally, click on the Download button to send the data contained in the file to the Mx4 board. The number that appears in the adjacent box is the number of points sent to the board.
The parameter "800" indicates at which starting index to begin downloading the data points in the Mx4 memory.

At this point, the Mx4 contains 16 rows of end-points.

Write a DSPL program to move the axes to target points along user defined trajectories

With the endpoints downloaded to the Mx4, we need to create a DSPL program which calculates the contouring data points and performs the cubic spline interpolation on the x, y, and z axes. The “wirebond.hll” DSPL program performs the above tasks on its own and independent of the host.

The “wirebond.hll” program uses the *#include* function to link in the “external” DSPL program files “xy_traj.hll” and “z_traj.hll”. These files generate the normalized data points, on the user defined trajectories. The “init.hll” DSPL file includes system initialization parameters such as control gains and maximum acceleration settings, etc.

The specific function of each of DSPL program file is contained in the commented documentation within the program listing itself.

```
;*************************************************************
;*
;*      Wire Bonding - A HIGH SPEED CONTOURING APPLICATION
;*
;*      This program performs very high speed (10-50 ms) contouring
;*      used primarily in IC bonding applications. The application
;*      uses x and y for table and z for vertical moves.
;*
;*      The external routines used in conjunction with this program
;*      are:
;*                "init.hll"       for initialization
;*                "xy_traj.hll"    for x y and
;*                "z_traj.hll"     z trajectory generations.
;*
;*      The target points for x,y and z are saved in
;*      data file "points.dat".  Before compiling this program
;*      "points.dat" must be down loaded to the Mx4 with an
;*      offset address.  For this program, we used 800 for offset
;*      address.
;*
;*************************************************************

#define  flag                var2
#define  period_xy           var3
#define  period_z            var57
#define  2pi                 var4
#define  aux4                var5
#define  aux5                var6
#define  aux6                var7
#define  aux1                var8
#define  aux2                var9
#define  index_cur_pos       var10
```

```
#define  index_inc_pos        var11
#define  scale                var13
#define  index_cur_posz       var14

#define  position             var20
#define  total_no_pts         var21
#define  x_target_pos         var22
#define  scaled_x             var23
#define  init_z_table         var26
#define  y_target_pos         var27
#define  z_target_pos         var28
#define  scaled_y             var29

#define  scaled_z             var30
#define  table_pointer        var33

#define  index_dec_pos        var42

#define  z_cur_pos            var50
#define  x_cur_pos            var51
#define  y_cur_pos            var52
#define  x_increment          var53
#define  y_increment          var54
#define  index_cur_posz       var59


#define  total_no_ptz         var60
#define  rate                 var61
#define  stay                 var62
#define  index_cur_posy       var63
#define  index_dec_posy       var64

#include "init.hll"
#include "z_traj.hll"
#include "xy_traj.hll"

PLC_PROGRAM:
    run_m_program(wire_bond)
END

;
;       Program wirebond.hll Performs A Stand Alone
;       3-Axis Contouring
;
wire_bond:

     rate = 5          ;rate is 1 ms
     call(INIT)        ;this routine is for gain initializations
     wait_until(var1 == 1);variable 1 is a flag which lets the main
                         ;program know it is done initializing

     period_xy = 50    ;period_xy holds x and y trajectory period
                       ; in ms
     period_z  = 30    ;period_z  holds z axis "stitching" period
                       ; in ms
     cubic_rate(rate)  ;cubic spline points are spaced by 1 ms

     period_xy = period_xy/2  ;this internal division by two is
                              ; necessary
```

```
    period_z =  period_z/20          ; because of the way
                                     ; trajectories are implemented

    total_no_pts = 2*period_xy + 2   ;total number of points for x
                                     ; and y
    total_no_ptz = 2*period_z + 2    ;total number of points for z

    scale = 1000                     ;scale holds the peak
                                     ; amplitude for position
    var25 = 2*total_no_pts           ;trajectory, var25 holds
                                     ; number of points for x and y
    var35 = total_no_pts
    var36 = total_no_pts-1

    call(xy_profile)                 ;this routine calculates the
                                     ; points on xy trajs
    wait_until(var2 == 1)

    call(z_profile)                  ;this routine calculates the
                                     ; points on z traj.
    wait_until(var2 == 1)

    init_z_table = 2*total_no_pts    ;holds the initial table point
                                     ; for z move
    stay = 2.5*total_no_pts          ;holds the delay to let z
                                     ; finish its move

;*****************************************************************
;*
;*        At this point program starts running all points
;*
;*****************************************************************

    table_pointer = 800       ;points to the initial table
                              ; location for target points.
    x_cur_pos = 0             ;initialize previously retrieved x
    y_cur_pos = 0             ;initialize previously retrieved y
    z_cur_pos = 0

    while(table_pointer < 848);start bonding 16 pins

       ;load target point for x
       x_target_pos = table_p(table_pointer)

       ;increment index variable table_pointer
       table_pointer = table_pointer+1

       ;load target point for y
       y_target_pos = table_p(table_pointer)

       ;increment index variable table_pointer
       table_pointer = table_pointer+1

       ;load target point for z
       z_target_pos = table_p(table_pointer)
```

```
        ;increment index variable table_pointer
        table_pointer = table_pointer+1

        ;when table finished loop over the table points
        if (table_pointer == 848)
               table_pointer = 800
        endif

        ;pos increment from the last x
        x_increment = x_target_pos - x_cur_pos
        ;pos increment from the last y
        y_increment = y_target_pos - y_cur_pos

         scaled_x = x_increment/scale  ;find scaling factor for x
         scaled_y = y_increment/scale  ;find scaling factor for y
         scaled_z = z_target_pos/scale ;find scaling factor for z

cubic_scale(0x7,scaled_x,x_cur_pos,scaled_y,y_cur_pos,scaled_z,0)

        ;run all x and y points
        cubic_int(total_no_pts,0,1, 0x3)

cubic_int(total_no_ptz,init_z_table,1, 0x4)   ;run z points

        x_cur_pos = x_target_pos      ;update x and y initial points
        y_cur_pos = y_target_pos

    wend

end

xy_profile:

;************************************************************
;*
;*      This routine calculates the points on xy trajectories
;*      and saves them in the table.
;*
;************************************************************

    flag = 0
    index_cur_pos = 0

    period_xy = 2*period_xy +2

    ;period_xy holds xy trajectory period in mS
    while (index_cur_pos <= period_xy)

        ;index into descending pos segment
        index_dec_pos = 2*period_xy - index_cur_pos

        2pi = 2*pi
        aux4 = 2pi/period_xy          ;calculates 2pi/T
        aux5 = 1/aux4                 ;calculates T/2pi
        aux6 = aux4*index_cur_pos     ;calculates 2pi*t/T

        ;calculates [(t - T/2pi*sin(2pi*t/T)]/T
        aux1 = (index_cur_pos – aux5*sin(aux6))/period_xy
```

```
            position = scale*aux1

            ;save  ascending X position
            table_p(index_cur_pos) = position

            ;save descending X position
            table_p(index_dec_pos) = position

            index_cur_posy = index_cur_pos + 1
            index_dec_posy = index_dec_pos + 1

            ;save ascending Y position
            table_p(index_cur_posy) = position

            ;save descending Y position
            table_p(index_dec_posy) = position

            index_cur_pos = index_cur_pos + 2
        wend

        flag = 1
        ret()

end
```

```
z_profile:

;************************************************************
;*
;*      This routine calculates the points on the z trajectory
;*      and saves them in the table.
;*
;************************************************************

   flag = 0

   ;start z data after the end of xy data
   index_cur_pos = 2*period_xy
   index_cur_pos = index_cur_pos + 2

   ;period_z holds the period
   period_z = period_z*2
   period_z = period_z + 2
   index_cur_posz = 0

   ;remember period_z is z period in mS
   while (index_cur_posz <= period_z)

          ;index into descending pos segment
          index_dec_pos = 2*period_z
          index_dec_pos = index_dec_pos + index_cur_pos
          index_dec_pos = index_dec_pos - index_cur_posz

          2pi = 2*pi
          aux4 = 2pi/period_z           ;calculates 2pi/T
          aux5 = 1/aux4                 ;calculates T/2pi
          aux6 = aux4*index_cur_posz    ;calculates 2pi*t/T
          aux1 = sin(aux6)

          aux1 = aux1*aux5       ;calculates (T/2pi)*sin(2*pi*t/T)
          aux1 = index_cur_posz-aux1

          ;calculates [t - T/2pi*sin(2pi*t/T)]/T
          aux1 = aux1/period_z

          position = scale*aux1

          index_inc_pos = index_cur_posz + index_cur_pos

          ;save  ascending position
          table_p(index_inc_pos) = position

          ;save descending position
          table_p(index_dec_pos) = position

           index_cur_posz = index_cur_posz + 1
      wend

    var12 = index_cur_posz
    flag = 1
    ret()

end
```

3-Axis Moves with Automatic Time / Length Computation

The differences between this example and the previous one are:

1) All moves reach their targets simultaneously
2) The equation for z is elliptical
3) The time to finish a move is a function of its length
4) Target points are passed (downloaded) to the Mx4 one set of (x, y, z) at a time

The host program that will download the target points to the DSPL program (one set at a time) is labeled as "*process.c*". We have included this C++ program in Appendix A of this chapter. To start this program you may use program "*target.exe*" which runs on Windows 95. This push button utility starts an endless transmission of data from the host to the Mx4 memories. You must remember that *process.c* program takes advantage of the Mx4's Visual Basic and C++ DLL. Therefore, to run this program you must have already installed the above DLL.

```
;************************************************************
;*
;*      This program performs time variable user defined
;*      trajectories for x, y and z:
;*
;*      1)  The host program sets end points for xyz and sets
;*          flag1=1 to signal dspl. The dspl calculates the time to
;*          finish the move and starts the move.
;*      2)  dspl clears flag1 to signal the host program it is ready
;*          to take new end points.
;*      3)  xy moves follow 1-cos(wt) for velocity and z moves are
;*          elliptic for z position as a function of
;*
;*              r = sqrt(x^2 + y^2).
;*
;*       The external routines used in conjunction with this program
;*       are:
;*                 "init.hll"  gain and position initialization
;*                 "xyz.hll"   generates norm trajectories for xyz
;*
;*      The target points for x,y,z as well as flag1 are at: var22,
;*      var27, var28, and var34 respectively.  The host program must
;*      first check flag1.  This flag must be zero before host can
;*      issue change_var.  Host needs to issue only one change_var
;*      command to change all above variables.
;*
```

```
;*****************************************************************

#define  flag2               var2
#define  period              var3
#define  2pi                 var4
#define  aux4                var5
#define  aux5                var6
#define  aux6                var7
#define  aux1                var8
#define  aux2                var9

#define  index_cur_pos       var10
#define  aux3                var11
#define  scale               var13
#define  position_z          var16
#define  last_z_pos          var18

#define  position            var20
#define  total_no_pts        var21
#define  x_target_pos        var22
#define  scaled_x            var23
#define  y_target_pos        var27
#define  z_target_pos        var28
#define  scaled_y            var29

#define  scaled_z            var30
#define  index_target_pos    var33
#define  flag1               var34
#define  xx                  var35
#define  yy                  var36
#define  zz                  var37

#define  index_dec_pos       var42

#define  z_cur_pos           var50
#define  x_cur_pos           var51
#define  y_cur_pos           var52
#define  x_increment         var53
#define  y_increment         var54
#define  z_increment         var55
#define  index_cur_pyz       var57
#define  index_dec_pyz       var58

#define  rate                var62
#define  max                 var63

#include "c:\mx4prov4\hll\init.hll"
#include "c:\mx4prov4\hll\xyz.hll"


PLC_PROGRAM:

     run_m_program(moves)

end
```

```
moves:
     ;this tells host it can not send move parameters yet
     flag1 = 1
     call(INIT)       ;this routine initializes gains

     ;variable 1 is a flag to show init is done
     wait_until(var1 == 1)

     period = 300            ;generates period for x,y and z

     ;routine to calculate the points on xyz trajectories
     call(xyz_profiler)

     wait_until(flag2 == 1)

    x_cur_pos = 0            ;initialize previously retrieved x
    y_cur_pos = 0            ;initialize previously retrieved y
    z_cur_pos = 0            ;initialize previously retrieved z

    x_target_pos = 0
    y_target_pos = 0
    z_target_pos = 0

    var1 = 1

    while(var1 == 1)                  ;start an endless loop

       x_cur_pos = cpos1
       y_cur_pos = cpos2
       z_cur_pos = cpos3

       ;x target point relative to current position
       x_increment = x_target_pos - x_cur_pos

       ;y target point relative to current position
       y_increment = y_target_pos - y_cur_pos

       ;z target point relative to current position
       z_increment = z_target_pos - z_cur_pos

       aux1 = x_target_pos
       aux2 = y_target_pos
       aux3 = z_target_pos

       ;scaled x target relative to current position
       scaled_x = x_increment/scale

       ;scaled y target relative to current position
       scaled_y = y_increment/scale

       ;scaled z target relative to current position
       scaled_z = z_increment/scale


       xx = abs(scaled_x)
       yy = abs(scaled_y)
       zz = abs(scaled_z)
```

```
        ;find the max length between target x, y and z
        if (xx >= yy)
                max=xx
        else
                max=yy
        endif
        if (zz >= max)
                max=zz
        endif

        ;make cubic spline rate proportional/max length
        rate = 10*max
        rate = int(rate)
        rate = rate + 5       ;minimum rate must be 5

        cubic_rate(rate)

        cubic_scale(0x7,scaled_x,x_cur_pos,scaled_y,y_cur_pos,
        scaled_z,z_cur_pos)

        ;this tells host it can change move parameters
        flag1 = 0

        ;run the previously entered moves
        cubic_int(total_no_pts,0,1,0x7)

        ;this has to be here to let cubic_int finish
        cubic_rate(5)

        axmove(0x7,1.9,aux1,100,1.9,aux2,100,1.9,aux3,100)

        wait_until(cpos1 == aux1)

        ;host sets flag1 = 1 and sets new target position with only
        ; one change_var
        wait_until(flag1 == 1)
    wend
end


xyz_profiler:

;************************************************************
;*
;*      This routine calculates the normalized
;*      points on xyz trajectories and saves them
;*      in the table.
;*
;************************************************************

     total_no_pts = 3*period

     ;total number of points for x,y and z
     total_no_pts = total_no_pts + 3

     scale = 810000   ;this is the max position in one move
     scale = scale/2  ;scale holds the peak amplitude for position
```

```
flag2 = 0
index_cur_pos = 0
last_z_pos = 0

period = period*3        ;three coordinates per point

;period holds xy trajectory periods in ms
while (index_cur_pos <= period)

   ;index into descending position
   index_dec_pos = 2*period
   index_dec_pos = index_dec_pos + 3
   index_dec_pos = index_dec_pos - index_cur_pos

   2pi = 2*pi
   aux4 = 2pi/period             ;calculates 2pi/T
   aux5 = 1/aux4                 ;calculates T/2pi
   aux6 = aux4*index_cur_pos     ;calculates 2pi*t/T

   aux4 = aux6/2pi
   aux4 = aux4*aux4
   aux4 = 1 - aux4               ;calculate 1 - (t/T)^2

   aux1 = sin(aux6)

   ;calculates (T/2pi)*sin(2*pi*t/T)
   aux1 = aux1*aux5

   aux1 = index_cur_pos - aux1

   ;calc. [(t-T/2pi*sin(2pi*t/T)]/T
   aux1 = aux1/period

   ;calc. sqrt(1 - (t/T)^2)
   aux4 = sqrt(aux4)
   aux4 = 1 - aux4

   position = scale*aux1
   position_z = scale*aux4

   ;save X position
   table_p(index_cur_pos) = position

   ;save for descending position
   table_p(index_dec_pos) = position

   index_cur_pyz = index_cur_pos + 1
   index_dec_pyz = index_dec_pos + 1

   ;save Y position
   table_p(index_cur_pyz) = position
   ;save for descending position
   table_p(index_dec_pyz) = position

   index_cur_pyz = index_cur_pyz + 1
   index_dec_pyz = index_dec_pyz + 1

   ;save Z position
```

```
    table_p(index_cur_pyz) = position_z
    ;save for descending position
    table_p(index_dec_pyz) = position_z

    index_cur_pos = index_cur_pos + 3
 wend

flag2 = 1
ret()

end
```

4-Axis Moves with Automatic Time/Length Computation

This example is similar to the previous one except the program is written for four axes.

The host program that downloads the target points to the DSPL program (one set at a time) is labeled as "*process.c*". We have included this C++ program in Appendix A of this chapter. To start this program you may use program "*target.exe*" which runs on Windows 95. This push button utility starts an endless transmission of data from the host to the Mx4 memories. You must remember that *process.c* program takes advantage of the Mx4's Visual Basic and C++ DLL. Therefore, to run this program you must have already installed the above DLL.

```
;*****************************************************************
;*
;*      This program performs user defined trajectory for
;*      x,y,z and w:
;*
;*      user set end points and flag1 to signal dspl
;*      dspl decides about the time to finish a move
;*
;*      The external routines used in conjunction with this program
;*      are:
;*              "init.hll"       gain and position initialization
;*              "xyzw.hll"       generates norm trajectories for xyz
;*
;*      The target points for x, y and z are at: var22, var27, and
;*      var28
;*      flag1 is at var34.  The host C programs can only issue a
;*      change_var when var34 = 0.  When var34 is 0, one change_var
;*      can change target points for x,y and z as well as flag1 =
;*      var34 to 1.
;*
;*****************************************************************

#define  flag2               var2
#define  period              var3
#define  2pi                 var4
#define  aux4                var5
#define  aux5                var6
#define  aux6                var7
#define  aux1                var8
#define  aux2                var9
#define  index_cur_pos       var10
#define  aux3                var11

#define  scale               var13
#define  w_cur_pos           var14
#define  w_target_pos        var15
```

```
#define  w_increment          var16
#define  scaled_w             var17
#define  ww                   var18
#define  coded_pve_vel        var19

#define  position             var20
#define  total_no_pts         var21
#define  x_target_pos         var22
#define  scaled_x             var23
#define  y_target_pos         var27
#define  z_target_pos         var28
#define  scaled_y             var29

#define  scaled_z             var30
#define  index_target_pos     var33
#define  flag1                var34
#define  xx                   var35
#define  yy                   var36
#define  zz                   var37
#define  aux0                 var38

#define  index_dec_pos        var42


#define  z_cur_pos            var50
#define  x_cur_pos            var51
#define  y_cur_pos            var52
#define  x_increment          var53
#define  y_increment          var54
#define  z_increment          var55

#define  index_cur_pyz        var57
#define  index_dec_pyz        var58

#define  rate                 var62
#define  max                  var63

#include "init_mx4.hll"
#include "c:\mx4prov4\hll\xyzw.hll"

PLC_PROGRAM:
     run_m_program(moves)

end


moves:
     flag1 = 1         ;this tells the host not to send move
                       ;parameters yet
     call(INIT_MX4)    ;this routine is for gain initializations
     wait_until(var1 == 1)    ;variable 1 is a flag which lets the
                       ;main program know it is done initializing

     period = 50       ;period holds minimum move time

     call(xyzw_profiler)      ;this routine calculates the points
                              ;on xyzy trajectory target points.
     wait_until(flag2 == 1)
```

```
x_cur_pos = 0            ;initialize previously retrieved x
y_cur_pos = 0            ;initialize previously retrieved y
z_cur_pos = 0            ;initialize previously retrieved z
w_cur_pos = 0

x_target_pos = 0
y_target_pos = 0
z_target_pos = 0
w_target_pos = 0

var1 = 1

while(var1 == 1)      ;

   x_cur_pos = cpos1
   y_cur_pos = cpos2
   z_cur_pos = cpos3
   w_cur_pos = cpos4

   ;x target point relative to current position
   x_increment = x_target_pos - x_cur_pos

   ;y target point relative to current position
   y_increment = y_target_pos - y_cur_pos

   ;z target point relative to current position
   z_increment = z_target_pos - z_cur_pos

   ;w target point relative to current position
   w_increment = w_target_pos - w_cur_pos

   aux1 = x_target_pos
   aux2 = y_target_pos
   aux3 = z_target_pos
   aux0 = w_target_pos

   ;scaled x target relative to current position
   scaled_x = x_increment/scale

   ;scaled y target relative to current position
   scaled_y = y_increment/scale

   ;scaled z target relative to current position
   scaled_z = z_increment/scale

   ;scaled w target relative to current position
   scaled_w = w_increment/scale

   xx = abs(scaled_x)
   yy = abs(scaled_y)
   zz = abs(scaled_z)
   ww = abs(scaled_w)

   if (xx >= yy)         ;find the max between x,y,z and w
         max=xx
   else
         max=yy
   endif
```

```
        if (zz >= max)
               max=zz
        endif

        if (ww >= max)
               max=ww
        endif

        rate = 5*max
        rate = int(rate)
        rate = rate + 5

        cubic_rate(rate)
        cubic_scale(0xf,scaled_x,x_cur_pos,
                        scaled_y,y_cur_pos,
                        scaled_z,z_cur_pos,
                        scaled_w,w_cur_pos)

        flag1 = 0

        ;run all x and y points
        cubic_int(total_no_pts,0,1, 0xF)
        cubic_rate(5)

        wait_until(flag1 == 1)
     wend

end


xyzw_profiler:
;*************************************************************
;*
;*     This routine calculates the normalized points on xyzw
;*     trajectories and saves them in the table.
;*
;*************************************************************

    total_no_pts = 4*period

    ;total number of points for x,y,z and w
    total_no_pts = total_no_pts + 4

    scale = 100010    ;this is the max position in one move
    scale = scale/2   ;scale holds the peak amplitude for
                      ; position

    flag2 = 0
    index_cur_pos = 0
    period = period*4

    ;period = xyzw trajectory periods in mS
    while (index_cur_pos <= period)

        ;index into descending pos seg
        index_dec_pos = 2*period
        index_dec_pos = index_dec_pos+8
        index_dec_pos = index_dec_pos - index_cur_pos
```

```
        2pi = 2*pi
        aux4 = 2pi/period             ;calculates 2pi/T
        aux5 = 1/aux4                 ;calculates T/2pi
        aux6 = aux4*index_cur_pos     ;calculates 2pi*t/T

        ;calculates (T/2pi)*sin(2*pi*t/T)
        aux1 = sin(aux6)
        aux1 = aux1*aux5

        ;calc. [(t - T/2pi*sin(2pi*t/T)]/T
        aux1 = index_cur_pos - aux1
        aux1 = aux1/period

        position = scale*aux1

        aux3 = index_cur_pos
        ;save position for X
        table_p(index_cur_pos) = position

        ;save for descending position
        table_p(index_dec_pos) = position

        index_cur_pyz = index_cur_pos + 1
        index_dec_pyz = index_dec_pos + 1

        ;save position for Y
        table_p(index_cur_pyz) = position
        table_p(index_dec_pyz) = position

        index_cur_pyz = index_cur_pyz + 1
        index_dec_pyz = index_dec_pyz + 1

        ;save position for Z
        table_p(index_cur_pyz) = position
        table_p(index_dec_pyz) = position

        index_cur_pyz = index_cur_pyz + 1
        index_dec_pyz = index_dec_pyz + 1

        ;save position for W
        table_p(index_cur_pyz) = position
        table_p(index_dec_pyz) = position

        index_cur_pos = index_cur_pos + 4

        wend

      flag2 = 1
      ret()

    end
```