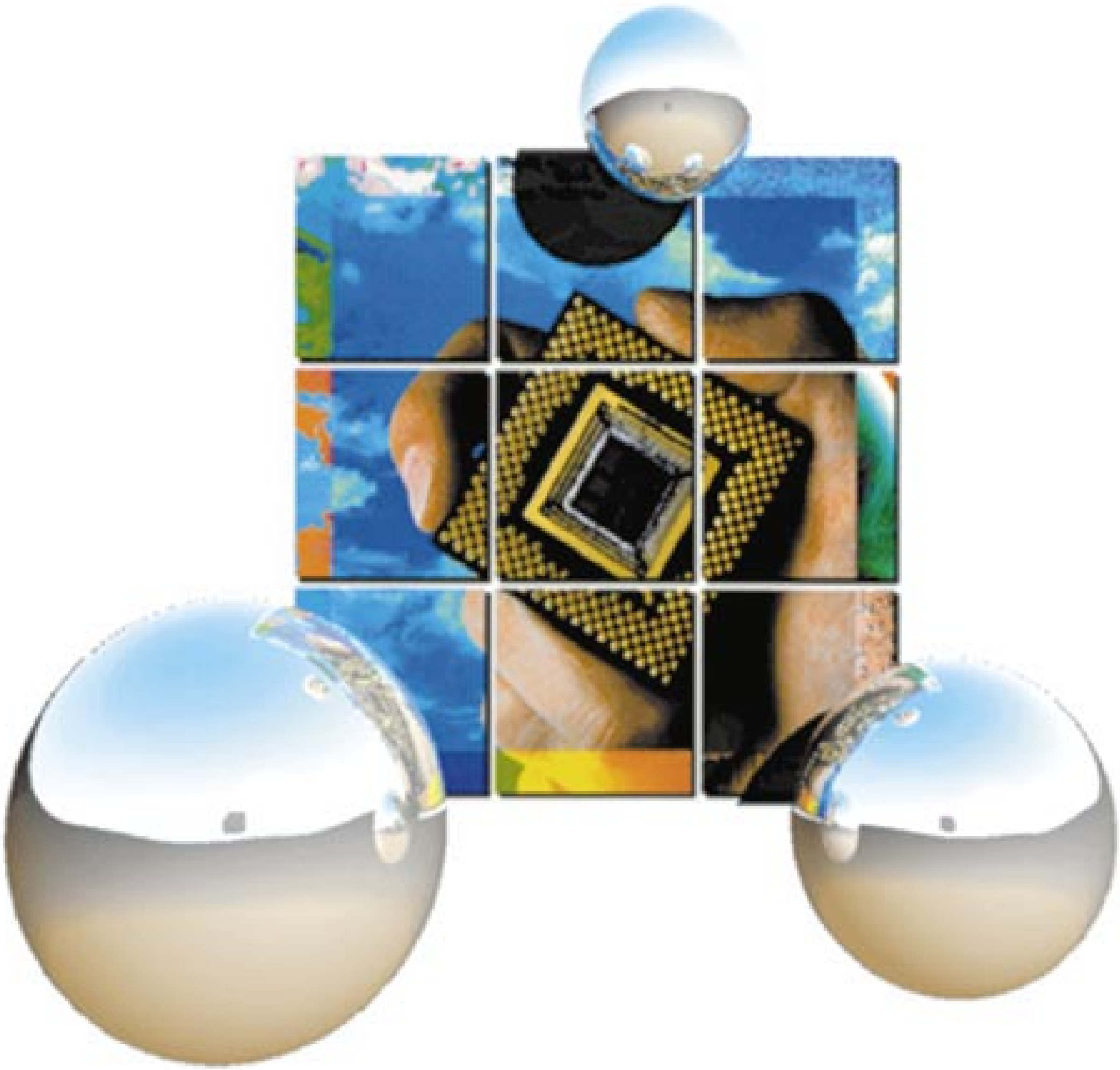
DSPL

Application Programs v3.1



DSPL

Application Programs

v3.1

DSP Control Group, Inc.
PO Box 39331
Minneapolis, MN 55439
Phone: (612) 831-9556
FAX: (612) 831-4697

Contents

This page intentionally blank.

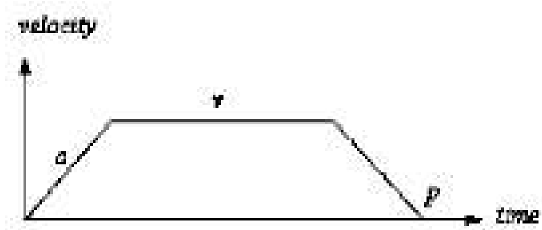
1 Motion Pallet

Point-to-Point Move Family

These commands facilitate point to point moves. Their function is simple: given the current and target positions, find a trapezoid or an s-curve path velocity to achieve the target. All commands in this family complete the motion (i.e. they bring the system to a complete stop.)

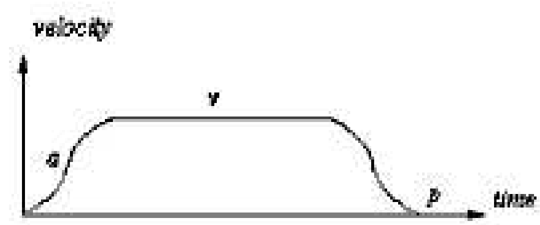
axmove

A *trapezoidal* move which uses traveling speed, acceleration and end point for its arguments. In a trapezoidal move the acceleration to reach slew speed is constant. Also, the time to achieve the target position is a function of this move's arguments.



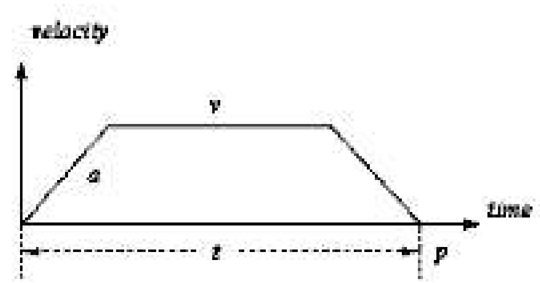
axmove_s

The arguments for this move are similar to those for AXMOVE - except, this command produces *S curve* velocity. Due to its finite jerk (derivative of acceleration with respect to time) compared to AXMOVE this command is gentler to the mechanical structure. You may achieve a better result with AXMOVE_S, when high acceleration AXMOVE results an unacceptable overshoot. Also, it must be noted that compared to AXMOVE, AXMOVE_S takes the same amount of time to finish the move.



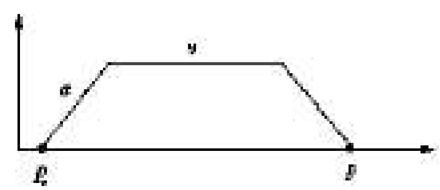
axmove_t

This command generates a trajectory similar in shape to AXMOVE except its arguments are, end position and *time* to finish a move. The instruction will automatically generate the trapezoidal profile to finish the move in a programmed time.



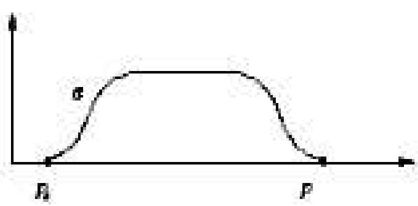
rel_axmove

An instruction similar to AXMOVE, only its target point is *relative* to the current position.



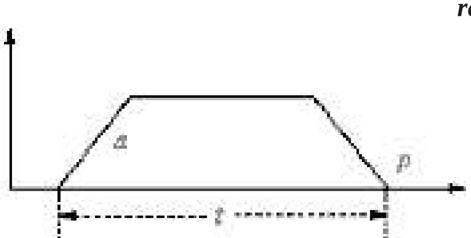
l_axmove_s

An instruction similar to AXMOVE_S, only its target point is *relative* to the current position.



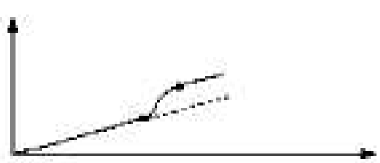
rel_axmove_t

An instruction similar to AXMOVE_T, only its target point is *relative* to the current position.



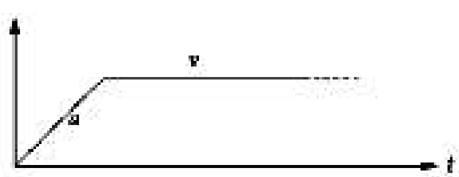
rel_axmove_slave

This instruction produces a superimposed move on the slave axes in a master slave application. Therefore, a slave will move a programmed REL_AXMOVE on the top of its slave motion.



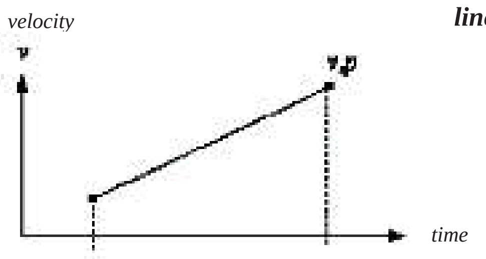
velmode

Velocity mode instruction regulates speed of a machine. It follows a trapezoidal profile to reach the slew speed.



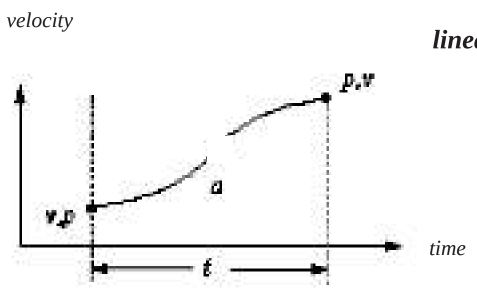
Linear Move Family

These commands facilitate segment moves. Their function is simple: given the current position and speed, they achieve a programmed target position/velocity by moving over a linear (or s-curve) velocity path. All commands in this category perform a *segment* motion (i.e. depending on target speed they may or may not bring the system to a complete stop).



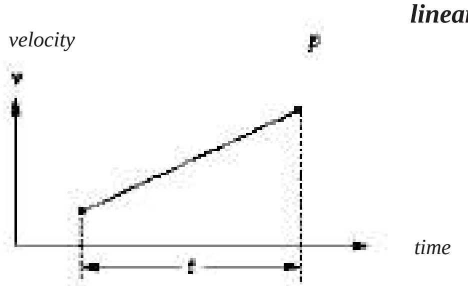
linear_move

This command brings a system form its current position and speed to a target position and speed over a linear speed trajectory. Its arguments are target position and velocity.



linear_move_s

This command brings a system form its current position and speed to a target position and velocity over an S-curve speed trajectory. Its arguments are current position/velocity, target position/velocity, acceleration and move time.



linear_move_t

This command is similar to LINEAR_MOVE except its arguments are final position and time to reach the final position. The command will gen erate a linear speed (within the maxi mum programmed accel eration) to achieve the position within the pro grammed time period.

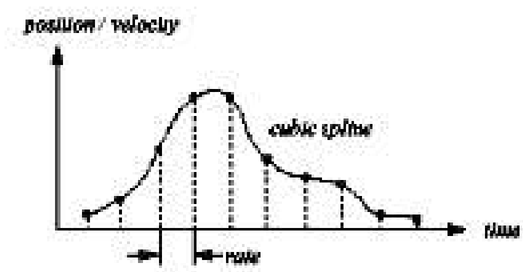
Cubic Spline Move

cubic_int

Using Cubic Spline interpolation, the user points specified by their positions and time interval (rate) can be connect ed together. The CUBIC_INT instruc tion arguments are:

1) number of move parameters to run;
2) initial index to move parameter table and;
3) number of times to loop through the selected points.

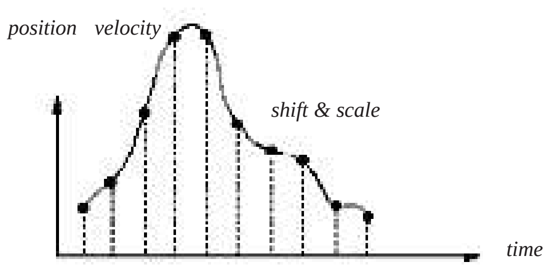
/



cubic_scale

This command performs a combination of shift and scale on the move parame ters. Shift relocates the origin of Cartesian coordinate and scale attenu ates or magnifies the moves.

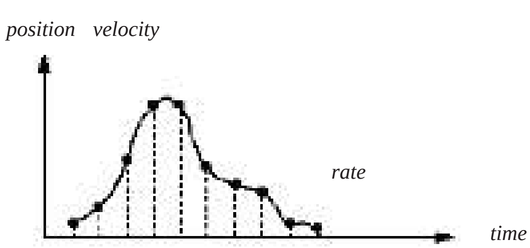
/



cubic_rate

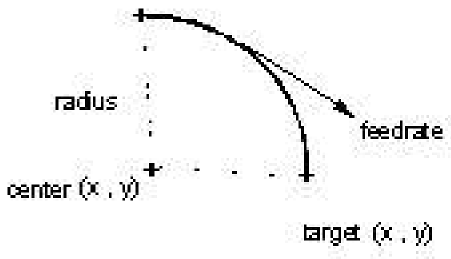
CUBIC_RATE determines the time interval between two adjacent points. This instruction's argument, time, can be changed on the fly.

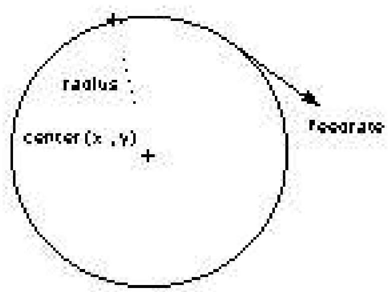
/



Arc and Circular Moves

These commands facilitate arcs, full circle or a circle *with compensation* for backlash or other non-linearity.





arc

The arc command performs an arc using the following arguments.

center (x,y)
radius,
feedrate,
target (x,y) relative to current position

circle

The circle command performs a circle using the following arguments:

center (x,y)
radius
feedrate

sine_on

This instruction turns on the sinusoidal interpolation involved in a circular motion.

sine_off

This instruction turns off the sinusoidal interpolation involved in a circular motion.

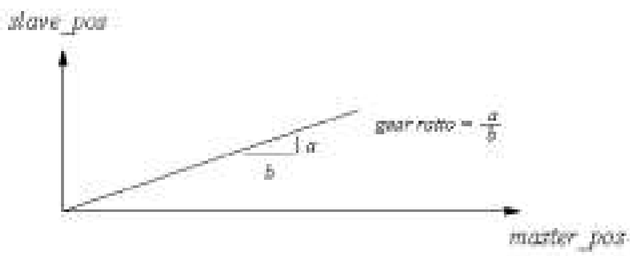
table_on

Turns on a normalized and user programmable path compensation table that will be added to the sinusoidal table used in a circular interpolation.

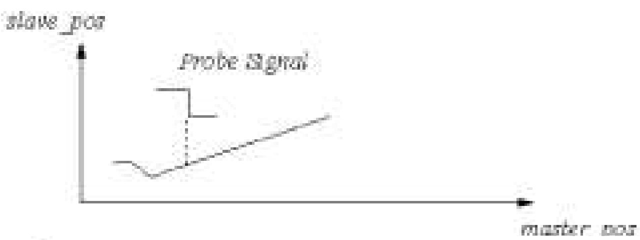
table_off

This instruction is opposite of TABLE_ON.

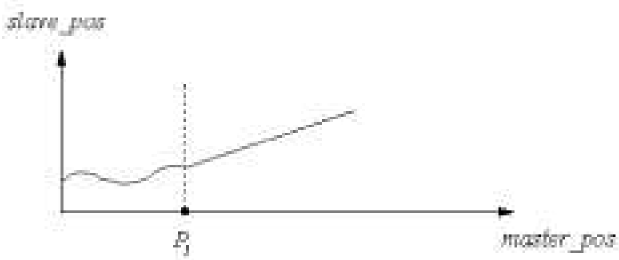
Master Slave Command Family



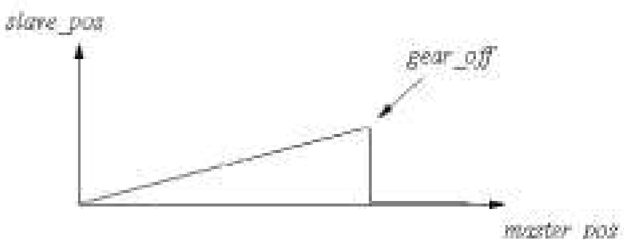
gear This command puts a selected number of axes (slaves) in ratioing relationships with master.



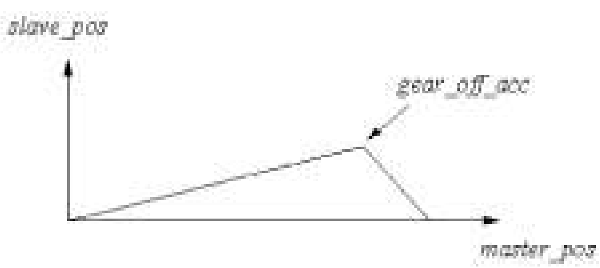
gear_probe This instruction is similar to GEAR except gear is engaged at active transi tion of an external signal (referred to as probe) from high to low.



gear_pos This instruction is similar to GEAR except gear is engaged when position is passed a programmed value, p_1.



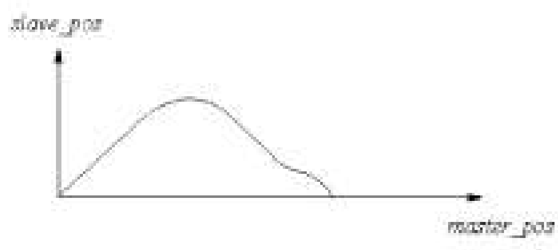
gear_off This command disengages master and slave instantly.



gear_off_acc This command disengages master and slave instantly. Slave stops at the deceleration rate programmed by MAX ACC.

cam

This instruction uses the selected master, selected slaves and the table number into which cam gear ratios have been already downloaded.

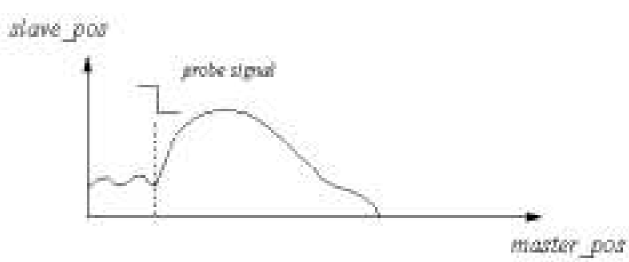


cam_tsize

This instruction uses the table number and size of cam table as its arguments.

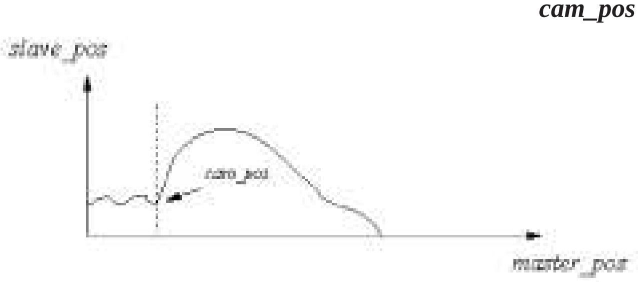
cam_probe

This instruction is similar to CAM except CAM is engaged at active transition of external signal (probe) from high to low.



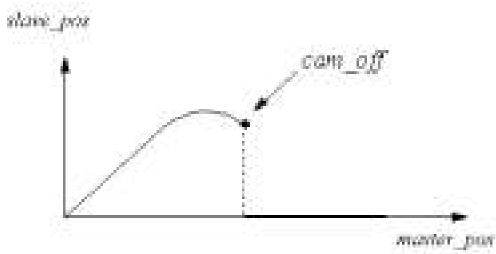
cam_pos

This instruction is similar to CAM except CAM is engaged when position is passed a programmed value.



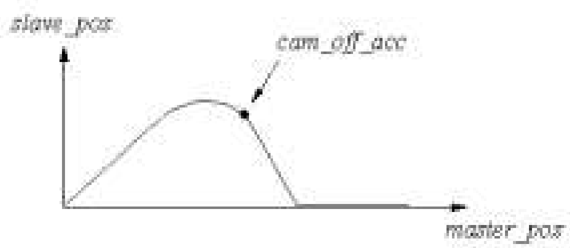
cam_off

This command disengages master and slave instantly.



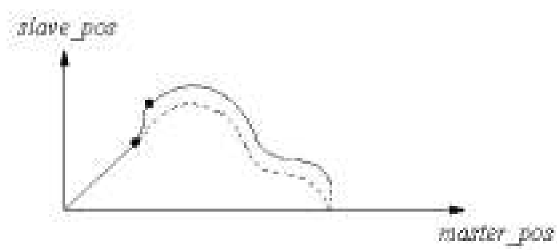
cam_off_acc

This command disengages master and slaveinstantly. Slave stops at the deceleration rate programmed by MAXACC.



rel_axmove_slave

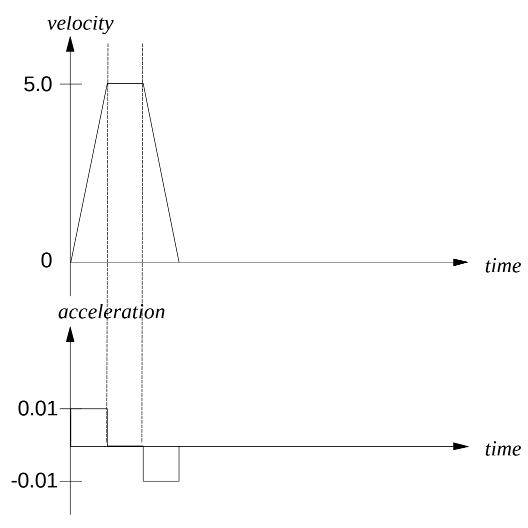
This instruction produces a superimposed move on the slave axes in a master slave application. Therefore, a slave will move a programmed position move in addition to its slave move.



2 Simple Point-to-Point Moves

Simple Trapezoidal Move

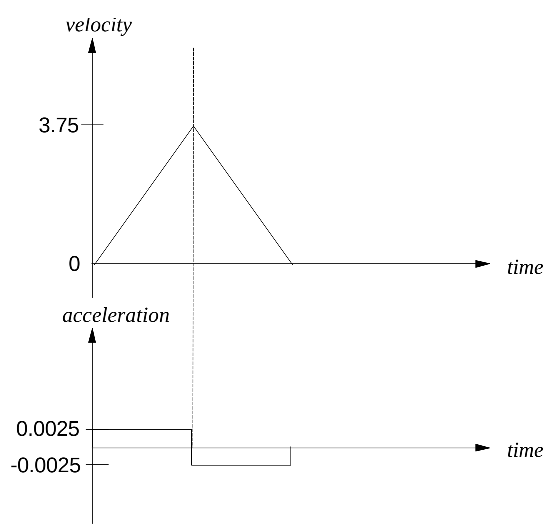
This simple motion program moves motor one from a preset position to a new position with a specified velocity profile characterized by its slew rate and acceleration.




```
trapezoid:

    pos_preset(0x1,0)
    axmove(0x1,0.010,5000,5.0)

end
```

Simple Triangular Move

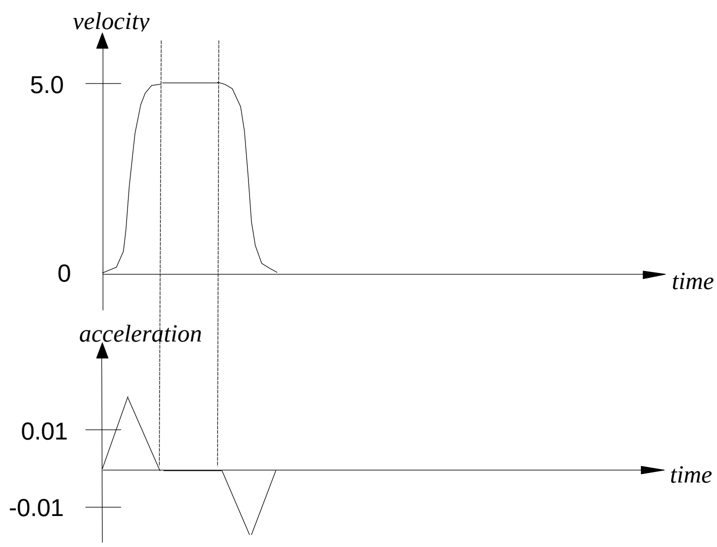
This program moves motor one from an initial position of 0 to a final position of 5,625 counts on a triangular velocity profile. This profile uses an acceleration of 0.0025 counts/(200 µs)^2 and target velocity of 5.0 counts/200µs.




```
triangle:

    pos_preset(0x1,0)
    axmove(0x1,0.0025,5625,5.0)

end
```

S-Curve Trapezoidal Move

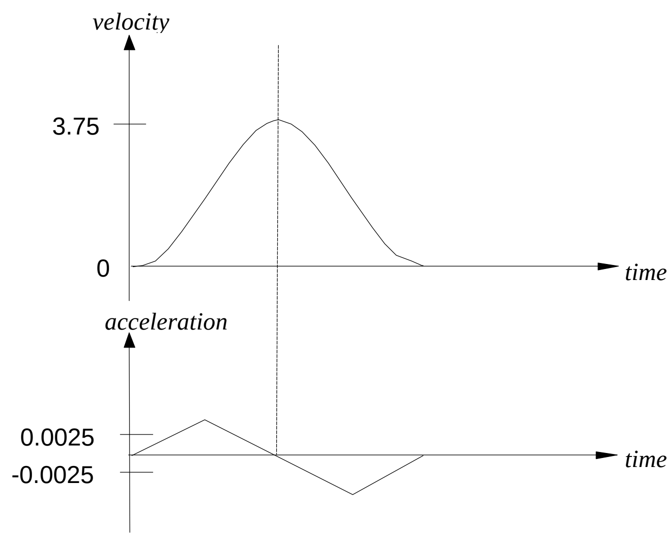
This simple motion program moves motor one from a preset position to a new position with a specified s curve velocity profile characterized by its slew rate and acceleration. Note that the maximum acceleration achieved in the move will be twice that programmed as the acceleration argument.




```
scurve_trapezoid:

    pos_preset(0x1,0)
    axmove_s(0x1,0.010,5000,5.0)

end
```

S-Curve Triangular Move

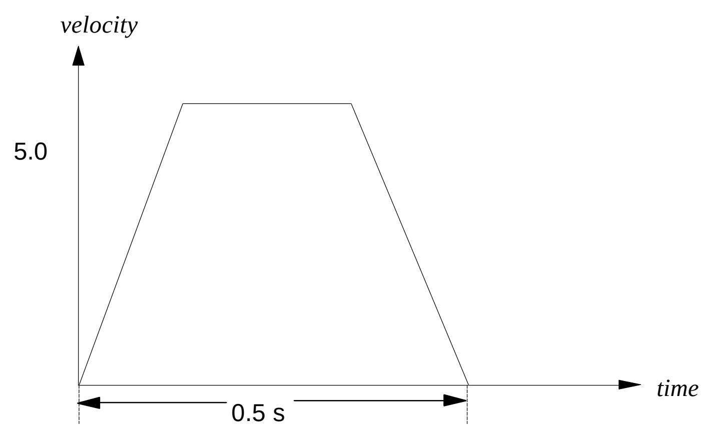
This program moves motor one from an initial position of 0 to a final position of 5,625 counts on a triangular s curve velocity profile. This profile uses an acceleration of 0.0025 counts/(200 µs)^2 and target velocity of 5.0 counts/200µs.




```
scurve_triangle:

    pos_preset(0x1,0)
    axmove_s(0x1,0.0025,5625,5.0)

end
```

Time Based Trapezoidal Move

This program moves motor one from an initial position of 0 to a final position of 20000 counts in 500 ms (or 2500*200µs) at acceleration = 1 count/(200 µs)^2. Velocity for this move will be automatically calculated by the Mx4.

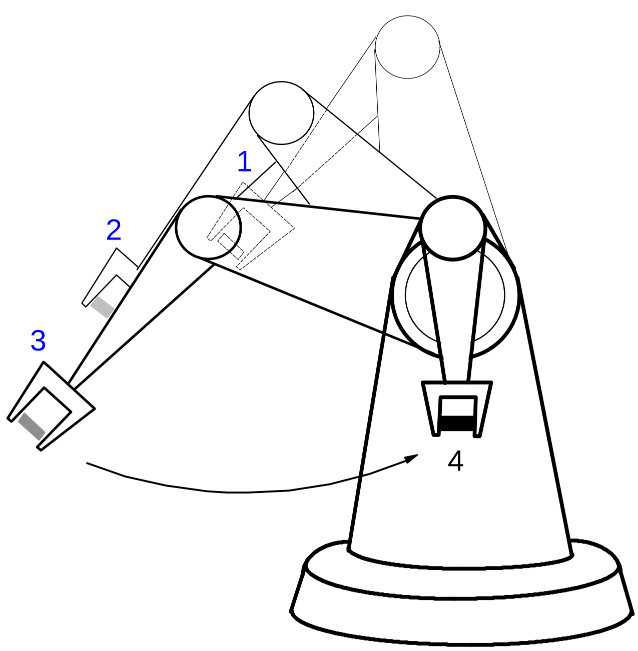



```
time_trapezoid:

    pos_preset(0x1,0)
    axmove_t (0x1,1,20000,2500)

end
```

This page intentionally blank.

3 Time Based Motion Programs

In the following application a series of moves for multiple joints are to be completed within the specified times: `t1,t2,...` respectively. This means that all motors must reach their intermediate target positions (`posx,posy,posz and posw`) simultaneously. The DSPL instruction `AXMOVE_T` is ideal for this application. It is important to note that a real time execution of `AXMOVE_T(or AXMOVE)`with its new move parameter(s) will intercept the one in progress. There are two ways to supply a DSPL program with target positions (and/or other move parameters). The first method allows the host to update move parameters using real time command `CHANGE_VAR`. This command is provided with the Mx4 C++ /Visual Basic 32-bit DLL. In the second method the DSPL retrieves the move parameters from its own table memory. Alternatively, the DSPL can use its own floating point math for real time computation of move parameters.



1) Host updates the target positions to reach in a specified time

In this case host updates the target points. The communication protocol between DSPL and host programs is as follows. First, the DSPL resets `flag = 0` to let host program know it can update target points. Host uses command `CHANGE_VAR` to update the target points. Upon the completion of variable update, host sets the `flag = 1` to let DSPL program know update is finished. The DSPL uses the recently updated variables as arguments for `AXMOVE_T` command and resets the `flag = 0` to let the host program know that once again host is allowed to update target points.

```
#define    accx   var2
#define    posx   var3
#define    t      var4
#define    accy   var5
#define    posy   var6
#define    accz   var7
#define    posz   var8
#define    accw   var9
#define    posw   var10
#define    flag   var11

#include “init_mx4.hll”

plc_program

    run_m_program(move_arm)

plc_end

move_arm:
    call(init_mx4)            ;initialize the gains
    t = 200                   ;set time to 200*200µsec = 40 ms
    flag = 0                  ;tell the host it can update motion parameters
    wait_until(flag == 1)     ;wait until host finished updating parameters

    while (var1 == 1)
        axmove_t(0xf, accx, posx, t, accy, posy, t, accz, posz, t, accw, posw, t)
        flag = 0           ;tell host it can change move parameters
        wait_until(cpos 1 == posx);wait until move is finished
        wait_until(flag == 1)     ;host sets flag upon updating motion parameters
    wend
end
```

2) DSPL calculates / retrieves the target positions to reach in a specified time

In this case, the target points are retrieved from the Mx4 table memory. The subroutine, `get_points` performs this data retrieval. The variable `size` holds the number of prestored target points. To download target position to the Mx4 table memory, you may use the `download position` facility provided with Mx4pro v4.

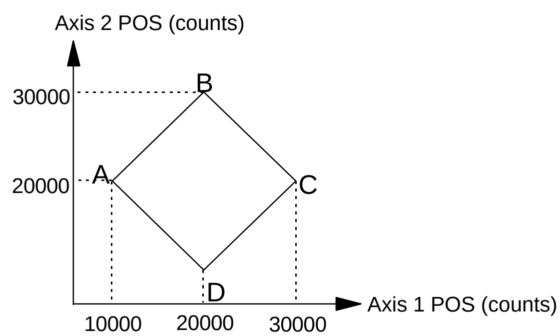
```
#define        size    var1
#define        accx    var2
#define        posx    var3
#define           t    var4
#define        accy    var5
#define        posy    var6
#define        accz    var7
#define        posz    var8
#define        accw    var9
#define        posw    var10
#define        flag    var11

#include “mx4_init.hll”

plc_program
    run_m_program(move_arm)

plc_end

move_arm:
t = 200                          ;set time to 200*200µsec = 40 ms
accx = 1
accy = 1
accz = 1
accw = 1

size = 500                       ;the total number of moves

call(get_points)
while (size >= 1)
    axmove_t(0xf, accx, posx, t, accy, posy, t, accz, posz, t, accw, posw, t)
    targetx = posx
    call(get_points)
    wait_until(cpos 1 == targetx) ;wait until move is finished
    var1 = var1 - 1
wend
```

```
get_points:
                  posx = table_p(index)  ;retrieve one set of 32-bit target points
                  index = index + 2
                  posy = table_p(index)
                  index = index + 2
                  posz = table_p(index)
                  index = index + 2
                  posw = table_p(index)

            ret()
            end
```

4 Linear & Circular Moves

Constant Acceleration Linear Move

The linear motion commands are used in motions where the velocity connecting point A to point B is linear. The starting position/velocity (defining point A) are those of an axis at the commencement of this command. The ending position and velocity are the command's arguments. The following example will trace a square shape as illustrated below.

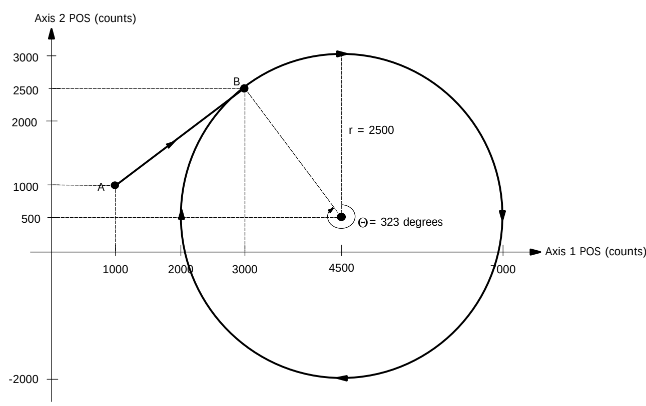



```
plc_program:

        run_m_program (square)

end_program

square:
        var23=1
        ctrl(0x3,0,1000,1000,1000,0,1000,1000,1000)
                              ;set control gains for motor 1
        pos_preset(0x3,10000,20000)                  ;point A
        while(var23==1)

            linear_move(0x3,15000,5,25000,5)         ;point AB/2
            linear_move(0x3,20000,0,30000,0)         ;point B

            linear_move(0x3,25000,5,25000,-5)        ;point BC/2
            linear_move(0x3,30000,0,20000,0)         ;point C

            linear_move(0x3,25000,-5,15000,-5)       ;point CD/2
```

```
            linear_move(0x3,20000,0,10000,0)          ;point D

            linear_move(0x3,15000,-5,15000,5)         ;point DA/2
            linear_move(0x3,10000,0,20000,0)          ;point A

        wend
end
```

Combined S-Curve Linear & Circular Moves

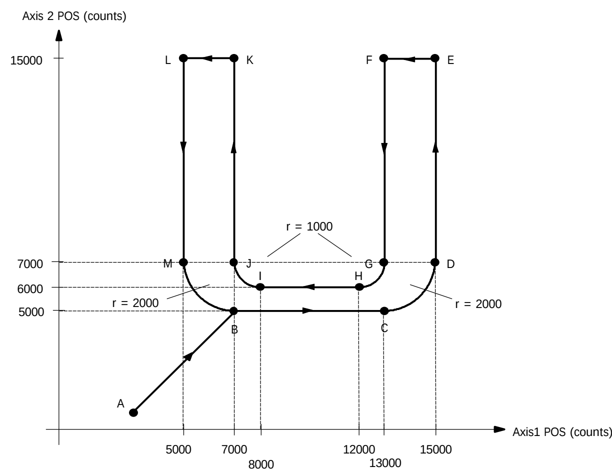
From position A (1000,1000) counts, move axes one and two to position (3000,2500) where the axes complete 360° of a circle centered at (4500,500). The circle feedrate is 1.0 counts/200µs.




```
circular:

        pos_preset(0x3,1000,1000)     ;preset position counters to
                                      ;point A
        linear_move_s(0x3,1000,0,3000,0.8,5000,0.00030,1000,0,2500,0.6,500
               0,0.00023)
                                      ;linear move from A to B
        circle(0x3,1500,-2000,2500,1.0,0,0)
                                      ;circle from B to B (360
                                      ;degrees clockwise)

end
```

Combined Linear & Arc Moves

This example demonstrates how to move an x-y table according to the shape illustrated below.



U-Shape:

```
pos_preset(0x3,3000,1000)            ;preset position counters
axmove(0x3,0.004,7000,1.0,0.004,5000,1.0)
                                     ;from A to B
wait_until(CPOS1==7000)              ;wait for completion of A
                                     ;to B motion

linear_move_s(0x1,7000,0,13000,1.0,0,0)
                                     ;from B to C
circle(0x3,0,2000,2000,-1.0,2000,2000)
                                     ;from C to D
linear_move_s(0x2,7000,1.0,15000,0,0,0)
                                     ;from D to E
wait_until((CPOS2==15000) and (CVEL2==0))
                                     ;wait for completion of D
                                     ;to E motion

axmove(0x1,0.004,13000,-1.0)         ;from E to F
wait_until((CPOS1==13000) and (CVEL1==0))
                                     ;wait for completion of E
                                     ;to F motion

linear_move_s(0x2,15000,0,7000,-0.4,0,0)
                                     ;from F to G
```

```
        circle(0x3,-1000,0,1000,0.4,-1000,-1000)
                                              ;from G to H
        linear_move_s(0x1,12000,-0.4,8000,-0.4,0,0)
                                              ;from H to I
        circle(0x3,0,1000,1000,0.4,-1000,1000)
                                              ;from I to J

        linear_move_s(0x2,7000,0.4,15000,0,0,0)
                                              ;from J to K
        wait_until((CPOS2==15000) and (CVEL2==0))
                                              ;wait for completion of J
                                              ;to K motion

        axmove(0x1,0.004,5000,-1.0)           ;from K to L
        wait_until((CPOS1==5000) and (CVEL1==0))
                                              ;wait for completion of K
                                              ;to L motion
        linear_move_s(0x2,15000,0,7000,-1.0,0,0)
                                              ;from L to M
        circle(0x3,2000,0,2000,-1.0,2000,-2000)
                                              ;from M to N

end
```

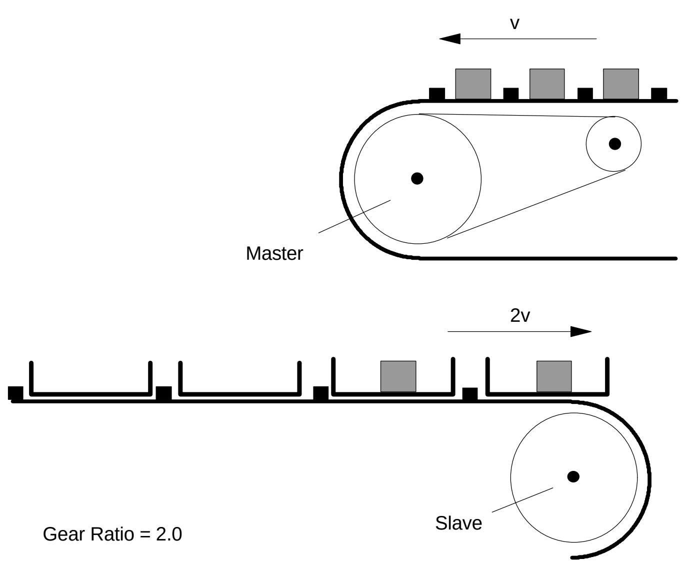
This page intentionally blank.

5 Electronic Gearing Programs

The four applications that will be covered in this section include:

1) Single gear ratio motion program
2) Variable gear ratio motion program
3) Engage in electronic gearing when external signal changes state
4) Engage in electronic gearing when master passes a programmed position

Illustrated below is an example of a packaging process that includes two conveyor belts. The upper belt contains the products equally positioned in between the logs. The master motor moves the product and drops each into the buckets. Clearly, this calls for a gearing mechanism that engages the master and slave, the conveyor belt moving the buckets. The gear ratio in this example is determined by the ratio of the space between the centers of adjacent buckets and the space between the products. In the following example, the motion program runs only one master/slave line. This line states master is motor 1, slave is motor 2 and gear ratio is 2.



1) Single gear ratio motion program

```
#define master      var2
#define slave       var3

#include    "init_mx4.hll"

plc_program
    run_m_program(electronic_gearing)
end_plc


master = 1            ;select axis 1 as master
slave  = 2            ;select axis 2 as slave

electronic_gearing:
           gear(master, slave, 2)
end
```

2) Variable gear ratio motion program

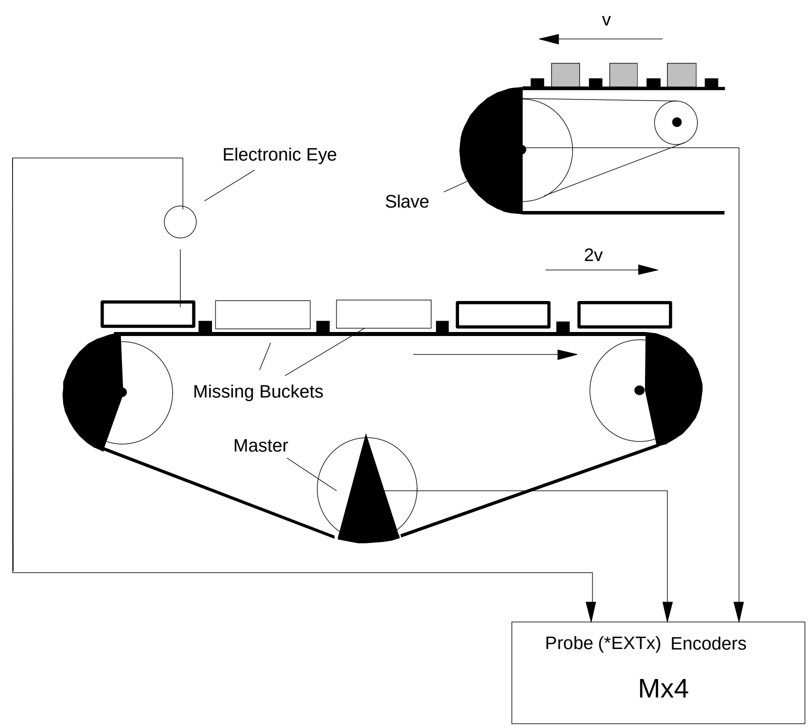
In this example, motion program `electronic_gearing` starts an endless loop in which variable `gear_ratio (VAR4)` is continually updated. You may use the second task (permitted in DSPL programming) to calculate `gear_ratios` on-the-fly. Alternatively, if the host is to update `gear_ratios`, the host based real time command `CHANGE_VAR` (contained in Mx4 C++ or Visual Basic DLL) can be used to update `VAR4`.

```
#define master      var2
#define slave       var3
#define     gear_ratio var4
#include    "init_mx4.hll"

plc_program
    run_m_program(electronic_gearing)
end_plc


master = 1            ;select axis 1 as master
slave  = 2            ;select axis 2 as slave
gear_ratio = 2
```

```
electronic_gearing:

    while (var1 == 1)      ;changing var1 (by host) disengages slave
            gear(master, slave, gear_ratio)
            delay(100)
    wend
    gear_off_acc(2)
end
```

3) Engage in electronic gearing by an external signal

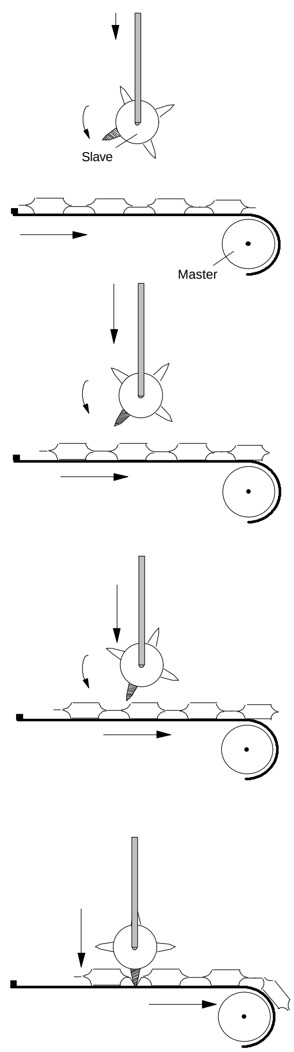
In this example, the slave is geared to the master motor only if the pulse sent by the electronic eye is switched to logic zero. This feature is useful in applications where there may be a problem on the line such as missing bucket.



```
#define master  var2
#define slave   var3
#define     gear_ratio var4
#include    "init_mx4.hll"

plc_program
    run_m_program(electronic_gearing)
end_plc


master = 1                          ;select axis 1 as master
slave  = 2                          ;select axis 2 as slave
gear_ratio = 2

electronic_gearing:

    velmode (1,5)                   ;put master in velocity control mode

    gear_probe(master, slave, 1, gear_ratio)

    wait_until(INP1_REG & 0x0002) ;wait until stop button is pushed
    gear_off_acc(2)
end
```

4) Engage in electronic gearing when master passes a programmed position

Products on the conveyor belt moved by the master motor are positioned uniformly. The slave motor cuts the film connecting the two adjacent products. The result of this cut is unsatisfactory if the knife lands vertically. It is preferred that while landing, the knife edge travels and is tightly geared to the position of film that must be cut. This is shown in the following figure.



```
#define master  var2
#define slave   var3

#include    "init_mx4.hll"

plc_program
    run_m_program(electronic_gearing)
end_plc


master = 1             ;select axis 1 as master
slave  = 2             ;select axis 2 as slave
gear_ratio = 1

electronic_gearing:


    gear_pos(master, slave, gear_ratio, 200);engage when master passed 200
    velmode (1,5)                        ;start master move
    wait_until(INP1_REG & 0x0002)        ;wait for stop button
    gear_off_acc(2)                      ;stop slave
    stop(1)                              ;stop master
end
```

6 Homing Programs

Single-Axis Homing

This program describes automatic homing for an axis. We assume that axis 1 home switch is connected to the Mx4 input IN1. The negative and positive homing speeds are set to a small value.

The process of homing starts with driving toward the home switch. Upon the recipt of this signal the axis decelerates to a stop, index (marker) pulse interrupt is enabled and a move in opposite direction is initiated. Upon the recipt of index pulse interrupt, the location of index pulse is saved in `reference_pos` and the axis decelerates to a stop. The move parameter, `reference_pos`, in conjunction with trapezoidal move command, `AXMOVE`, will drive the axis to the marker position.

```
#define neg_homing_vel      var2
#define pos_homing_vel      var3
#define reference_pos       var4

plc_program:

   run_m_program(go_home)

end

go_home:
   neg_homing_vel = -.5           ;negative homing velocity
   pos_homing_vel = .1            ;positive homing velocity

   ; Assume the Mx4 Input IN1 is connected to the home position switch.

   velmode(0x1, neg_homing_vel)   ;move toward home switch

   wait_until(inp1_reg & 0x0002)  ;while home switch isn't set

   int_reg_clr(0x0001, 0x1)       ;clear index pulse interrupt
   en_index(0x1)                 ;enable index pulse interrupt ax1
```

```
stop(0x1)                        ;stop immediately

while(~index_reg & 0x0001)       ;while no index interrupt set
  velmode(0x1, pos_homing_vel)   ;move towards home
wend
stop(0x1)                         ;stop immediately

  reference_pos = index_pos1      ;reference position saves the marker position
  axmove(1, .1, reference_pos, neg_homing_pos)
                                  ;go to reference position
end
```

Multi-Axis Homing

This program describes automatic homing for multiple axes. We assume that axis 1 and axis 2 home switches are connected to the Mx4 inputs IN1 and IN3 respectively. The negative and positive homing speeds are set to small values. The process of homing starts with driving toward the home switches. Upon receipt of these signals the two axes decelerate to a stop, index (marker) pulse interrupt is enabled and a move in opposite direction is initiated. Upon the receipt of index pulse interrupt, the locations of these index pulses are saved in `reference_pos1` and `reference_pos2`, and both axes decelerate to a stop. The move parameters, `reference_pos1`, and `reference_pos2`, in conjunction with trapezoidal move command, `axmove`, will move the axes to the marker position.

```
#define neg_homing_vel   var2
#define pos_homing_vel   var3
#define reference_pos1   var4
#define reference_pos2   var5

plc_program:

       run_m_program(go_home)

end

go_home:
   neg_homing_vel = -.5              ;negative homing velocity
   pos_homing_vel = .1               ;positive homing velocity

   velmode(0x3, neg_homing_vel, neg_homing_vel)  ; move toward home switch

   wait_until((inp1_reg & 0x0002) OR (inp1_reg & 0x0004))
                                    ; wait for home switches for axis 1 or
                                    ; axis 2 to set
   stop(0x3)                         ;stop axis 1 & 2 immediately
   while(~inp1_reg & 0x0002)         ;test to see if axis 1 is at home
        velmode(0x1, neg_homing_vel);axis 1 go towards home switch
   wend
   stop(0x1)                         ;stop axis 1 immediately

   while(~inp1_reg & 0x0004)         ;test to see if axis 2 is at home
        velmode(0x2, var2)           ;axis 2 go towards home switch
   wend
   stop(0x2)                         ;stop axis 2 immediately
```

```
  int_reg_clr(0x0001, 0x1)              ;clear index pulse interrupts
  en_index(0x1)                         ;enable index pulse interrupt ax1 & 2

  while(~index_reg & 0x0001)            ;while no index interrupt set
     velmode(0x1, var3)                 ;move towards home
  wend
  stop(0x1)

  int_reg_clr(0x0001, 0x2)              ;clear index pulse interrupts
  en_index(0x2)                         ;enable index pulse interrupt ax1 & 2

  while(~index_reg & 0x0002)            ;while no index interrupt set
     velmode(0x2, var3)                 ;move towards home
  wend
  stop(0x2)

  reference_pos1 = index_pos1      ; reference position saves the marker position
  reference_pos2 = index_pos2      ; reference position saves the marker position
  axmove(0x3, .1, reference_pos1, neg_homing_pos, .1, reference_pos2, neg_homing_pos)
                                   ; go to reference position
end
```

7 External Signal Interrupt

High Speed Position Capture Using External Interrupt

This program describes high speed position capture using external interrupt signal (*EXTx, referred to as probe).

The program will first run axis 1 in velocity mode. Second, one of the two external interrupts (*EXT2) is enabled. This is done after this signal's corresponding interrupt register is cleared. Upon the recipt of probe interrupt, the captured positions for axes 1 through 4 are saved. To indicate the termination of capture, and only as a test, we preset the position of axis 4 to this value. Make sure axis 4 is not connected to an amplifier or amplifier is disabled.

```
#define captured_pos1   var3
#define captured_pos2   var4
#define captured_pos3   var5
#define captured_pos4   var6

plc_program:
  run_m_program (capture_position)
end

capture_position:

  velmode(0x1, 1)

  int_reg_clr(0x0008, 0x2)       ; clear probe_int register
  en_probe(2, 2)                 ; enable probe 2, and echo to DPR


  wait_until(probe_reg & 0x0002)  ; wait for probe 2

  captured_pos1 = probe_pos1     ; position of axis 1 at time of probe int
  captured_pos2 = probe_pos2     ; position of axis 2 at time of probe int

  captured_pos3 = probe_pos3     ; position of axis 3 at time of probe int
  captured_pos4 = probe_pos4     ; position of axis 4 at time of probe int
  pos_preset(0x8, captured_pos4)   ; preset position of axis 4 to indicate ;capture

end
```

This page intenionally blank.

8 Position Break-Point Interrupt

Position Break-Point Activated Outputs

The position break-point interrupt is helpful in applications where interrupt is to be generated based on the position of an axis passing a programmed set point while move is in progress. The DSPL command which initiates such interrupt is `EN_POSBRK`. In addition to generation of interrupt, DSPL command `POSBRK_OUT` sets the programmed logic outputs.

The following DSPL program enables position break-point interrupt. This is done after clearing the corresponding interrupt register and programming the outputs to turn on (see `POSBRK_OUT`) at the break-point position. The position break-point interrupt is enabled to trigger at x=15000 and at y=15000. This is followed by a trapezoidal move command `AXMOVE` to move both axes to positions 28000. Clearly, in the process of achieving 28000, they must pass 15000 at which point interrupt is generated. The receipt of this interrupt is acknowledged by seven(7) output signals turned on. Next the position break-point interrupt is re-enabled to trigger at location x=3000 y=3000. The second `AXMOVE` command moves axes 1 and 2 to positions 0 and 0. The program waits until a position break-point interrupt is generated. This happens while move is in progress. The receipt of this interrupt is acknowledged by turning off all previously turned on signals.

```
plc_program:

  run_m_program (set_output_logic)

end


set_output_logic:

  int_reg_clr(0x0002, 0x3)          ;clear the pos_brk int register
  posbrk_out(0x1,0x1555,0x0000)     ;set output on mask
  en_posbrk(0x3, 15000, 15000)      ;enable position interrupt for axes 1,2


             ;to set at x=15000, y= 15000
```

```
    axmove(0x3, .1, 28000, 5, .1, 28000, 5)
    wait_until(posbrk_reg & 0x0003)   ;wait until position passed 15000

    int_reg_clr(0x0002, 0x3)          ;clear the pos_brk int register
    posbrk_out(0x1,0x0000,0x1555)     ;set outputs off
    en_posbrk (0x3, 3000, 3000)       ;enable position break-point
                                      ;to set at x=3000, y= 3000
    axmove(0x3, .1, 0, 5, .1, 0, 5)
    wait_until(posbrk_reg & 0x0003)   ;wait until position passed 3000

end
```

Axis Exceeds Set Position Interrupt

Position break-point interrupt is helpful in applications where interrupt is generated based on the position of an axis passing a programmed set point during the move. The DSPL command which will initiate such interrupt is EN_POSBRK

The program first enables position break-point interrupt. This is done after clearing the corresponding interrupt register. The positions break-point interrupt is enabled to trigger at x=15000 and y=15000. This is followed by a trapezoidal move command AXMOVE to move both axes to position 28000. Clearly, in the process of achieving 28000, position will pass 15000 at which point interrupt is generated. The receipt of this interrupt is acknowledged by presetting axis 4 to 444. Make sure axis 4 is not connected to an amplifier. Next the position break-point interrupt is re-enabled to trigger at location x=3000 y=3000. The second AXMOVE command moves axes 1 and 2 to positions 0 and 0. The program waits until a position break-point interrupt is generated. This happens while move command is in progress. The receipt of this interrupt is acknowledged by presetting axis 4 to 555.

```
plc_program:

    run_m_program (issue_position_int)

end
```

```
issue_position_int:

int_reg_clr(0x0002, 0x3)          ;clear the pos_brk int register
en_posbrk(0x3, 15000, 15000)      ;enable position interrupt for axes 1,2
                                  ;to set at x=15000, y= 15000
```

```
axmove(0x3, .1, 28000, 5, .1, 28000, 5)
wait_until(posbrk_reg & 0x0003) ;wait until position passed 15000

pos_preset(0x8, 444)            ;indicate the occurrence of the interrupt

int_reg_clr(0x0002, 0x3)        ;clear the pos_brk int register
en_posbrk (0x3, 3000, 3000)     ;enable position break-point
                                ;to set at x=3000, y= 3000
axmove(0x3, .1, 0, 5, .1, 0, 5)
wait_until(posbrk_reg & 0x0003) ;wait until position passed 3000
pos_preset(0x8, 555)            ;indicate the occurrence of this interrupt

end
```

This page intentionally blank.

9 Motion Complete Interrupt

Motion complete (MC) interrupt indicates the completion of motion generated by the following commands:

```
AXMOVE       (all family members)
STOP
CUBIC_INT
```

MC interrupt, doesn't need to be re-enabled each time one is generated. However, to detect additional MC interrupts, after each MC occurrence, the MC interrupt register must be cleared.

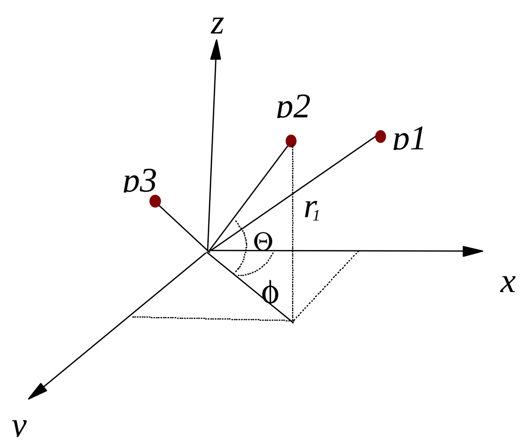
The program first enables the motion complete interrupt. This is done after the signals interrupt register is cleared. A trapezoidal motion command (AXMOVE) for axes 1 and 2 moves these axes to position 30000. Upon the receipt of an MC interrupt we preset the position of axis 4 (unconnected to an amplifier) to the value 444. Next, the MC interrupt register is cleared to accept another interrupt. The second AXMOVE command moves axes 1 and 2 back to position 0, 0. Upon the receipt of an MC interrupt we preset the position of axis 4 to the value 555.

```
plc_program:

        run_m_program (motion_complete_int)

end

motion_complete_int:

int_reg_clr(0x4, 0x3)          ;clear motion complete interrupt reg
en_motcp(0x3)                  ;enable motion complete interrupt
axmove(0x3, .1, 30000, 5, .1, 30000, 5)
wait_until(motcp_reg & 0x0003);wait for motion of axes 1&2 completed
pos_preset(0x8, 444)           ;indicate the completion of motion
int_reg_clr(0x4, 0x3)          ;clear motion complete interrupt reg


axmove(0x3, .1, 0, 5, .1, 0, 5)    ;move axes back to the starting point
wait_until(motcp_reg & 0x0003)     ;wait until motion is completed
pos_preset(0x8, 555)

end
```

This page intentionally blank

10 Moves in Polar Coordinate

This application describes the DSPL programming for moves in polar co-ordinate.



The application program moves a three-axis motion system from *p*1 to *p*2 and *p*3 in the polar coordinate. The three points, *p*1, *p*2 and *p*3 are characterized by their *r*, Q and f as follows:

*p*1: r_1, Q_1 and f_1
*p*2: r_2, Q_2 and f_2
*p*3: r_3, Q_3 and f_3

The following illustrates `"main.hll"` that performs the required moves. This program uses external routines contained in programs `"coordinate_xfer.hll"` and `"get_a_point.hll"`.

Polar Coordinate Move, ‘main.hll’

```
#define   x   var20
#define   y   var21
#define   z   var22

#define   teta    var23
#define   phi var24
#define   r   var25
#define index var26
#include “coordinate_xfer.hll”
#include “get_a_point.hll”

plc_program:

  run_m_program (move_in_polar_coordinate)

end_plc

move_in_polar_coordinate:

  var1 = 1
  while (var1 == 1)

    call (get_a_new_point)      ;get a point provided by either
    call (polar2cartesian)      ;the Mx4(case 1) or the host(case 2)

  wend
end
```

Point Retrieving Subroutine, ‘get_a_point.hll’

Case 1: All points are computed and stored in Mx4 by the Mx4’s own DSPL

```
get_a_new_point:

    ;*********************************************************
    ;*
    ;*  this routine is useful if end points are computed
    ;*  by the Mx4 and stored in the Mx4 table.
    ;*
    ;*********************************************************

    r = table_p(index) ;pick r, teta and phi
    index = index + 1
    teta  = table_p(index)
    index = index + 1
    phi = table_p (index)
    index = index + 1

    ret()
end
```

Case 2: All points are provided to the Mx4 in real time by the host

```
get_a_new_point:

    ;*********************************************************
    ;*
    ;*  this routine is useful if end points are provided
    ;*  by the Mx4 and stored in the Mx4 table.
    ;*
    ;*********************************************************

    r = var30    ;host uses instruction change_var to update points
    teta = var31 ;to update points characterized by:r,teta and phi
    phi  = var32

    ret()

end
```

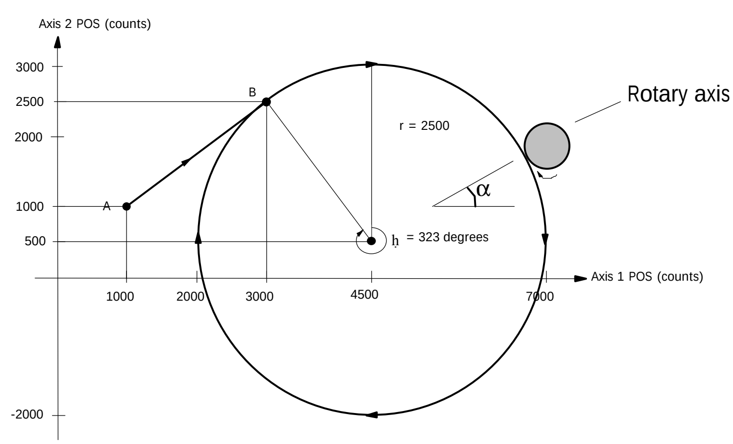
Polar to Cartesian Xformation, ‘coordinate_xfer.hll’

```
polar2cartesian:

    ;**********************************************************
    ;*
    ;*  this routine transfers polar to Cartesian
    ;*  coordinate.  And executes a trapezoidal move
    ;*  to reach the target point within a specified time
    ;*
    ;**********************************************************

    x = r * cos (phi)
    y = r * sin (phi)

    x = x * cos (teta)
    y = y * cos (teta)
    z = r * sin (teta)

    axmove_t(0x7, x_accel, x, y_accel, y, time, z, z_accel, time)

    ret()
end
```

11 Rotary Axis Tangent

Rotary Axis Tangent to x-y Trajectory

This application requires the motion of a rotary axis to remain tangent to the path created by x and y axes. The x-y trajectory in this example is circular. Assuming 1000 encoder lines/mech. rev. (i.e. 4000 counts/rev), one radian move of rotary axis generates 637 encoder counts. Thus, in conjunction with α in radians, this conversion factor must be used.




```
#define  del_x  var1
#define  del_y  var2
#define  a       var3
#define  alpha  var4
#define  flag   var5
#define  rotary var6

plc_program:

    run_m_program(tangential_path)

end
```

```
tangential_path:

    flag = 1
    pos_preset (0x7,1000,1000,0)    ;preset to point A
    linear_move_s(3,1000,0,3000,0.8,5000,0.0003,1000,0,2500,0.6,5000,0.00023); start AB line
    circle(3,1500,-2000,2500,1,0,0) ;continue with x-y circle
                                    ;compute position for rotary
                                    ;axis
    while (flag == 1)
        del_x = cvel1               ;obtain rate of change of position in x direction
        del_y = cvel2               ;obtain rate of change of position in y direction

        a = del_y/del_x             ;calculate tangent of alpha
        alpha = arctan(a)           ;find alpha in radians
        rotary = 637 * alpha        ;use conversion factor 637 to find encoder lines
        axmove(0x8, 0.5, rotary, 10)
                                    ;move rotary axis(3) to the computed position
    wend

end
```

12 Cubic Spline Programming

Introduction

Motion control applications requiring fine moves through a set of points require cubic spline interpolation. The Mx4 can run cubic splines either in contouring mode (in which the host continually updates Mx4's DPR with a new set of points), or in table mode (Mx4's table is pre-loaded with a set of points only once). In table mode the user array can be up to 2,000 points long. Each point specifies the position and velocity of only one motor.

The DSPL commands useful for cubic spline applications are:

`CUBIC_RATE`	Specifies the “time” interval between the two adjacent points in a cubic spline table. This instruction is similar to `BTRATE` (used in dual port RAM-based contouring applications).
`CUBIC_SCALE`	Specifies," position/velocity_multiplier” and “position_shift” for all points of a spline table.
`CUBIC_INT`	To run on "m" points of cubic spline table, "n" number_of_times. Starting from “si” starting index.

Three Steps to Run Cubic Spline

1) Download the data points using the *Tables* option in Mx4pro v4 on Windows 95/NT or *down_cub.exe* on DOS, (located in the *Mx4 Utilities diskette*).

Also, the DSPL offers floating point arithmatic and trigonometric functions by which new move parameters can be calculated in real time and stored in the table memory.

2) Run the DSPL command `CUBIC_RATE`. This command ***must*** run before issuing `CUBIC_INT`.

3) Use `CUBIC_INT` in your DSPL or host-based program.

We will now discuss six DSPL programs **--** starting from simple leading to more advanced applications.

Cubic Spline Trajectory on A Single Axis

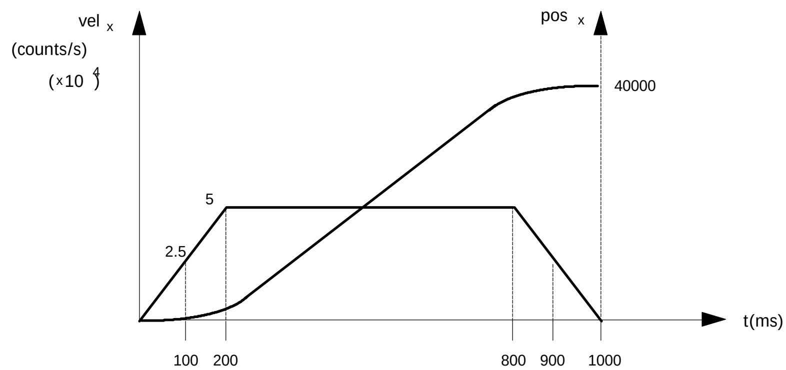
Consider a single axis move as illustrated. This trajectory is characterized by its position and velocity at times starting at zero and incrementing every 100 ms. In order to perform cubic spline contouring you must follow the steps as follows:

Step 1: Generate points

Step 2: Form an ASCII file that contains the points and download it to Mx4

Step 3: In your DSPL program use relevant instructions:

```
CUBIC_RATE()
CUBIC_SCALE()
CUBIC_INT()
```



This example helps you understand how a data table is organized.

The Data File for One-Axis Contouring Process

You need to generate an ASCII file similar to the following and save it under any name followed by .DAT, (e.g., CUB1.DAT).

Position (counts)	Velocity (counts/s)
0.00000000000000e+000	0.000e+000
1.25000000000000e+003	2.5000e+004
5.00000000000000e+003	5.0000e+004
1.00000000000000e+004	5.0000e+004
1.50000000000000e+004	5.0000e+004
2.00000000000000e+004	5.0000e+004
2.50000000000000e+004	5.0000e+004
3.00000000000000e+004	5.0000e+004
3.50000000000000e+004	5.0000e+004
3.87500000000000e+004	2.5000e+004
4.00000000000000e+004	0.0000e+004
3.87500000000000e+004	-2.5000e+004
3.50000000000000e+004	-5.0000e+004
3.00000000000000e+004	-5.0000e+004
2.50000000000000e+004	-5.0000e+004
2.00000000000000e+004	-5.0000e+004
1.50000000000000e+004	-5.0000e+004
1.00000000000000e+004	-5.0000e+004
5.00000000000000e+003	-5.0000e+004
1.25000000000000e+003	-2.5000e+004
0.00000000000000e+000	0.0000e+000

You may now download all (21) points to the Mx4 memory.

Memory Capacity

The Mx4 memory size dedicated to cubic spline is 8000 words. Each point on cubic spline contour is characterized by its position (32-bit) and velocity (32-bit), thus requiring four words. As a result, the total number of points that may be saved in an Mx4 cubic spline table is 2000.

Downloading a Table

To download your table at the DOS prompt type:

```
down_cub cub1.dat 1 0xd0000
```

This instruction downloads `CUB1.DAT` file for axis 1 in an Mx4 card located in address location 0xd0000 (see the *Mx4 User's Guide*, *Installing Your Mx4 Hardware*). Alternatively, you may use the Table download facility in Mx4pro v4 on Windows 95/NT.

DSPL Program

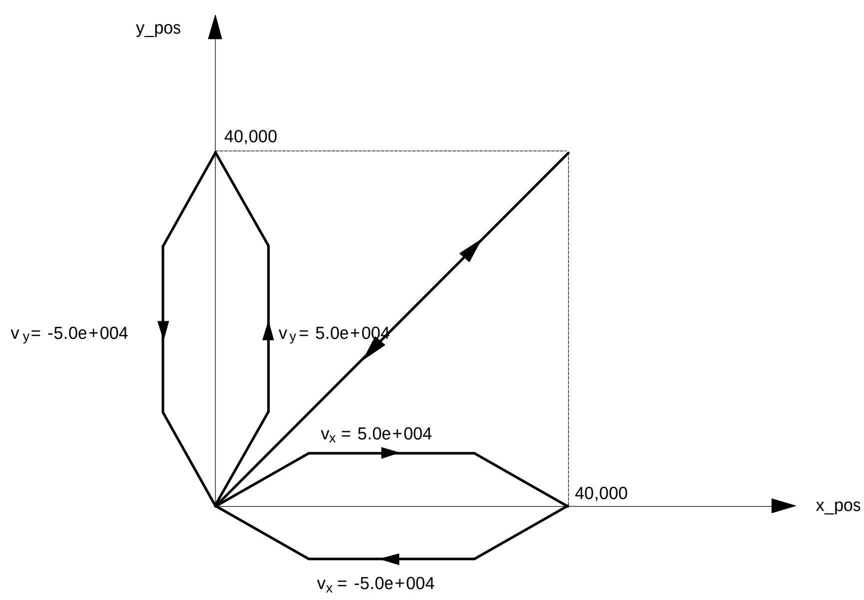
The steps following the transmission of the data table includes setting block transfer rate (`CUBIC_INT`), scaling (`CUBIC_SCALE`) and, running through the points (`CUBIC_INT`).

The following illustrates the DSPL program that runs through 21 points of `cub1.dat`.

```
plc_program:
  run_m_program(cubic)
end

cubic:
  cubic_rate(500)      ;set the cubic spline time interval to 100ms
  cubic_scale(0x1,1,0) ;set the pos and vel scales to 1 with no shift
  cubic_int(21,0,1)    ;run 21 points of the table only once
end
```

Cubic Spline Trajectory on Two Axes

This example is similar to the first one and is only modified for two axes. Our objective here is to show how the data points for an additional axis must appear in the data file.



To simplify our presentation, we use similar motions for x and y. In a general case x and y may have any arbitary shape.

ASCII File for Two-Axis Contouring Process

Position (counts)	Velocity (counts/s)	
0.00000000000000e+000	0.000e+000	← for axis x
0.00000000000000e+000	0.000e+000	← for axis y
1.25000000000000e+003	2.5000e+004	← for axis x
1.25000000000000e+003	2.5000e+004	← for axis y
5.00000000000000e+003	5.0000e+004	
5.00000000000000e+003	5.0000e+004	
1.00000000000000e+004	5.0000e+004	
1.00000000000000e+004	5.0000e+004	
1.50000000000000e+004	5.0000e+004	
1.50000000000000e+004	5.0000e+004	
2.00000000000000e+004	5.0000e+004	
2.00000000000000e+004	5.0000e+004	
2.50000000000000e+004	5.0000e+004	
2.50000000000000e+004	5.0000e+004	
3.00000000000000e+004	5.0000e+004	
3.00000000000000e+004	5.0000e+004	
3.50000000000000e+004	5.0000e+004	
3.50000000000000e+004	5.0000e+004	
3.87500000000000e+004	2.5000e+004	
3.87500000000000e+004	2.5000e+004	
4.00000000000000e+004	0.0000e+004	
4.00000000000000e+004	0.0000e+004	
3.87500000000000e+004	-2.5000e+004	
3.87500000000000e+004	-2.5000e+004	
3.50000000000000e+004	-5.0000e+004	
3.50000000000000e+004	-5.0000e+004	
3.00000000000000e+004	-5.0000e+004	
3.00000000000000e+004	-5.0000e+004	
2.50000000000000e+004	-5.0000e+004	
2.50000000000000e+004	-5.0000e+004	
2.00000000000000e+004	-5.0000e+004	
2.00000000000000e+004	-5.0000e+004	
1.50000000000000e+004	-5.0000e+004	
1.50000000000000e+004	-5.0000e+004	
1.00000000000000e+004	-5.0000e+004	
1.00000000000000e+004	-5.0000e+004	
5.00000000000000e+003	-5.0000e+004	
5.00000000000000e+003	-5.0000e+004	
1.25000000000000e+003	-2.5000e+004	
1.25000000000000e+003	-2.5000e+004	
0.00000000000000e+000	0.0000e+000	
0.00000000000000e+000	0.0000e+000	

Save this ASCII file as CUB2.DAT and download it to the Mx4 memory.

DSPL Program for Two-Axis Contouring

The following illustrates the DSPL program modified for two motors.

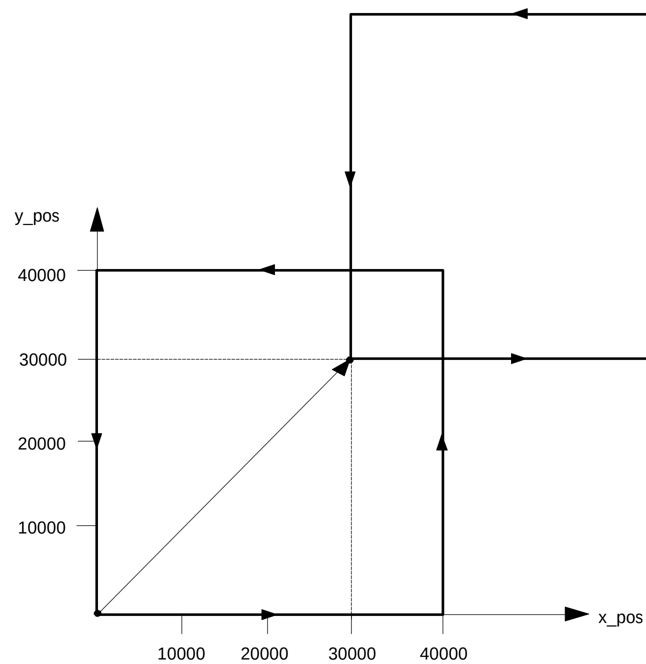
```
plc_program:
 run_m_program(cubic)
end

cubic:
 cubic_rate(500)          ;set the cubic spline time interval to 100ms
 cubic_scale(0x3,1,0,1,0) ;scale the pos and velocity scales to 1 and no shift
 cubic_int(42,0,1)         ;run 42 points of cub2.dat file only once
end
```

Dynamic Scaling and Coordinate Transformation

Motion control applications involving cubic spline may be scaled or coordinate transformed. Scaling means the real-time multiplication of "all" positions and/or velocities by a set value. This feature may be used to change coordinated speed, vectorially. The position vector may be magnified or attenuated accordingly.

Coordinate transformation (shift) performs the real-time position shift of Cartesian coordinates. That is, this command in conjunction with cubic spline will shift, the position of all axes to a new origin. The RTC used for this task is CUBIC_SCALE.

Consider our previous example, in which the system continually repeats the same motion. Now imagine after cutting a shape, the operator, wishes to transform the coordinates to a new origin specified by its positions in x and y directions (e.g.,30000,30000).

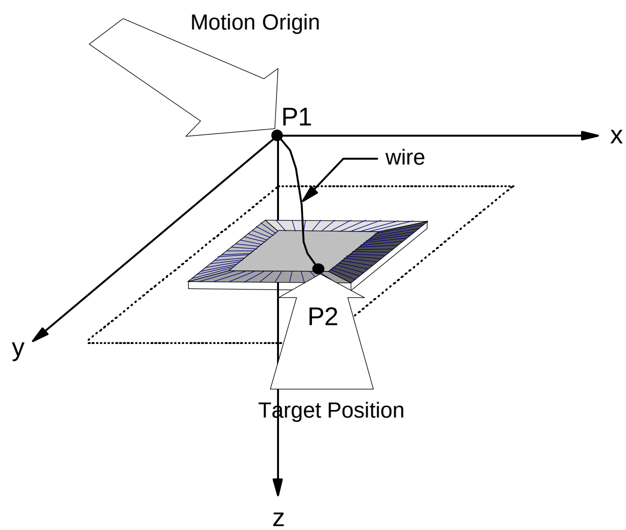


The following command shows how this coordinate transfer is accomplished:

```
CUBIC_SCALE(0x3,1,30000,1,30000)
```

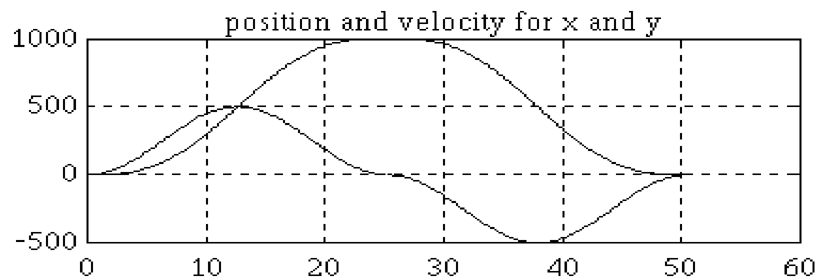
High Speed Moves with User Defined Trajectories

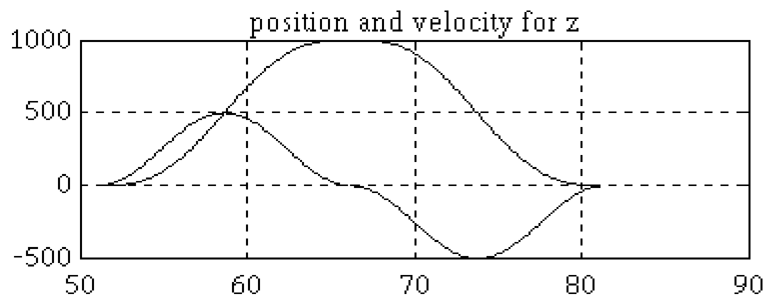
This application coordinates x,y, z (and w in a later example) axes to perform series of high speed (10-50 ms travel time) contouring moves. An example of such application is semiconductor wire bonding. We describe the DSPL programs that achieve the target points for x,y, and z along the user-defined 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 download utilities:

   I) *down_tbl.exe* in DOS or;
   ii) *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 which 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 down_tbl.exe utility as follows:

```
c:\>down_tbl  points.dat  800
```

The parameter "800" indicates at which starting index to begin downloading the data points in the Mx4 memory. Alternatively, you may use the Mx4pro v4's Windows 95 or NT table download.

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 functions of each of DSPL programs /files 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 xy 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  aux3           var11
#define  index_cur_vel  var12
```

```
#define  scale          var13
#define  index_cur_posz var14
#define  velocity       var15
#define  coded_pve_vel  var19

#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  index_neg_vel  var43
#define  coded_neg_vel  var46

#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_vyz  var55
#define  index_neg_vyz  var56
#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/2     ;because of the way trajectories are implemented

total_no_pts = 2*period_xy
```

```
 total_no_pts = total_no_pts + 2 ;total number of points for x and y
 total_no_ptz = 2*period_z
 total_no_ptz = total_no_ptz + 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(z_profile)            ;this routine calculates the points on z traj.
 wait_until(var2 == 1)

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

 index_cur_pos = 0

 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

    x_target_pos = table_p(table_pointer)  ;load target point for x
    table_pointer = table_pointer+1        ;increment index variable table_pointer

    y_target_pos = table_p(table_pointer)  ;load target point for y
    table_pointer = table_pointer+1        ;increment index variable table_pointer

    z_target_pos = table_p(table_pointer)  ;load target point for z
    table_pointer = table_pointer+1        ;increment index variable table_pointer
    if (table_pointer == 848)
             table_pointer = 800           ;when table finished loop over the table points
    endif

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

    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)
    cubic_int(total_no_pts,0,1)            ;run all x and y points
    cubic_int(total_no_ptz,init_z_table,1) ;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.  It also codes the x and y
        ;*               axes participation by adjusting the most
        ;*               significant nibble of velocity.
        ;*
        ;************************************************************

    flag = 0
    index_cur_pos = 0
    var3 = period_xy*4
    index_neg_vel = period_xy+1                  ;compensation for xy axes
    while (index_cur_pos <= period_xy)           ;period_xy holds xy trajectory periods in ms

              index_dec_pos = 2*period_xy
              index_dec_pos = index_dec_pos+4
              index_dec_pos = index_dec_pos - index_cur_pos ; index into descending pos segment

              index_neg_vel = index_neg_vel + 4  ;index into negative velocity  segment

              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
              aux1 = sin(aux6)
              aux2 = cos(aux6)

              aux2 = 1 - aux2                         ;calculates [1 - cos (2pi*t/T)]
              aux2 = aux2/period_xy  ;
              aux2 = aux2/5                           ;calculates [1 - cos(2pi*t/T)]/(5*T)
                                                      ;velocity is in c/200 us
              aux1 = aux1*aux5                        ;calculates (T/2pi)*sin(2*pi*t/T)

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

                position = scale*aux1

                aux3 = index_cur_pos
                table_p(index_cur_pos) = position     ;save position
                table_p(index_dec_pos) = position     ;save for descending position

                index_cur_posy = index_cur_pos + 2
                index_dec_posy = index_dec_pos + 2

                table_p(index_cur_posy) = position
                table_p(index_dec_posy) = position

                index_cur_vel = aux3 + 1
                coded_pve_vel = aux2*scale
                velocity = coded_pve_vel

           ;******************************************************************
           ;*
           ;*                 The following shows how the DSPL
           ;*                 deals with the issue of coding axes
```

```
;*                 into the most significant nibble of
;*                 velocity.  You may read about this
;*                 coding requirement in the Mx4 User's
;*                 Guide under cubic spline contouring.
;*
;*****************************************************************
;
;     coded_pve_vel = coded_pve_vel + 4096    ;coding axis 1 positive
      coded_pve_vel = coded_pve_vel + 12288   ;coding axes 1 and 2 positive
;     coded_pve_vel = coded_pve_vel + 28672   ;coding axes 1,2 and 3 positive
;     coded_pve_vel = coded_pve_vel + 61440   ;coding axes 1,2,3 and 4 positive
;     coded_pve_vel = coded_pve_vel + 16384   ;coding axis 3 positive

      coded_pve_vel = coded_pve_vel*65536

      coded_neg_vel = -velocity
      coded_neg_vel = 65536*coded_neg_vel

;     coded_neg_vel=coded_neg_vel+536870912   ;coding axis1 negative
      coded_neg_vel=coded_neg_vel+1073741824  ;coding axes 1 and 2 negative
;     coded_neg_vel=coded_neg_vel+2147483648  ;coding axes 1,2 and 3 negative
;     coded_neg_vel=coded_neg_vel+0           ;coding axes 1,2,3 and 4 negative
;     coded_neg_vel=coded_neg_vel+1342177280  ;coding axis 3 negative

      table_p(index_cur_vel) = coded_pve_vel  ;velocity with axis coding
      table_p(index_neg_vel) = coded_neg_vel  ;save for negative velocity
      index_cur_vyz = index_cur_vel + 2
      index_neg_vyz = index_neg_vel + 2
      table_p(index_cur_vyz)=coded_pve_vel
      table_p(index_neg_vyz)=coded_neg_vel
      index_cur_pos = index_cur_pos + 4

 wend
 flag = 1
ret()
end



z_profile:

;*************************************************************
;*
;*              This routine calculates the
;*              points on z trajectory and saves them
;*              in the table.  It also codes the third
;*              axis participation by adjusting the most
;*              significant nibble of velocity.
;*
;*************************************************************

     flag = 0
     index_cur_pos = 8*period_xy
     index_cur_pos = index_cur_pos + 8      ;compensation for all segments
     period_z = period_z*2                  ;period_z holds the period
     index_neg_vel = period_z+1              ;that is the new period
     index_cur_posz = 0                     ;this plays the role of old index_cur_pos

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

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

```
      index_neg_vel = period_z + 1
      index_neg_vel = index_neg_vel + index_cur_pos
      index_neg_vel = index_neg_vel + 2   ;index into negative velocity segment
      index_neg_vel = index_neg_vel + 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)
      aux2 = cos(aux6)

      aux2 = 1 - aux2                      ;calculates [1 - cos(2pi*t/T)]
      aux2 = aux2/period_z                 ;
      aux2 = aux2/5                        ;calculates [1 - cos(2pi*t/T)]/(5*T)
                                           ;velocity is in c/200 us
      aux1 = aux1*aux5                     ;calculates (T/2pi)*sin(2*pi*t/T)

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

      position = scale*aux1

      index_cur_posz = index_cur_posz
      index_cur_posz = index_cur_posz+index_cur_pos
      table_p(index_cur_posz) = position       ;save position
      table_p(index_dec_pos) = position        ;save for descending position
      index_cur_vel = index_cur_posz + 1

      coded_pve_vel = aux2*scale
      velocity = coded_pve_vel

      ;*****************************************************************
      ;*
      ;*
      ;*                 The following shows how the DSPL
;*                deals with the issue of coding axes
;*                into the most significant nibble of
;*                velocity.  You may read about this
;*                coding requirement in the Mx4 User's
;*                Guide under cubic spline contouring.
;*
;*
;*****************************************************************
;
;     coded_pve_vel = coded_pve_vel + 4096    ;coding axis 1 positive
;     coded_pve_vel = coded_pve_vel + 12288   ;coding axes 1 and 2 positive
;     coded_pve_vel = coded_pve_vel + 28672   ;coding axes 1,2 and 3 positive
;     coded_pve_vel = coded_pve_vel + 61440   ;coding axes 1,2,3 and 4 positive
      coded_pve_vel = coded_pve_vel + 16384   ;coding axis 3 positive

      coded_pve_vel = coded_pve_vel*65536


      coded_neg_vel = -velocity
      coded_neg_vel = 65536*coded_neg_vel


;     coded_neg_vel=coded_neg_vel+536870912   ;coding axis1 negative
;     coded_neg_vel=coded_neg_vel+1073741824  ;coding axes 1 and 2 negative
;     coded_neg_vel=coded_neg_vel+2147483648  ;coding axes 1,2 and 3 negative
;     coded_neg_vel=coded_neg_vel+0           ;coding axes 1,2,3 and 4 negative
      coded_neg_vel=coded_neg_vel+1342177280  ;coding axis 3 negative
```

```
            table_p(index_cur_vel) = coded_pve_vel  ;velocity with axis coding
            table_p(index_neg_vel) = coded_neg_vel  ;save for negative velocity
            index_cur_posz = index_cur_posz+2
 wend
 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 which will down load 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. Also, 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 eliptic
;*          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  index_cur_vel    var12
#define  scale            var13
#define  coded_pve_velz   var14
#define  coded_neg_velz   var15
#define  position_z       var16
#define  velocity_z       var17
#define  last_z_pos       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  index_dec_pos    var42
#define  index_neg_vel    var43
#define  coded_neg_vel    var46

#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  velocity         var56
#define  index_cur_pyz    var57
#define  index_dec_pyz    var58

#define  index_cur_vyz    var60
#define  index_neg_vyz    var61
#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:
     flag1 = 1                          ;this tells host it can not send move parameters yet
     call(INIT)                         ;this routine initializes gains
     wait_until(var1 == 1)              ;variable 1 is a flag to show init is done
```

```
                                      ;program know it is done initializing

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

     call(xyz_profiler)               ;routine to calculates the points on xyz trajs
     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_increment = x_target_pos - x_cur_pos   ;x target point relative to current position
        y_increment = y_target_pos - y_cur_pos   ;y target point relative to current position
        z_increment = z_target_pos - z_cur_pos   ;z target point relative to current position

        aux1 = x_target_pos
        aux2 = y_target_pos
        aux3 = z_target_pos

        scaled_x = x_increment/scale             ;scaled x target relative to current position
        scaled_y = y_increment/scale             ;scaled y target relative to current position
        scaled_z = z_increment/scale             ;scaled z target relative to current position


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


        rate = 10*max                          ;make cubic spline rate proportional/max length
        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)

        flag1 = 0                               ;this tells host it can change move parameters
        cubic_int(total_no_pts,0,1)             ;run the previously entered moves
        cubic_rate(5)                           ;this has to be here to let cubic_int finish
        axmove(0x7,1.9,aux1,100,1.9,aux2,100,1.9,aux3,100) ;
        wait_until(cpos1 == aux1)

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

```
xyz_profiler:

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

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


     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*6
     index_neg_vel = period+1                   ;compensation for xy axes
     while (index_cur_pos <= period)            ;period holds xy trajectory periods in ms

               index_dec_pos = 2*period
               index_dec_pos = index_dec_pos+6
               index_dec_pos = index_dec_pos - index_cur_pos ;index into descending position

               index_neg_vel = index_neg_vel + 6             ;index into negative velocity

               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)
               aux2 = cos(aux6)

             aux2 = 1 - aux2                               ;calculates [1 - cos (2pi*t/T)]
             aux2 = aux2/period                            ;
             aux2 = aux2/5                                 ;calculates [1 - cos(2pi*t/T)]/(5*T)
                                                           ;velocity is in c/200 us
             aux1 = aux1*aux5                              ;calculates (T/2pi)*sin(2*pi*t/T)

             aux1 = index_cur_pos - aux1
             aux1 = aux1/period                            ;calc. [(t-T/2pi*sin(2pi*t/T)]/T
             aux4 = sqrt(aux4)                             ;calc. sqrt(1 - (t/T)^2)
             aux4 = 1 - aux4

             position = scale*aux1
             position_z = scale*aux4

             aux3 = index_cur_pos
             table_p(index_cur_pos) = position             ;save position
             table_p(index_dec_pos) = position             ;save for descending position

             index_cur_pyz = index_cur_pos + 2
             index_dec_pyz = index_dec_pos + 2
```

```
    table_p(index_cur_pyz) = position
    table_p(index_dec_pyz) = position
    index_cur_pyz = index_cur_pyz + 2
    index_dec_pyz = index_dec_pyz + 2
    table_p(index_cur_pyz) = position_z
    table_p(index_dec_pyz) = position_z

    index_cur_vel = aux3 + 1
    coded_pve_vel = aux2*scale
    velocity = coded_pve_vel

    velocity_z = position_z - last_z_pos
    velocity_z = velocity_z/5
    coded_pve_velz = velocity_z


;*****************************************************************
;*
;*
;*               The following segment shows how the DSPL
;*               codes the participating axes
;*               into the most significant nibble of
;*               velocity.  You may read about this
;*               coding requirement in the Mx4 User's
;*               Guide under cubic spline contouring.
;*
;*****************************************************************
;
;     coded_pve_vel = coded_pve_vel + 4096    ;coding axis 1 positive
;     coded_pve_vel = coded_pve_vel + 12288   ;coding axes 1 and 2 positive
      coded_pve_vel = coded_pve_vel + 28672   ;coding axes 1,2 and positive
      coded_pve_velz = coded_pve_velz + 28672 ;coding axis 3 positive
;     coded_pve_vel = coded_pve_vel + 61440   ;coding axes 1,2,3 and 4 positive
;     coded_pve_vel = coded_pve_vel + 16384   ;coding axis 3 positive

      coded_pve_vel = coded_pve_vel*65536
      coded_neg_vel = -velocity
      coded_neg_vel = 65536*coded_neg_vel
      coded_pve_velz = coded_pve_velz*65536
      coded_neg_velz = -velocity_z
      coded_neg_velz = 65536*coded_neg_velz

;     coded_neg_vel=coded_neg_vel+536870912   ;coding axis1 negative
;     coded_neg_vel=coded_neg_vel+1073741824  ;coding axes 1 and 2 negative
      var64 = 2147483647
      var64 = var64+1

      coded_neg_vel=coded_neg_vel+var64       ;coding axes 1,2 and 3 negative
      coded_neg_velz = coded_neg_velz + var64

;     coded_neg_vel=coded_neg_vel+0           ;coding axes 1,2,3 and 4 negative
;     coded_neg_vel=coded_neg_vel+1342177280  ;coding axis 3 negative

      table_p(index_cur_vel) = coded_pve_vel  ;velocity with axis coding
      table_p(index_neg_vel) = coded_neg_vel  ;save for negative velocity

      index_cur_vyz = index_cur_vel + 2
      index_neg_vyz = index_neg_vel + 2

      table_p(index_cur_vyz)=coded_pve_vel
      table_p(index_neg_vyz)=coded_neg_vel

      index_cur_vyz = index_cur_vyz + 2
      index_neg_vyz = index_neg_vyz + 2
```

```
            table_p(index_cur_vyz)=coded_pve_velz
            table_p(index_neg_vyz)=coded_neg_velz
            last_z_pos = position_z

            index_cur_pos = index_cur_pos+6
 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 which 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. Also, 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  index_cur_vel  var12
#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  index_neg_vel    var43
#define  coded_neg_vel    var46

#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  velocity         var56
#define  index_cur_pyz    var57
#define  index_dec_pyz    var58

#define  index_cur_vyz    var60
#define  index_neg_vyz    var61
#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 it can not 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 trajs
     wait_until(flag2 == 1)

                               ;target points.
                               ;
    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_increment = x_target_pos - x_cur_pos   ;x target point relative to current position
        y_increment = y_target_pos - y_cur_pos   ;y target point relative to current position
        z_increment = z_target_pos - z_cur_pos   ;z target point relative to current position
        w_increment = w_target_pos - w_cur_pos   ;w target point relative to current position

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

        scaled_x = x_increment/scale             ;scaled x target relative to current position
        scaled_y = y_increment/scale             ;scaled y target relative to current position
        scaled_z = z_increment/scale             ;scaled z target relative to current position
        scaled_w = w_increment/scale             ;scaled w target relative to current position

        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

        cubic_int(total_no_pts,0,1)             ;run all x and y points
        cubic_rate(5)
;        axmove(0xf,1.9,aux1,100,1.9,aux2,100,1.9,aux3,100,1.9,aux0,100)
        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_no_pts = total_no_pts + 4           ;total number of points for x,y,z and w

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*8
index_neg_vel = period+1                  ;compensation for xy axes
while (index_cur_pos <= period)           ;period holds xyzw trajectory periods in ms

          index_dec_pos = 2*period
          index_dec_pos = index_dec_pos+8
          index_dec_pos = index_dec_pos - index_cur_pos ;index into descending pos seg
          index_neg_vel = index_neg_vel + 8             ;index into negative vel.segment

          2pi = 2*pi
          aux4 = 2pi/period                             ;calculates 2pi/T
          aux5 = 1/aux4                                 ;calculates T/2pi
          aux6 = aux4*index_cur_pos                     ;calculates 2pi*t/T
          aux1 = sin(aux6)
          aux2 = cos(aux6)

          aux2 = 1 - aux2                               ;calculates [1 - cos (2pi*t/T)]
          aux2 = aux2/period                            ;
          aux2 = aux2/5                                 ;calculates [1 - cos(2pi*t/T)]/(5*T)
                                                        ;velocity is in c/200 us
          aux1 = aux1*aux5                              ;calculates (T/2pi)*sin(2*pi*t/T)

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

          position = scale*aux1

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

          index_cur_pyz = index_cur_pos + 2
          index_dec_pyz = index_dec_pos + 2
          table_p(index_cur_pyz) = position             ;save position for Y
          table_p(index_dec_pyz) = position

          index_cur_pyz = index_cur_pyz + 2
          index_dec_pyz = index_dec_pyz + 2
          table_p(index_cur_pyz) = position             ;save position for Z
          table_p(index_dec_pyz) = position

          index_cur_pyz = index_cur_pyz + 2
          index_dec_pyz = index_dec_pyz + 2
          table_p(index_cur_pyz) = position             ;save position for W
          table_p(index_dec_pyz) = position



          index_cur_vel = aux3 + 1
          coded_pve_vel = aux2*scale
          velocity = coded_pve_vel
```

```
;*****************************************************************
;*
;*
;*                  The following segment shows how the DSPL
;*                  codes the participating axes
;*                  into the most significant nibble of
;*                  velocity.  You may read about this
;*                  coding requirement in the Mx4 User's
;*                  Guide under cubic spline contouring.
;*
;*****************************************************************
;
;      coded_pve_vel = coded_pve_vel + 4096    ;coding axis 1 positive
;      coded_pve_vel = coded_pve_vel + 12288   ;coding axes 1 and 2 positive
;      coded_pve_vel = coded_pve_vel + 28672   ;coding axes 1,2 and 3 positive
       coded_pve_vel = coded_pve_vel + 61440   ;coding axes 1,2,3 and 4 positive
;      coded_pve_vel = coded_pve_vel + 16384   ;coding axis 3 positive

       coded_pve_vel = coded_pve_vel*65536
       coded_neg_vel = -velocity
       coded_neg_vel = 65536*coded_neg_vel

;      coded_neg_vel=coded_neg_vel+536870912   ;coding axis1 negative
;      coded_neg_vel=coded_neg_vel+1073741824  ;coding axes 1 and 2 negative
       var64 = 2147483647
       var64 = var64+1
;      coded_neg_vel=coded_neg_vel+var64       ;coding axes 1,2 and 3 negative
       coded_neg_vel=coded_neg_vel+0           ;coding axes 1,2,3 and 4 negative
;      coded_neg_vel=coded_neg_vel+1342177280  ;coding axis 3 negative

       var64 = 2*var64                         ;coding when axis 4 is involved
       coded_pve_vel =  var64 - coded_pve_vel  ;coding when axis 4 is involved
       coded_pve_vel = -coded_pve_vel          ;coding when axis 4 is involved

       table_p(index_cur_vel) = coded_pve_vel  ;velocity with axis coding for X
       table_p(index_neg_vel) = coded_neg_vel  ;save for negative velocity

       index_cur_vyz = index_cur_vel + 2
       index_neg_vyz = index_neg_vel + 2
       table_p(index_cur_vyz)=coded_pve_vel   ;velocity with axis coding for Y
       table_p(index_neg_vyz)=coded_neg_vel

       index_cur_vyz = index_cur_vyz + 2
       index_neg_vyz = index_neg_vyz + 2

       table_p(index_cur_vyz)=coded_pve_vel  ;velocity with axis coding for Z
       table_p(index_neg_vyz)=coded_neg_vel

       index_cur_vyz = index_cur_vyz + 2
       index_neg_vyz = index_neg_vyz + 2

       table_p(index_cur_vyz)=coded_pve_vel  ;velocity with axis coding for W
       table_p(index_neg_vyz)=coded_neg_vel

       index_cur_pos = index_cur_pos+8
 wend
 flag2 = 1
ret()
end
```

Appendix A

```
/*********************************************************************
```

Program Process.c

```
        This application will send X, Y, Z, and W end points to the Mx4 card
        using the C/C++ DLL, MX4WPL.DLL.  The functions mainly used are
        monitor_var, change_var, and var.

        The algorithm is as follows,

         1. Everytime Process() is called, var34 on the Mx4 card is
                checked.  If var34 = 1, then we exit the Process() procedure.
                If var34 = 0, then we continue on...

         2. At this point, var34 = 0.  Now we send the new end points for
                X, Y, Z, and W to the Mx4 card.  That is we set var22 = X end point
                var27 = Y end point, and var28 = Z end point.

         3. We set var34 = 1 to notify the DSPL that we have sent the
                new end points.

**********************************************************************/

#include <windows.h>
#include "mx4wpl.h"
#include "Process.h"

void Process(HWND hwnd)
{
        static double dX = 0 ;                      // X target position
        static double dY = 0 ;                      // Y target position
        static double dZ = 0 ;                      // Z target position
        static double dW = 0 ;                      // W target position
        static int iIndex = 0 ;                     // Index into points

        // Hard coded end points, these could come from a file instead
        static double dPts[20] = {0,1,2,3,4,5,6,7,8,9,10,9,8,7,6,5,4,3,2,1};


        // Set the new end points
        dX = dPts[iIndex] * 1000.0 ;
        dY = dPts[iIndex] * 1000.0 + 250.0;
        dZ = dPts[iIndex] * 1000.0 + 500.0;
        dW = dPts[iIndex] * 1000.0 + 750.0;

        // Set axis Z to 100000 to test if the cubic rate is changing
        if(iIndex == 5)
                dZ = 100000 ;

        // Set axis Z to 10000 to test if the cubic rate is changing
        if(iIndex == 15)
                dZ = 10000 ;
```

```
        // Check if Flag = 0, NOTICE: This requires that var39 is being
        // updated to VARIABLE viewing window #1
        if(var(1) == 1.0)
                return ;

        // Change the variables to the new end points
        begin_RTC();
                change_var(22, dX);
                change_var(27, dY);
                change_var(28, dZ);
                change_var(15, dW);
        end_RTC();

        // Flag the DSPL that vars have been changed
        change_var(34, 1.0);

        // Get the new index point into the endpoints table
        iIndex = (iIndex + 1) % 20;
}

***************************************************************************

// Header file for Processing The Handshaking of points

void Process(HWND hwnd);

****************************************************************************
/**********************************************************************
```

Program Target.c

```
        This application will send X, Y, Z, and W end points to the Mx4 card
        using the C/C++ DLL, MX4WPL.DLL.  The functions mainly used are
        monitor_var, change_var, and var.

        The algorithm for this program (without the window handling)
        is as follows,

         1.     Every TIMER ms (see the #define below) the procedure Process()
                is called.

        The algorithm for Process() is as follows,

         1. Everytime Process() is called, var34 on the Mx4 card is
                checked.  If var34 = 1, then we exit the Process() procedure.
                If var34 = 0, then we continue on...

         2. At this point, var34 = 0.  Now we send the new end points for
                X, Y, and Z to the Mx4 card.  That is we set var22 = X end point
                var27 = Y end point, and var28 = Z end point.

         3. We set var34 = 1 to notify the DSPL that we have sent the
                new end points.

**********************************************************************/


#include <windows.h>
#include <string.h>
#include "mx4wpl.h"
#include "Process.h"

// Global definitions
```

```
#define ID_START_BUTTON 100
#define ID_STOP_BUTTON  101
#define ID_CLOSE_BUTTON 102

// Timer in milliseconds
#define TIMER 50

// Global handles
HWND hposition;
HWND herror;
HWND hvelocity;

// Function prototypes
long FAR PASCAL TargetWndProc( HWND hwnd, UINT message,
                                                      WPARAM wparam, LPARAM lparam );


/**********************************************************************

  WinMain

        This is the main windows procedure.  Processes the message loop.

**********************************************************************/
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                                        LPSTR lpCmdLine, int nCmdShow)
{
        WNDCLASS wc;                                    // Window Class
        HWND hwnd;                                      // Handle to the main window
        MSG msg;                                        // The message
        static char buffer[20];                         // For checking the signature

        if (!hPrevInstance){
                wc.style = NULL;
        wc.lpfnWndProc = TargetWndProc;
                wc.cbClsExtra = 0;
                wc.cbWndExtra = 0;
                wc.hInstance = hInstance;
        wc.hIcon = LoadIcon( hInstance, "Target");
                wc.hCursor = LoadCursor(NULL, IDC_ARROW);
                wc.hbrBackground = (HBRUSH) (COLOR_BTNFACE+1);
                wc.lpszMenuName =  NULL;
        wc.lpszClassName = "TargetWndClass";

                // Register the class
                if (!RegisterClass(&wc))
                        return FALSE;
        }

        // Verify that the Mx4 or DM4 was found at the address in the DSPCG.INI file
        if (_fstrncmp( signature( buffer ), "MX4", 3 )!= 0){
                if (_fstrncmp( signature( buffer ), "DM4", 3 )!= 0){
                        MessageBox( NULL, "Mx4 Not Found", "", MB_OK );
                        return NULL;
                }
        }

        // Set up the position and time units for the DLL
        time_unit(1);
        position_unit(1);

        // Create the windows
        hwnd = CreateWindow("TargetWndClass","Target", WS_SYSMENU | WS_OVERLAPPED,
                    CW_USEDEFAULT, CW_USEDEFAULT, 125, 180, NULL,NULL, hInstance, NULL );
```

```
        CreateWindow( "button", "Start", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                            10, 10, 100, 35, hwnd, ID_START_BUTTON, hInstance, 0L );

        CreateWindow( "button", "Stop", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                            10, 60, 100, 35, hwnd, ID_STOP_BUTTON, hInstance, 0L );

        CreateWindow( "button", "Close", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                            10, 110, 100, 35, hwnd, ID_CLOSE_BUTTON, Instance, 0L );

        // Show and update the windows
        ShowWindow(hwnd, nCmdShow);
        UpdateWindow(hwnd);

        // Process the messages
        while (GetMessage(&msg,NULL,NULL,NULL)){
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }

        return (msg.wParam);
}

/**********************************************************************

  TargetWndProc

        Handles the messages.

**********************************************************************/
long FAR PASCAL TargetWndProc( HWND hwnd, UINT message,
                                                WPARAM wparam, LPARAM lparam )
{
        switch( message ){

                case WM_COMMAND:

                        switch ( wparam ){

                                case ID_START_BUTTON:

                                        // Send the monitor var RTC
                                        monitor_var(1, 34);       // Flag variable

                                        // Start the timer
                                        SetTimer( hwnd, 1, TIMER, NULL );

                                        break;

                           case ID_STOP_BUTTON:

                                        // Kill the timer
                                        KillTimer( hwnd, 1 );

                                        break;

                           case ID_CLOSE_BUTTON:

                                     // Send the close message
                                     SendMessage( hwnd, WM_CLOSE, 0, 0L );
                                     break;
                        }
                        break;

                case WM_TIMER:
```

```
                        // Process the handshaking
                        Process(hwnd) ;
                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;

                default:
                        return DefWindowProc(hwnd, message, wparam, lparam);
        }
        return NULL;
}
```

13 Cam Applications

The DSPL commands useful for cam applications are:

i) Commands used by all cam applications

```
CAM         ;engages a cam function unconditionally
CAM_OFF     ;disengages cam
CAM_OFF_ACC ;disengages cam and decelerates slaves to a stop
CAM_POS     ;engages cam based on a programmed position
CAM_PROBE   ;engages cam when an external signal is set
CAM_TSIZE   ;sets the total table length
```

ii) Command used by applications requiring cyclic error corrections

```
REL_AXMOVE_SLAVE  ;moves slaves relative to slave position(s)
```

iii) Command used by applications requiring several Mx4 cards (one master and up to 127 slaves)

```
SYNC       ;synchronizes a slave Mx4 card to a master Mx4 card
```

The following starts from general to more specific applications.

1. Ordinary cam used in a four-axis master/slave application (one axis is master and up to three axes are slaves).

2. Ordinary cam used in an up to 128-axis master/slave application (one axis is master and the remaining axes, using several Mx4 cards, are slaves).

3. Cam functions used in cyclic slave position corrections.

Simple Cam Function with One Master & up to Three Slaves

The first application uses a single Mx4 card. One of the axes is selected as master and up to three axes are slaves. There are three DSPL commands that turn on a CAM, function. The first command, CAM, starts cam unconditionally.

The second command, `CAM_POS`, starts cam when master axis has passed a programmed position. Finally, the third command, `CAM_PROBE`, starts cam upon the resetting of an external high speed input signal referred to as probe (*EXTx).

There are two cam disengaging commands: `CAM_OFF` and `CAM_OFF_ACC`. The first, `CAM_OFF`, disengages a cam function immediately. The second command, `CAM_OFF_ACC`, disengages the slave(s) and stops them at the programmed acceleration rate.

The procedure to run a complete cam function involves the following steps.

1) Choose a “master position space” defined as the master position displacement for the adjacent gear ratios of a cam table. For example, master position space of 5 means for every 5 counts of master move the index to the gear ratio table (also referred to as cam table) will be incremented by one.

2) Download the cam table to the Mx4 memory.

   The functions required in steps 1 and 2 are combined in a DOS level executable file called down_cam.exe. You may find this file in the TABLE subdirectory of your *Mx4 utilities diskette*. Alternatively, you may use the *Tables* option on the Mx4pro v4 for Win 95/NT to select master position spacing and table down load.

3) Depending on your application need, choose one of the following DSPL commands: `CAM, CAM_POS or CAM_PROBE`.

4) You may use one of the following DSPL commands to stop (disengage) a cam function: `CAM_OFF or CAM_OFF_ACC`.

The above four steps establish a command sequence for all cam applications.

How to Download a Table Along with Its Position Spacing

Steps 1 and 2 are combined in a single DOS executable called DOWN_CAM.EXE. This file is saved in the TABLE sub directory of the *Mx4 utilities diskette*. The syntax for this file is:

```
down_cam  table_name.dat  table_number  table_spacing  Mx4_card_address
```

where:

```
down_cam         ;name of the executable file
table_name       ;name of the ASCII table containing gear ratios
table_spacing    ;value specifying the master's position space
                 ;between adjacent gear ratios of the cam table
table_number     ;either 1 or 2, selecting one of the two tables
Mx4_card_address ;segment address for the Mx4 card
```

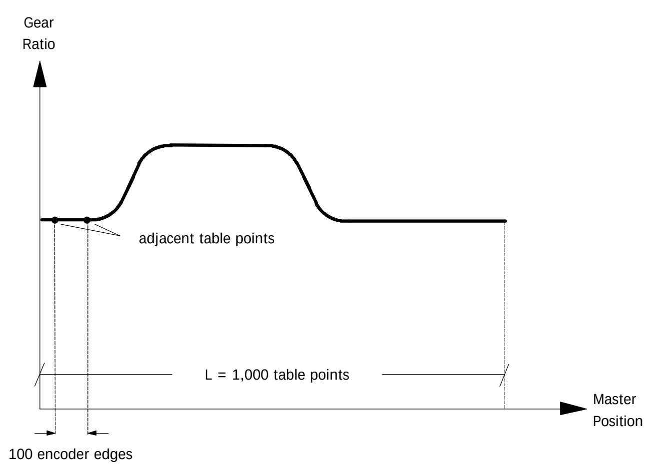
For example,

```
down_cam   tab.dat 2 500 0xd0000
```

means download ASCII file `TAB.DAT` to table 2 and use table position spacing of 500 for an Mx4 card located at segment address 0xd0000 (see Chapter 2 of the *Mx4 User's Guide* for hardware address settings).

Example

In a two-axis application axis 2 is the master and axis 1 is the slave. In this application the master must run at a constant speed of 10 counts/200 μsec. The slave must follow the master over the cam profile to be down loaded to table 1 as illustrated below. The position spacing between two adjacent points (gear ratios) of the cam table is 100 and the table length is 1000. (this means that there are 1000 gear ratios stored in the table) Write a DSPL program that puts the master and slaves in a cam relationship only when the master's position exceeds 200,000 counts.



Steps 1 and 2

Following the command sequence described earlier in this section, use DOS executable DOWN_CAM.EXE to download the cam table and table spacing value:

```
down_cam   ratio.dat 1 100 0xd0000
```

The following describes the DSPL program for this application:

```
PLC_PROGRAM:

 var1=0                  ;VAR1 is the initialization procedure flag
 run_m_program(INIT)     ;starts running the initialization program
 run_m_program (CAM_EX1) ;starts running the CAM_EX1 program

end


INIT:

 maxacc(0x3,0.1,0.1)     ;sets the maximum acc. for axes 1 & 2
 pos_preset(0x3,0,0)     ;presets the position of axes 1 & 2 to 0

 ctrl(0x3,0,28000,5000,1600,0,28000,5000,1600)

                         ;sets control law parameters for axes 1 & 2
   var1=1                ;initialization procedure has finished

end


CAM_EX1:

 wait_until(var1==1)     ;waits until the initialization finishes
 cam_tsize(1,1000)       ;sets the length of cam table 1 to 1000
 cam_pos(0x2,0x1,1,200000);engages CAM when the position of the master
                         ;axis exceeds 200,000 counts
   velmode (0x2,10)      ;runs axis 2 (master) in velocity mode

end
```

Use of Multiple Mx4 Cards in Cam Master/Slaving

Applications requiring more than three slaves need multiple Mx4 cards. The figure below illustrates the hardware diagram of a multi-card operation.

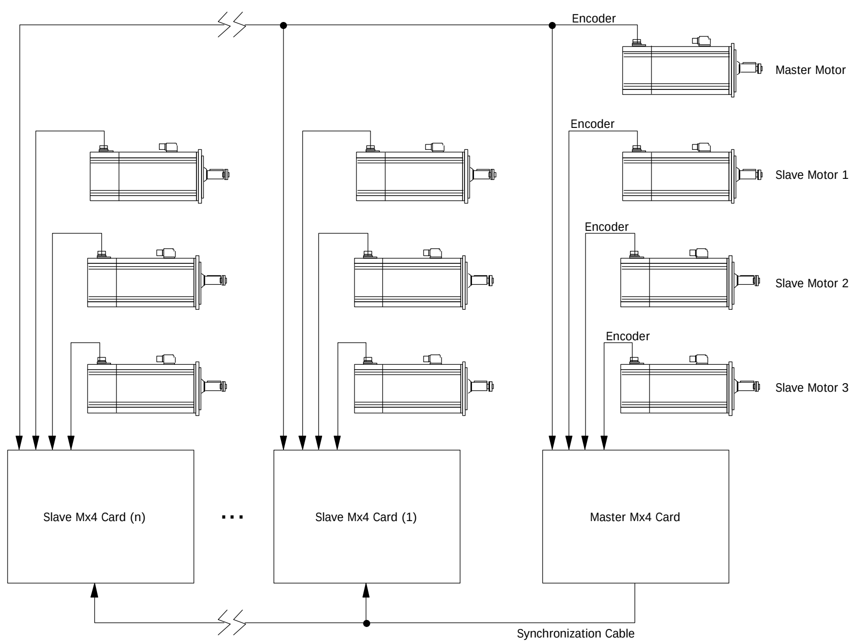


Figure: Multiple Mx4 Cards in Cam Master/Slaving

The position of the master position is used by the first axis of each Mx4 card. Therefore each card can only accept three slaves.

Hardware Settings for Multi-Card Cam Operation

Daisy-chaining several Mx4 boards and proper jumper settings for their synchronization is described in the *Mx4 User's Guide*, *Installing Your Mx4 Hardware*.

Software Commands for Multi-Card Cam Operation

The only difference between multiple- and single-card cam operations is that in multi-card operation, you must let a slave Mx4 card know that it has been selected as a slave. The master Mx4 card does not need to be notified!

On a slave card, the DSPL command that needs to precede those listed for a single card cam application (see Example 1) is:

```
SYNC
```

Note 1: The DSPL command `sync` ***must*** precede those listed in the first example.

Note 2: The above DSPL command `sync` is only ***required to run on a slave*** Mx4 card.

Cam Operation with Dynamic Error Correction on Slaves

Industrial applications such as flying shear with mark registration or synchronous cutting require frequent error correction. These cyclic motions are similar to those described in the previous two examples. The only difference is that the slave position must be corrected once every master cycle.

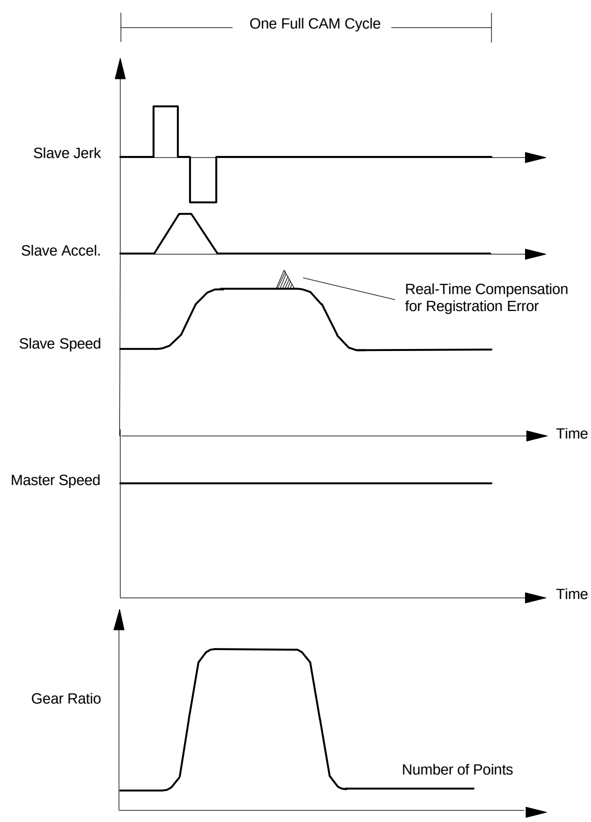


Figure: Master/Slave Cam Profile

The registration error (measured in real time by the DSPL) is used as the relative target position with instruction `REL_AXMOVE_SLAVE`. This command compensates for any slave position retardation.

Example

Consider Example 1 in a cyclic operation. This example uses the DSPL language and does not involve the host computer. The cutting error is defined as:

Cutting Error = (position of slave index marker) **-** (position of slave at the registration mark)

This value can be calculated in real time by the DSPL program and used as position argument with `REL_AXMOVE_SLAVE`. The command `REL_AXMOVE_SLAVE` superimposes a relative trapezoidal move on the top of the slave's motion. Therefore, it adds to slave position at a specified relative velocity and acceleration. In flying shear application, this compensation is done when the knife (slave) is disengaged. This way, during the next cycle, by the time the knife is engaged again, the slave has already recovered the error.

A DSPL Program Example

In the following example, axis 1 is master and axis 2 is slave. The cam table, 'RATIO.DAT' consisting 1000 gear ratios has already been downloaded to cam table 1 location via DOS command line:

```
down_cam   ration.dat  1 500 0xd0000
```

This means the master position spacing between adjacent gear ratios in cam table is equal to 500, and the Mx4 card is in address location 0xd0000.

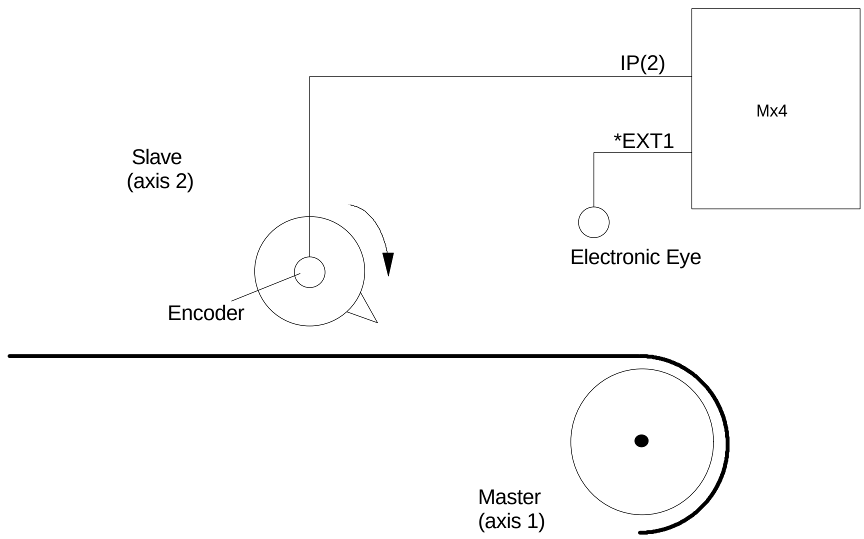


Figure: Flying Shear With Mark Registration

Figure shows that the registering electronic eye is connected to the probe signal (*EXT1) and index pulse of the knife (slave) registers the slave location. Enabling the probe interrupt will capture the position of all four axes upon the falling edge of *EXT1. Enabling the index pulse interrupt will capture the position of all four axes upon the rising edge of, IP(2). Upon the recipt of one of the two interrupts the index and probe positions are captured. Clearly, one of the interrupts may occur earlier than the other. The program waits until both interrupts within a single move cycle are received. `VAR5` calculates the distance between the positions of slave at the times of the two interrupts. This distance is used as a relative position in conjunction with `REL_AXMOVE_SLAVE` command to advance the motion of slave.

The following DSPL program implements the “flying shear” application.

```
PLC_PROGRAM:

   var1=0                                  ;VAR1 is the initialization procedure flag
   run_m_program(INIT)                     ;starts running the initialization program
   run_m_program(CAM_EX3)                  ;starts running the CAM_EX3 program

end

INIT:

   maxacc(0x3,0.1,0.1)                     ;sets the maximum acc. for axes 1 and 2
   pos_preset(0x3,0,0)                     ;presets the position of axes 1 and 2 to 0

   ctrl(0x3,0,28000,5000,1600,0,28000,5000,1600)

                                           ;sets control parameters for axes 1 and 2
   en_probe(1,1,0)                         ;enables probe 1 interrupt
   en_index(2)                             ;enables index pulse interrupt for axis 2
   var1=1                                  ;initialization procedure has finished

end

CAM_EX3:

   wait_until(var1==1)                     ;waits until the initialization finishes
   cam_tsize(1,1000)                       ;sets the length of cam table 1 to 1000
   cam(0x1,0x2,1)                          ;enables cam, axis 1 master, axis 2 slave
   velmode (0x1,5)                         ;runs axis 1 in velocity mode
   var2=0;                                 ;var2 is used as a control flag for the
                                           ;while loop

   while(var2==0)

     if ((probe_reg & 0x01) AND (index_reg & 0x02))
                                           ;checks for both interrupt conditions
       var3=probe_pos2                     ;stores the position of slave at the time
                                           ;the probe signal was set
       var4=index_pos2                     ;stores the position of slave at the time
                                           ;the index pulse was set
       var5=var4-var3                      ;computes the shift of slave position
       rel_axmove_slave(0x2,1.5,var5,20)   ;adjusts the position of slave
       int_reg_clr(0x09,0x02,0x01)         ;clears probe_reg and index_reg
       en_probe(1,1,0)                     ;enables probe 1 interrupt
       en_index(2)                         ;enables index pulse interrupt for axis 2

      endif

   wend

end
```