Sample Applications

Overview

The SSC provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the SSC memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed.

In addition to standard motion commands, the SSC provides commands that allow the SSC to make its own decisions. These commands include jumps, repeat loops, and subroutines.

For greater programming flexibility, the SSC provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss aspects of creating applications programs.

Example Applications

WIRE CUTTER

An operator activates a start switch. This causes an actuator/motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allow 100 ms for the cutting cycle to complete.

Suppose that the actuator has a 2 turn per inch screw. Also assume that the encoder resolution is 1000 lines per revolution. One inch of travel equals:

(2 rev/inch) * (1000 counts/rev) = 2000 counts/inch

To set up, use Controller Setup. Set the scaling for 2000 pulses per unit and the user unit to inches.

The input signal may be applied to input 1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Fig. 6.1.

EXAMPLE APPLICATIONS

The program starts at a state that we define as START. Here the controller waits for the input pulse on Input1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to START for a new cycle.

INSTRUCTION	INTERPRETATION
#Insert label #START.	Program Flow. Label
Wait for Input #1 to go LOW	Program Flow. Wait for Condition
Single Axis Move	
Axis X, incremental 10 in., etc.	Motion. Single Axis Move Set up for 10 inch move @ speed of 5 in/sec., accel of 10 in/sec.2
Wait for x axis to complete motion	Program Flow. Wait for Condition
Set Output #1 On	I/O. Output
Wait 20 ms	Program Flow. Wait
Set Output #1 Off	I/O. Output
Wait 80 ms	Program Flow. Wait
Jump to START	Program Flow. Jump

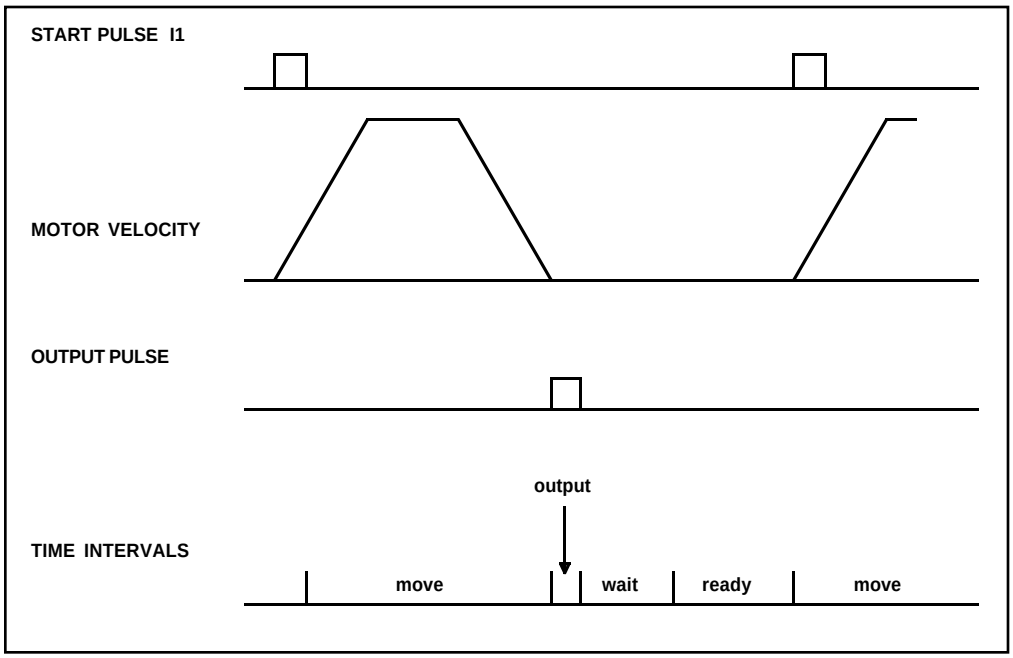


Figure 6.1 - Motor Velocity and the Associated input/output signals

X-Y TABLE CONTROLLER

An X-Y-Z system must cut the pattern shown in Fig. 6.2. The X-Y table moves the plate while the Z-axis raises and lowers the cutting tool.

Cutting must be performed at one inch per second. Non-cutting moves should be performed at 5 inches per second. The acceleration rate is 0.1 g.

The motion starts at point A, with the Z-axis raised. An X-Y motion to point B is followed by lowering the Z-axis and performing a cut along the circle. Once the circular motion is completed, the Z-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution or 4,000 counts per revolution in quadrature. This results in the relationship:

1 inch = 40,000 counts

an acceleration rate of 0.1g equals

0.1g = 38.6 in/s2

Note that the circular path has a radius of 2", and the motion starts at the angle of 270˚ and traverses 360˚ in the CW (negative direction). Such a path is specified with the Motion. 2D Circular interpolation instruction.

Further assume that the Z must move 2" at a linear speed of 2" per second. The required motion is performed by the following instructions:

INSTRUCTION	INTERPRETATION
#Insert Label START	Program Flow.....Label
#XY Linear Interpolation, Incremental Axis X 4 in, Axis Y 4 in, speed 5 in/s, accel 38.6 in/s2	Motion.....Linear Interpolation
#Single Axis Move Axis Z, -2 in., speed 2in/s, accel 38.6 in/s2	Motion.....Single Axis Move
# Wait for z-axis motion complete	Program Flow.....Wait for Condition
#XY Circular Motion, Clockwise Radius 2in., Start 270, Sweep 360, Speed 1in/s, accel 38.6 in/s2	Motion 2D Circular Motion
#Wait for y axis motion complete	Program Flow.....Wait for Condition

EXAMPLE APPLICATIONS

INSTRUCTION	INTERPRETATION
#Single Axis Move, Incremental Axis Z 2 in, Speed 2 in/s, accel 38.6 in/s2 Axis X 2 in, Speed 5.3 in/s, accel 38.6 in/s2	Motion.....Single Axis Move
# Single Axis Move Axis Z, -2., speed 2 in/s, accel 38.6 in/s2	Motion.....Single Axis Move
# wait for z-axis motion complete	Program Flow..... Wait fro Condition
#XY Circular Motion, Clockwise Radius 2 in., Start 270, Sweep 360, speed 1 in/s, accel 38.6 in/s2	Motion.....2D Circular Motion
# Wait for y axis motion complete	Program Flow.....Wait for Condition
#Single Axis Move Axis Z, -2 in., speed 2 in/s, accel 38.6 in/s2	
#Wait for z-axis motion complete	Program Flow.....Wait for Condition
#XY Linear Interpolation Axis X -9.3 in., Axis Y -4 in., speed 5 in/s, accel 38.6 in/s2	Motion.....Linear Interpolation
#Jump to START	Program Flow.....Jump

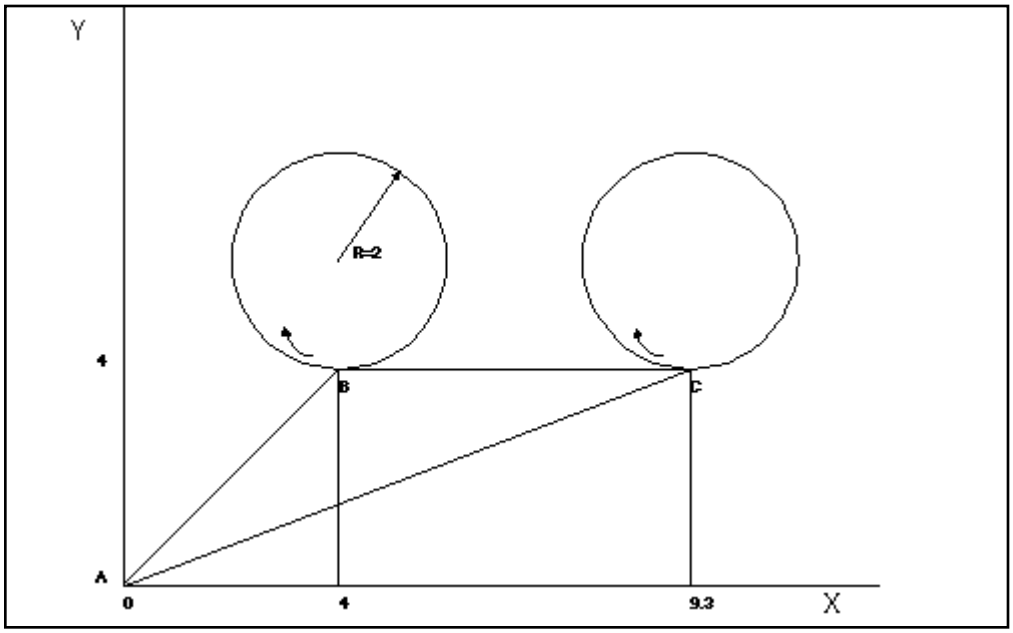


Figure 6.2 - Motor Velocity and the Associated input/output signals

Two-Letter Command Syntax

7

Introduction

The SSC provides over 100 two-letter commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter.

The SSC two-letter instruction set is BASIC-like. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion.

Commands can be sent "live" over the serial port for immediate execution by the SSC, or an entire group of commands can be downloaded into the SSC memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter 9.

This section describes the SSC instruction set and syntax. A summary of commands as well as a complete listing of all SSC instructions is included in the two-letter Command Reference chapter.

Command Syntax

SSC instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <enter> is used to terminate the instruction for processing by the SSC command interpreter. Note: If you are using a Tol-O-Motion's SSC terminal window, commands will not be processed until an <enter> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <enter> command.

IMPORTANT: All SSC commands must be sent in upper case.

For example, the command

INSTRUCTION	INTERPRETATION
PR 4000 <enter>	Position relative

PR is the two character instruction for position relative. 4000 is the argument which represents the required position value in counts. The

COMMAND SYNTAX

<enter> terminates the instruction. The space between PR and 4000 is optional.

For specifying data for the X,Y,Z and W axes, commas are used to separate the axes. If no data is specified for an axis, a comma is still needed as shown in the examples below. If no data is specified for an axis, the previous value is maintained. The space between the data and instruction is optional.

INSTRUCTION	INTERPRETATION
PR 500, 725, 300, 1000	Specify position relative 500 counts on X, 725 counts on Y, 300 counts on Z and 1000 counts on W

The SSC provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as X,Y,Z or W. An equals sign is used to assign data to that axis. For example:

INSTRUCTION	INTERPRETATION
PRX=1000	Specify a position relative movement for the X axis of 1000
ACY=200000	Specify acceleration for the Y axis as 200000

Instead of data, some commands request action to occur on an axis or group of axes. For example, ST XY stops motion on both the X and Y axes. Commas are not required in this case since the particular axis is specified by the appropriate letter X Y Z or W. If no parameters follow the instruction, action will take place on all axes. Here are some examples of syntax for requesting action:

INSTRUCTION	INTERPRETATION
BG X	Begin X only
BG Y	Begin Y only
BG XYZW	Begin all axes
BG YW	Begin Y and W only
BG	Begin all axes

COORDINATED MOTION WITH MORE THAN 1 AXIS

When requesting action for coordinated motion, the letter S is used to specify the coordinated motion. For example:

INSTRUCTION	INTERPRETATION
BG S	Begin coordinated sequence
BG SW	Begin coordinated sequence and W axis

PROGRAM SYNTAX

Chapter 8 explains how to write and execute motion control programs in 2-letter command format.

Controller Response to DATA

The SSC returns a : for valid commands.
The SSC returns a ? for invalid commands.
For example, if the command BG is sent in lower case, the SSC will return a ?.

INSTRUCTION	INTERPRETATION
:bg <enter>	invalid command, lower case
?	SSC returns a ?

When the controller receives an invalid command the user can request the error code. The error code will specify the reason for the invalid command response. To request the error code type the command: TC1 For example:

INSTRUCTION	INTERPRETATION
?TC1 <enter>	Tell Code command
1 Unrecognized command	Returned response

There are many reasons for receiving an invalid command response. The most common reasons are: unrecognized command (such as typographical entry or lower case), command given at improper time (such as during motion), or a command out of range (such as exceeding maximum speed). A complete list of all error codes can be found with the description of the TC command in the Command Reference, Chapter 12.

Interrogating the Controller

INTERROGATION COMMANDS

The SSC has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 12 for the two-letter Command Reference.

INTERROGATING THE CONTROLLER

Summary of Interrogation Commands

RP	Report Command Position
RL	Report Latch
^R ^V	Firmware Revision Information
SC	Stop Code
TB	Tell Status
TC	Tell Error Code
TD	Tell Dual Encoder
TE	Tell Error
TI	Tell Input
TP	Tell Position
TR	Trace
TS	Tell Switches
TT	Tell Torque
TV	Tell Velocity

For example, the following example illustrates how to display the current position of the X axis:

INSTRUCTION	INTERPRETATION
TP X <enter>	Tell position X
0000000000	Controllers Response
TP XY <enter>	Tell position X and Y
0000000000,0000000000	Controllers Response

ADDITIONAL INTERROGATION METHODS.

Most commands can be interrogated by using a question mark. For information specific to a particular axis, type the command followed by a ? for each axis requested.

INSTRUCTION	INTERPRETATION
PR 1000	Specify X only as 1000
PR ,2000	Specify Y only as 2000
PR ,,3000	Specify Z only as 3000
PR ,,,4000	Specify W only as 4000
PR 2000,4000,6000,8000	Specify X Y Z and W
PR ,8000,,9000	Specify Y and W only
PR ?,?,?,?	Request X,Y,Z,W values
PR ,?	Request Y value only

The controller can also be interrogated with operands.

Most SSC commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid SSC expressions. For example, to display the value of an operand, the user could use the command:

INSTRUCTION	INTERPRETATION
MG 'operand'	where 'operand' is a valid SSC operand

All of the command operands begin with the underscore character (_). For example, the value of the current position on the X axis can be assigned to the variable, V, with the command:

INSTRUCTION
V=_TPX

The Command Reference denotes all commands which have an equivalent operand as "Used as an Operand". For further information, see description of operands in Chapter 8.

Command Summary

For a complete command summary, see Chapter 12, Command Reference and the two-letter Command Summary in the Appendix.

Notes:

Programming Motion
With Two-Letter Command Syntax

Overview

The SSC can be commanded to do the following modes of motion: Absolute and relative independent positioning, jogging, linear interpolation (up to 4 axes), linear and circular interpolation (2 axes with 3rd axis of tangent motion), electronic gearing, electronic cam motion and contouring. These modes are discussed in the following sections.

The SSC1 is a single axis controller and uses X-axis motion only. Likewise, the SSC2 uses X and Y, the SSC3 uses X,Y and Z, and the SSC4 uses X,Y,Z and W.

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the SSC profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is sent by the SSC profiler. **Note: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.**

The Begin (BG) command can be issued for all axes either simultaneously or independently. XYZ or W axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The speed (SP) and the acceleration (AC) can be changed at any time during motion, however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. **Remember, motion is complete when the profiler is finished, not when the actual motor is in position.** The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

INDEPENDENT AXIS POSITIONING

An incremental position movement (IP) may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the new end position. The IP command does not require a begin. Note: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

COMMAND SUMMARY - INDEPENDENT AXIS

COMMAND	DESCRIPTION
PR X, Y, Z, W	Specifies relative distance
PA x, y, z, w	Specifies absolute position
SP x, y, z, w	Specifies slew speed
AC x, y, z, w	Specifies acceleration rate
DC x, y, z, w	Specifies deceleration rate
BG X, Y, Z, W	Starts motion
ST XYZW	Stops before end of move
IP x, y, z, w	Changes target position
IT x, y, z, w	Time constant for independent motion smoothing
AM XYZW	Trippoint for profiler complete
MC XYZW	Trippoint for "in position"

The lower case specifiers (x,y,z,w) represent position values for each axis.

OPERAND SUMMARY - INDEPENDENT AXIS

OPPERAND	DESCRIPTION
_ACx	Return acceleration rate for tha axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Return speed for the axis specified by 'x'
_PAx	Return current destination if 'x' axis is moving, otherwise returns tthe current command postion if in a move
_PRx	Returns the current incremental distance specified for the 'x' axis

INDEPENDENT POSITIONING EXAMPLES

Absolute Position Movement

INSTRUCTION	INTERPRETATION
PA 10000,20000	Specify absolute X,Y position
AC 1000000,1000000	Acceleration for X,Y
DC 1000000,1000000	Deceleration for X,Y
SP 50000,30000	Speeds for X,Y
BG XY	Begin motion

Multiple Move Sequence

REQUIRED MOTION PROFILES:

INSTRUCTION	INTERPRETATION
X-Axis Position	1000 counts
20000 count/sec	Speed
500000 counts/sec2	Acceleration
Y-Axis Position	300 counts
10000 count/sec	Speed
500000 counts/sec2	Acceleration
Z-Axis Position	25 counts
2500 counts/sec	Speed
500000 counts/sec	Acceleration

This example will specify a relative position movement on X, Y and Z axes. The movement on each axis will be separated by 20 msec. Fig. 8.1 shows the velocity profiles for the X,Y and Z axis.

INSTRUCTION	INTERPRETATION
#A	BEGIN PROGRAM
PR 1000,300,25	Specify relative position movement of 1000, 300 and 25 counts for X,Y and Z axes.
SP 20000,10000,2500	Specify speed of 20000, 10000, and 2500 counts / sec
AC 500000,500000,500000	Specify acceleration of 500000 counts / sec2 for all axes
DC 500000,500000,500000	Specify deceleration of 500000 counts / sec2 for all axes
BG X	Begin motion on the X axis
WT 20	Wait 20 msec
BG Y	Begin motion on the Y axis
WT 20	Wait 20 msec
BG Z	Begin motion on Z axis
EN	End Program

INDEPENDENT AXIS POSITIONING

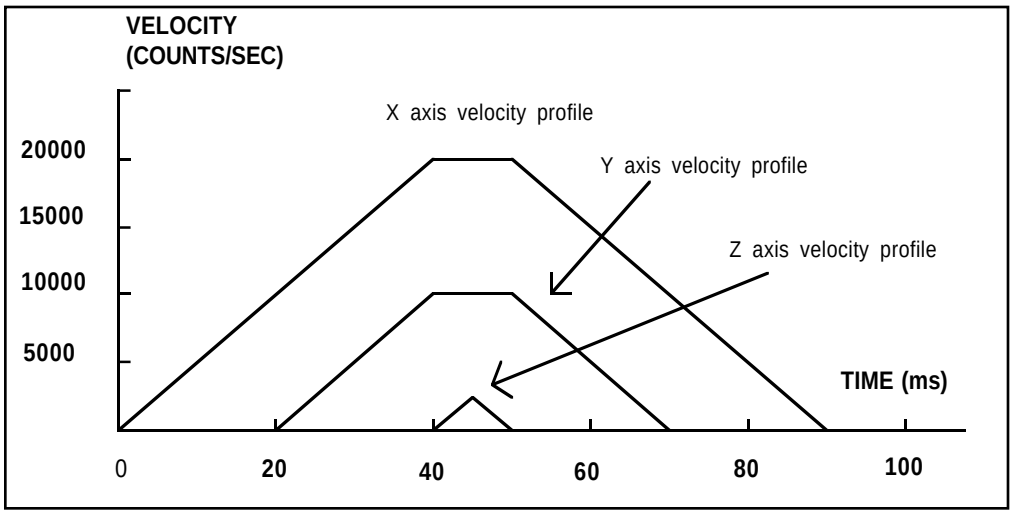


Figure 8.1 - Velocity Profiles of XYZ

Notes on Fig. 8.1: The X and Y axis have a 'trapezoidal' velocity profile, while the Z axis has a 'triangular' velocity profile. The X and Y axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The Z axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, hence, the slope of the rising and falling edges of all 3 velocity profiles are the same.

Independent Jogging

The jog mode of motion allows the user to change speed, direction and acceleration during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate for each axis. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) command is issued. If the jog speed is changed during motion, the controller will make an accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the

current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The SSC converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

COMMAND SUMMARY - JOGGING

COMMAND	DESCRIPTION
AC x,y,z,w	Specifies the acceleration rate
BG X,Y,Z,W	Begins motion
DC x,y,z,w	Specifies deceleration rate
IP x,y,z,w	Increments position instantly
IT x,y,z,w	Time constant for independent motion smoothing
JG +/- x,y,z,w	specifies jog speed and direction
ST XYZW	stops motion

Parameters can be set with individual axes specifiers such as JGY=2000 (set jog speed for Y axis to 2000)

OPERAND SUMMARY - INDEPENDENT AXIS

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Return the jog speed for the axis specified by 'x'
_TVx	Returns the actual velocity of the axis specified by 'x' (averaged over .25 sec)

INDEPENDENT JOGGING

JOG EXAMPLES

Jog in X Only

Jog X motor at 50000 count/s. After X motor is at its jog speed, begin jogging Z in reverse direction at 25000 count/s.

INSTRUCTION	INTERPRETATION
#A	
AC 20000,,20000	Specify X,Z acceleration of 20000 cts/sec2
DC 20000,,20000	Specify X,Z deceleration of 20000 cts/sec2
JG 50000,,-25000	Specify jog speed and direction for X and Z axis
BG X	Begin X motion
AS X	Wait until X is at speed
BG Z	Begin Z motion
EN	

Example - Joystick Jogging

The jog speed can also be changed using an analog input such as a joystick. Assume that for a 10 Volt input the speed must be 50000 counts/sec.

INSTRUCTION	INTERPRETATION
#JOY	Label
JG0	Set in Jog Mode
BGX	Begin motion
#B	Label for Loop
V1 = @AN[1]	Read analog input
VEL = V1*50000/2047	Compute speed
JG VEL	Change JG speed
JP #B	Loop

Linear Interpolation Mode

The SSC provides a linear interpolation mode for 2 or more axes. In linear interpolation mode, motion between the axes is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM command selects the Linear Interpolation mode and axes for interpolation. For example, LM YZ selects only the Y and Z axes for linear interpolation.

When using the linear interpolation mode, the LM command only needs to be specified once unless the axes for linear interpolation change.

SPECIFYING LINEAR SEGMENTS

The command LI x,y,z,w specifies the incremental move distance for each axis. This means motion is prescribed with respect to the current axis position. Up to 511 incremental move segments may be given prior to the Begin Sequence (BGS) command. Once motion has begun, additional LI segments may be sent to the controller.

The clear sequence (CS) command can be used to remove LI segments stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB. The command, ST, causes a decelerated stop. The command, AB, causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The Linear End (LE) command must be used to specify the end of a linear move sequence. This command tells the controller to decelerate to a stop following the last LI command. If an LE command is not given, an Abort AB1 must be used to abort the motion sequence.

It is the responsibility of the user to keep enough LI segments in the SSC sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent at PC bus speeds.

The instruction _CS returns the segment counter. As the segments are processed, _CS increases, starting at zero. This function allows the host computer to determine which segment is being processed.

LINEAR INTERPOLATION MODE

SPECIFYING VECTOR ACCELERATION, DECELERATION AND SPEED:

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration. The SSC computes the vector speed based on the axes specified in the LM mode. For example, LM XYZ designates linear interpolation for the X,Y and Z axes. The vector speed for this example would be computed using the equation:

$VS^2=XS^2+YS^2+ZS^2$, where XS, YS and ZS are the speed of the X,Y and Z axes.

The controller always uses the axis specifications from LM, not LI, to compute the speed.

In cases where the acceleration causes the system to 'jerk', the SSC provides a vector motion smoothing function. VT is used to set the S-curve smoothing constant for coordinated moves.

ADDITIONAL COMMANDS

The SSC provides commands for additional control of vector motion and program control. Note: Many of the commands used in Linear Interpolation motion also applies to Vector motion described in the next section.

TRIPPOINTS

The command AV n is the 'After Vector' trippoint, which halts program execution until the vector distance of n has been reached.

In this example, the XY system is required to perform a 90° turn. In order to slow the speed around the corner, we use the AV 4000 trippoint, which slows the speed to 1000 count/s. Once the motors reach the corner, the speed is increased back to 4000 cts / s.

TRIPPOINT EXAMPLE

INSTRUCTION	INTERPRETATION
#LMOVE	Label
DP 0,0	Define position of X and Y axes to be 0
LMXY	Define linear mode between X and Y axes.
LI 5000,0	Specify first linear segment
LI 0,5000	Specify second linear segment
LE	End linear segments
VS 4000	Specify vector speed
BGS	Begin motion sequence

AV 4000	Set trippoint to wait until vector distance of 4000 is reached
VS 1000	Change vector speed
AV 5000	Set trippoint to wait until vector distance of 5000 is reached
VS 4000	Change vector speed
EN	Program end

SPECIFYING VECTOR SPEED FOR EACH SEGMENT

The instruction VS has an immediate effect and, therefore, must be given at the required time. In some applications, such as CNC, it is necessary to attach various speeds to different motion segments. This can be done by the instruction.

LI x,y,z,w < n

This instruction attaches the vector speed, n, to the motion segment LI. As a consequence, the program #LMOVE can be written in the alternative form:

VECTOR SPEED EXAMPLE

INSTRUCTION	INTERPRETATION
#ALT	Label for alternative program
DP 0,0	Define Position of X and Y axis to be 0
LMXY	Define linear mode between X and Y axes.
LI 4000,0 <4000	Specify first linear segment with a vector speed of 4000
LI 1000,0 < 1000	Specify second linear segment with a vector speed of 1000
LI 0,5000 < 4000	Specify third linear segment with a vector speed of 4000
LE	End linear segments
BGS	Begin motion sequence
EN	Program end

CHANGING FEEDRATE:

The command VR n allows the feedrate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feedrate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

LINEAR INTERPOLATION MODE

COMMAND SUMMARY - LINEAR INTERPOLATION

COMMAND	DESCRIPTION
LM xyzw	Specify axis for linear interpolation
LM?	Returns number of available spaces for linear segments in the XX sequence buffer. Zero means buffer full. 512 means buffer empty
LI x,y,z,w < n	Specify incremental distance relative to current position, and assign vector speed n.
VSn	Specify vector speed
VAn	Specify vector acceleration
VDn	Specify vector deceleration
VRn	Specify the vector speed ratio
BGS	Begin linear sequence
CS	Clear Sequence
LE	Linear End- Required at end of LI sequence
LE?	Returns the length of the vector (resets after 22147483647)
AMS	Trippoint for after sequence complete
AV n	Trippoint for after relative vector distance, n
VT	S curve smoothing constant for vector moves

OPERAND SUMMARY - LINEAR INTERPOLATION

OPERAND	DESCRIPTION
_AV	Return distance traveled
_CS	Segment counter - returns number of the segment in the sequence. Starting at zero
_LE	Returns the length of vector (resets after 2147483647)
_LM	Returns number of available spaces for linear segments in sequence buffer Zero means buffer full. 512 means buffer empty
_VPm	Return the absolute coordinate of the last data point along the trajectory. (m= X.Y.Z or W)

To illustrate the ability to interrogate the motion status, consider the first motion segment of our example, #LMOVE, where the X axis moves toward the point X=5000. Suppose that when X=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS, _VPX and _VPY will be zero.

Now suppose that the interrogation is repeated at the second segment when Y=2000. The value of _AV at this point is 7000, _CS equals 1, _VPX=5000 and _VPY=0.

LINEAR INTERPOLATION EXAMPLE

Linear Move

Make a coordinated linear move in the ZW plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec2.

INSTRUCTION	INTERPRETATION
#TEST	Label
LM ZW	Specify axes for linear interpolation
LI,,40000,30000	Specify ZW distances
LE	Specify end move
VS 100000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
BGS	Begin sequence
AMS	After motion sequence ends
EN	End program

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VZ and VW. The axis speeds are determined by the SSC from:

The resulting profile is shown in Figure 8.2.

LINEAR INTERPOLATION MODE

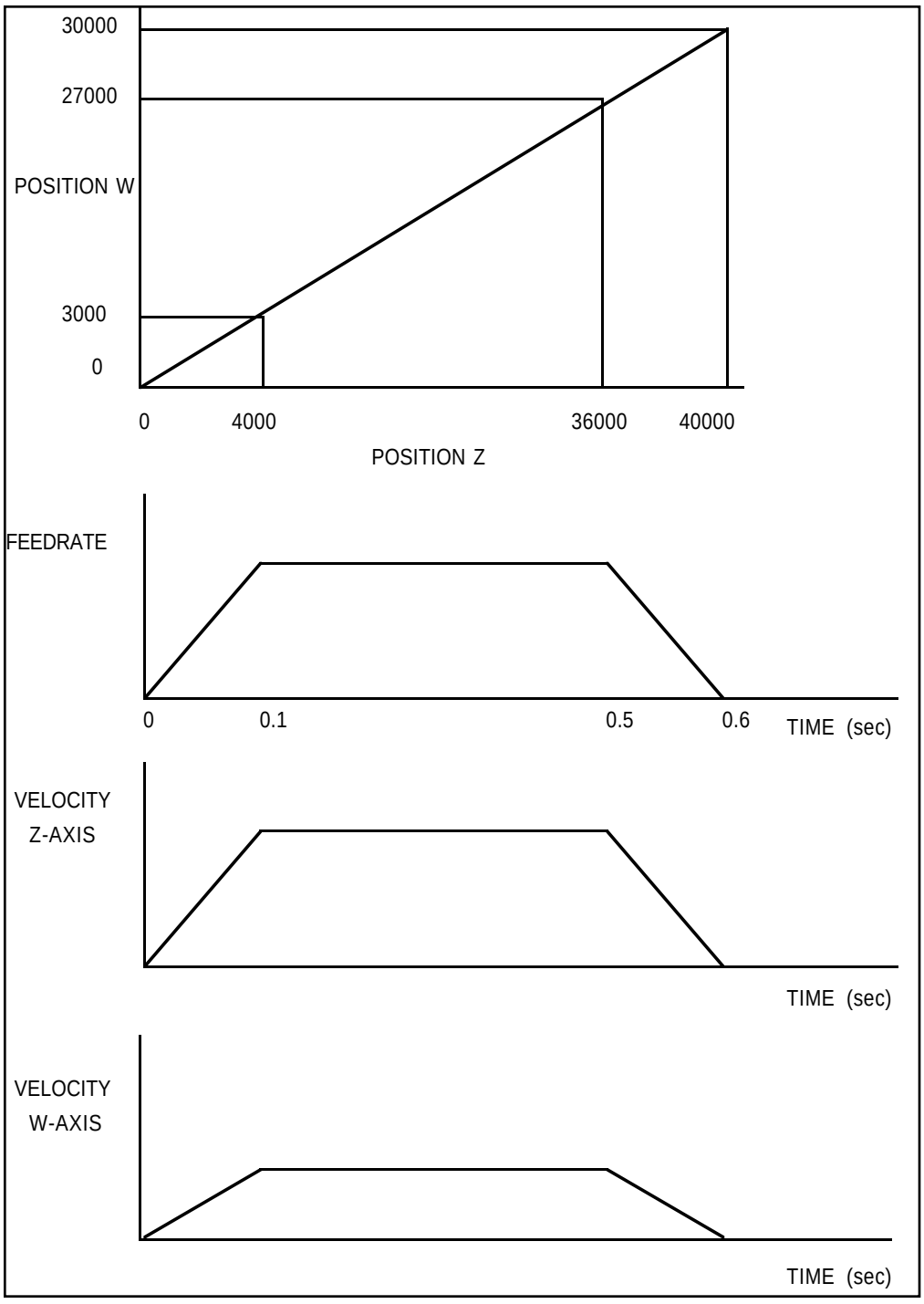


Figure 8.2 - Linear Interpolation

Multiple Moves

This example makes a coordinated linear move in the XY plane. The Arrays VX and VY are used to store 750 incremental distances which are filled by the program #LOAD.

INSTRUCTION	INTERPRETATION
#LOAD	Load Program
DM VX [750],VY [750]	Define Array
COUNT=0	Initialize Counter
N=0	Initialize position increment
#LOOP	LOOP
VX [COUNT]=N	Fill Array VX
VY [COUNT]=N	Fill Array VY
N=N+10	Increment position
COUNT=COUNT+1	Increment counter
JP #LOOP,COUNT<750	Loop if array not full
#A	Label
LM XY	Specify linear mode for XY
COUNT=0	Initialize array counter
#LOOP2;JP#LOOP2,_LM=0	If sequence buffer full, wait
JS#C,COUNT=500	Begin motion on 500th segment
LI VX[COUNT],VY[COUNT]	Specify linear segment
COUNT=COUNT+1	Increment array counter
JP #LOOP2,COUNT<750	Repeat until array done
LE	End Linear Move
AMS	After Move sequence done
MG “DONE”	Send Message
EN	End program
#C;BGS;EN	Begin Motion Subroutine

Vector Mode: Linear and Circular Interpolation Motion

The SSC allows a long 2-D path consisting of linear and arc segments to be prescribed. Motion along the path is continuous at the prescribed vector speed even at transitions between linear and circular segments. The SSC performs all the complex computations of linear and circular interpolation, freeing the host PC from this time intensive task.

The coordinated motion mode is similar to the linear interpolation mode. Any pair of two axes may be selected for coordinated motion consisting of linear and circular segments. In addition, a third axis can be controlled

VECTOR MODE: LINEAR AND CIRCULAR INTERPOLATION MOTION

such that it remains tangent to the motion of the selected pair of axes. Note that only one pair of axes can be specified for coordinated motion at any given time.

The command VM m,n,p where 'm' and 'n' are the coordinated pair and p is the tangent axis (Note: the commas which separate m,n and p are not necessary). For example, VM XWZ selects the XW axes for coordinated motion and the Z-axis as the tangent.

SPECIFYING VECTOR SEGMENTS

The motion segments are described by two commands; VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence. Note: This 'local' definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.

The command, VP xy specifies the coordinates of the end points of the vector movement with respect to the starting point. The command, CR r,q,d define a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counter-clockwise (CCW) rotation is positive.

Up to 511 segments of CR or VP may be specified in a single sequence and must be ended with the command VE. The motion can be initiated with a Begin Sequence (BGS) command. Once motion starts, additional segments may be added.

The Clear Sequence (CS) command can be used to remove previous VP and CR commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. ST stops motion at the specified deceleration. AB1 aborts the motion instantaneously.

The Vector End (VE) command must be used to specify the end of the coordinated motion. This command requires the controller to decelerate to a stop following the last motion requirement. If a VE command is not given, an Abort (AB1) must be used to abort the coordinated motion sequence.

It is the responsibility of the user to keep enough motion segments in the SSC sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at PC bus speeds.

The operand _CS can be used to determine the value of the segment counter.

SPECIFYING VECTOR ACCELERATION, DECELERATION AND SPEED:

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration. The SSC computes the vector speed based on the two axes specified in the VM mode. For example, VM YZ designates vector mode for the Y and Z axes. The vector speed for this example would be computed using the equation:

$VS^2=YS^2+ZS^2$, where YS and ZS are the speed of the Y and Z axes.

In cases where the acceleration causes the system to 'jerk', the SSC provides a vector motion smoothing function. VT is used to set the S-curve smoothing constant for coordinated moves.

ADDITIONAL COMMANDS

The SSC provides commands for additional control of vector motion and program control. Note: Many of the commands used in Vector Mode motion also applies Linear Interpolation motion described in the previous section.

TRIPPOINTS

The command AV n is the 'After Vector' trippoint, which halts program execution until the vector distance of n has been reached.

Specifying Vector Speed for Each Segment

The vector speed may be specified by the immediate command VS. It can also be attached to a motion segment with the instructions

VP x,y, < n CR r,q,d < n

VECTOR MODE: LINEAR AND CIRCULAR INTERPOLATION MOTION

Both cases assign a vector speed of n count/s to the corresponding motion segment.

CHANGING FEEDRATE:

The command VR n allows the feedrate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feedrate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

COMPENSATING FOR DIFFERENCES IN ENCODER RESOLUTION:

By default, the SSC uses a scale factor of 1:1 for the encoder resolution when used in vector mode. If this is not the case, the command, ES can be used to scale the encoder counts. The ES command accepts two arguments which represent the number of counts for the two encoders used for vector motion. The smaller ratio of the two numbers will be multiplied by the higher resolution encoder. For more information, see ES command in Chapter 12, Command Summary.

TANGENT MOTION:

Several applications, such as cutting, require a third axis (i.e. a knife blade), to remain tangent to the coordinated motion path. To handle these applications, the SSC allows one axis to be specified as the tangent axis. The VM command provides parameter specifications for describing the coordinated axes and the tangent axis.

VM m,n,p m,n specifies coordinated axes p specifies tangent axis such as X,Y,Z,W p=N turns off tangent axis

Before the tangent mode can operate, it is necessary to assign an axis via the VM command and define its offset and scale factor via the TN m,n command. m defines the scale factor in counts/degree and n defines the tangent position that equals zero degrees in the coordinated motion plane. The _TN can be used to return the initial position of the tangent axis.

TANGENT MOTION EXAMPLE

XY Table Control

Assume an XY table with the Z-axis controlling a knife. The Z-axis has a 2000 quad counts/rev encoder and has been initialized after power-up to point the knife in the +Y direction. A 180° circular cut is desired, with a radius of 3000, center at the origin and a starting point at (3000,0). The motion is CCW, ending at (-3000,0). Note that the 0° position in the XY plane is in the +X direction. This corresponds to the position -500 in the Z-axis, and defines the offset. The motion has two parts. First, X,Y and Z are driven to the starting point, and later, the cut is performed. Assume that the knife is engaged with output bit 0.

INSTRUCTION	INTERPRETATION
#EXAMPLE	Example program
VM XYZ	XY coordinate with Z as tangent
TN 2000/360,-500	2000/360 counts/degree, position -500 is 0 degrees in XY plane
CR 3000,0,180	3000 count radius, start at 0 and go to 180 CCW
VE	End vector
CB0	Disengage knife
PA 3000,0,_TN	Move X and Y to starting position, move Z to initial tangent position
BG XYZ	Start the move to get into position
AM XYZ	When the move is complete
SB0	Engage knife
WT50	Wait 50 msec for the knife to engage
BGS	Do the circular cut
AMS	After the coordinated move is complete
CB0	Disengage knife
MG “ALL DONE”	
EN	End program

VECTOR MODE: LINEAR AND CIRCULAR INTERPOLATION MOTION

COMMAND SUMMARY - VECTOR MODE MOTION

COMMAND	DESCRIPTION
VM m,n, p	Specifies the axes for the planar motion where m and n represent the planar axes and p is the tangent axis
VP m,n	Define target coordinates of a straight line segment in a z-axis motion sequence.
CR r, q , ΔΘ	Specifies arc segment where r is the radius, q is the starting angle and ΔΘ is the travel angle. Positive direction is CCW
VS n	Specify vector speed or feedrate of a sequence
VA n	Specify vector acceleration along the sequence
VD n	Specify vector deceleration along the sequence
VR n	Specify vector speed ratio
BGS	Begin sequence
CS	Clear sequence
AV n	Trippoint after relative vector distance, n
AMS	Holds execution of next command until motion sequence is complete
ES m,n	Ellipse scale factor
VT	S curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in sequence buffer. Zero means buffer is full. 512 means buffer is empty

OPERAND SUMMARY - VECTOR MODE MOTION

OPERAND	DESCRIPTION
_VPM	The absolute coordinate of the axes at the last intersection along the sequence.
_AV	Distance traveled
_LM	Number of available spaces fir linear and circular segments in sequence buffer. Zero means buffer is full 512 means buffer is empty
_CS	Segment counter - number of the segment in the sequence, starting at zero

When AV is used as an operand, _AV returns the distance traveled along the sequence.

The operands _VPX and _VPY can be used to return the coordinates of the last point specified along the path.

VECTOR MODE EXAMPLE

Traverse the path shown in Fig. 8.3. Feedrate is 20000 counts/sec. Plane of motion is XY

INSTRUCTION	INTERPRETATION
VM XY	Specify motion plane
VS 20000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
VP -4000,0	Segment AB
CR 1500,270,-180	Segment BC
VP 0,3000	Segment CD
CR 1500,90,-180	Segment DA
VE	End of sequence
BGS	Begin Sequence

The resulting motion starts at the point A and moves toward points B, C, D, A. Suppose that we interrogate the controller when the motion is halfway between the points A and B.

The value of _AV is 2000

The value of _CS is 0

_VPX and _VPY contain the absolute coordinate of the point A

Suppose that the interrogation is repeated at a point, halfway between the points C and D.

The value of _AV is 4000+1500 * TC + 2000=10,712

The value of _CS is 2

_VPX,_VPY contain the coordinates of the point C

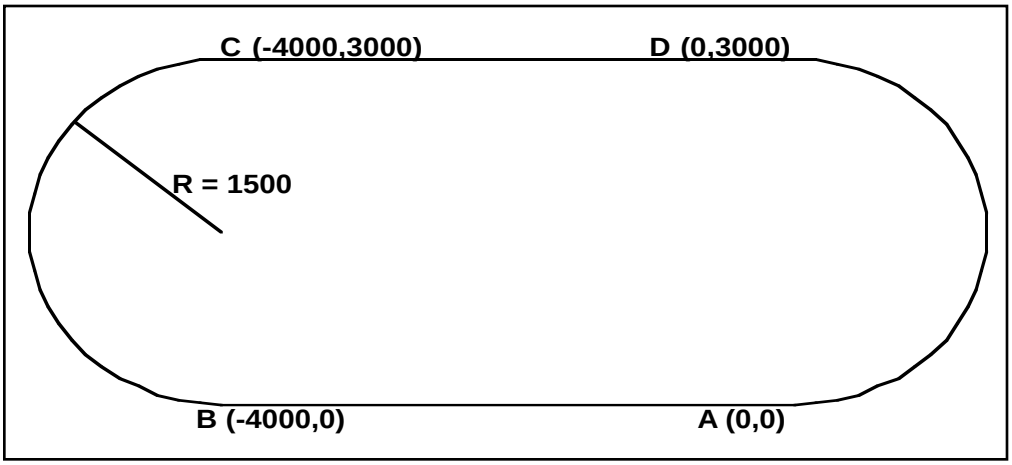


Figure 8.3 - The Required Path

Electronic Gearing

This mode allows up to 4 axes to be electronically geared to one master axis. The master may rotate in both directions and the geared axes will follow at the specified gear ratio. The gear ratio may be different for each axis and changed during motion.

The command GAX or GAY or GAZ or GAW specifies the master axis. There may only be one master. GR x,y,z,w specifies the gear ratios for the slaves where the ratio may be a number between +/-127.9999 with a fractional resolution of .0001. GR 0,0,0,0 turns off electronic gearing for any set of axes. A limit switch will also disable electronic gearing for that axis. GR causes the specified axes to be geared to the actual position of the master. The master axis is commanded with motion commands such as PR, PA or JG.

When the master axis is driven by the controller in the jog mode or an independent motion mode, it is possible to define the master as the command position of that axis, rather than the actual position. The designation of the commanded position master is by the letter, C. For example, GACX indicates that the gearing is the commanded position of X.

An alternative gearing method is to synchronize the slave motor to the commanded vector motion of several axes performed by GAS. For example, if the X and Y motor form a circular motion, the Z axis may move in proportion to the vector move. Similarly, if X,Y and Z perform a linear interpolation move, W can be geared to the vector move.

Electronic gearing allows the geared motor to perform a second independent or coordinated move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR, or JG, commands, or VP, or LI.

COMMAND SUMMARY - ELECTRONIC GEARING

COMMAND	DESCRIPTION
GA n	Specifies master axis for gearing where n= X,Y,Z, or W for main encoder as master
	n= XC,YC,ZC or WC for auxiliary encoders
	n= DX, CY,DZ or DW for auxiliary encoders
	n= S vector move as master
GR x,y,z,w	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis
MR x,y,z,w	Trippoint for reverse motion past specified value. Only one field may be used
MF x,y,z,w	Trippoint for reverse motion past specified value. Only one field may be used

OPERAND SUMMARY - ELECTRONIC GEARING

OPPERAND	DESCRIPTION
_GR x	Contains the value of gear ratio for axis 'x'

ELECTRONIC GEARING EXAMPLES

Simple Master Slave

Master axis moves 10000 counts at slew speed of 100000 counts/sec. Y is defined as the master. X,Z,W are geared to master at ratios of 5,-.5 and 10 respectively.

INSTRUCTION	INTERPRETATION
GAY	Specify master axes as Y
GR 5,,-.5,10	Set gear ratios
PR ,10000	Specify Y position
SP ,100000	Specify Y speed
BGY	Begin motion

Electronic Gearing

Objective: Run two geared motors at speeds of 1.132 and -0.045 times the speed of an external master. The master is driven at speeds between 0 and 1800 RPM (2000 counts/rev encoder).

Solution: Use a SSC controller, where the Z-axis is the master and X and Y are the geared axes.

INSTRUCTION	INTERPRETATION
MO Z	Turn Z off, for external master
GA Z	Specify master axis
GR 1.132,-.045	Specify gear ratios

Now suppose the gear ratio of the X-axis is to change on-the-fly to 2. This can be achieved by commanding:

INSTRUCTION	INTERPRETATION
GR 2	Specify gear ratio for X axis to be 2

In applications where both the master and the follower are controlled by the SSC controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

For example, assume that a gantry is driven by two axes, X,Y, on both sides. The X-axis is the master and the Y-axis is the follower. To synchronize Y with the commanded position of X, use the instructions:

INSTRUCTION	INTERPRETATION
GA XC	Specify master as commanded position of X
GR,1	Set gear ratio for Y as 1:1
PR 3000	Command X motion
BG X	Start motion on X axis

You may also perform profiled position corrections in the electronic gearing mode. Suppose, for example, that you need to advance the slave 10 counts. Simply command:

INSTRUCTION	INTERPRETATION
IP ,10	Specify an incremental position movement of 10 on Y axis.

Under these conditions, this IP command is equivalent to:

INSTRUCTION	INTERPRETATION
PR,10	Specify position relative movement of 10 on Y axis
BGY	Begin motion on Y axis

Often the correction is quite large. Such requirements are common when synchronizing cutting knives or conveyor belts.

Synchronize two conveyor belts with trapezoidal velocity correction.

INSTRUCTION	INTERPRETATION
GAX	Define master axis as X
GR,2	Set gear ratio 2:1 for Y
PR,300	Specify correction distance
SP,5000	Specify correction speed
AC,100000	Specify correction acceleration
DC,100000	Specify correction deceleration
BGY	Start correction

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of several axes of motion. Up to 3 axes can be slaved to one master axis. The master axis encoder must be input through a main encoder port.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes.

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis Y, when the master is X. Such a graphic relationship is shown in Figure 8.4.

ELECTRONIC CAM

STEP 1. SELECTING THE MASTER AXIS

The first step in the electronic cam mode is to select the master axis. This is done with the instruction:

INSTRUCTION

EAp where p = X,Y,Z,W

p is the selected master axis

STEP 2. SPECIFY THE MASTER CYCLE AND THE CHANGE IN THE SLAVE AXIS (ES).

In the electronic cam mode, the position of the master is always expressed modulo one cycle. In this example, the position of x is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both x and y are redefined as zero. To specify the master cycle and the slave cycle change, we use the instruction EM.

INSTRUCTION

EM x,y,z,w

where x,y,z,w specify the cycle of the master and the total change of the slaves over one cycle.

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instruction:

INSTRUCTION

EM 6000,1500

STEP 3. SPECIFY THE MASTER INTERVAL AND STARTING POINT.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 256 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

INSTRUCTION

EP m,n

where m is the interval width in counts, and n is the starting point.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

INSTRUCTION

EP 2000,0

STEP 4. SPECIFY THE SLAVE POSITIONS

Next, we specify the slave positions with the instruction

INSTRUCTION

ET[n]=x,y,z,w

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameters x,y,z,w indicate the corresponding slave position. For this example, the table may be specified by

INSTRUCTION

ET[0]=,0
ET[1]=,3000
ET[2]=,2250
ET[3]=,1500

This specifies the ECAM table.

STEP 5. ENABLE THE ECAM

To enable the ECAM mode, use the command

INSTRUCTION

EB n

where n=1 enables ECAM mode and n=0 disables ECAM mode.

STEP 6. ENGAGE THE SLAVE MOTION

To engage the slave motion, use the instruction

INSTRUCTION

EG x,y,z,w

where x,y,z,w are the master positions at which the corresponding slaves must be engaged.

If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

ELECTRONIC CAM

STEP 7. DISENGAGE THE SLAVE MOTION

To disengage the cam, use the command

INSTRUCTION

EQ x,y,z,w

where x,y,z,w are the corresponding slave axes are disengaged.

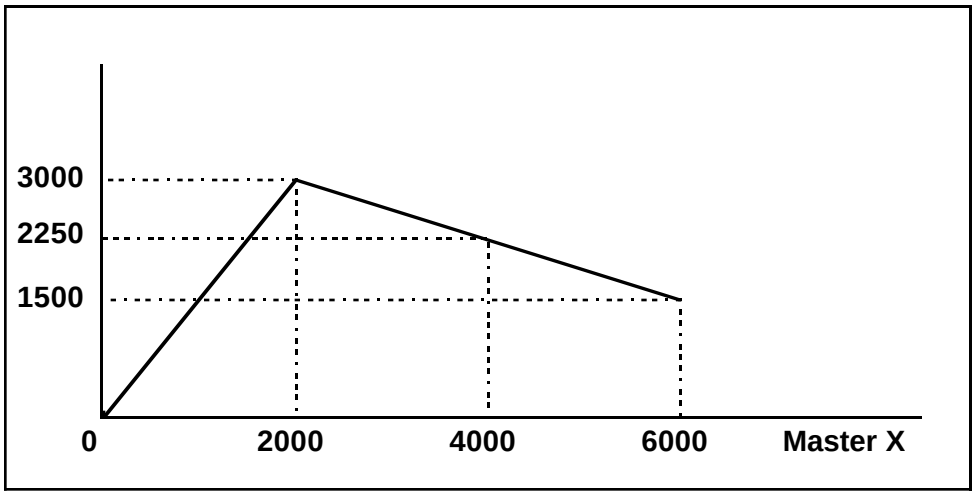


Figure 8.4 - Electronic Cam Example

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

Programmed start and stop can be used only when the master moves forward.

Example

To illustrate the complete process, consider the cam relationship described by the equation:

INSTRUCTION

Y = 0.5 * X + 100 sin (0.18*X) where X is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAX defines Y as the master axis. The cycle of the master is 2000. Over that cycle, X varies by 1000. This leads to the instruction EM 2000,1000.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is EP 20,0.

The following routine computes the table points. As the phase equals 0.18X and X varies in increments of 20, the phase varies by increments of 3.6°. The program then computes the values of Y according to the equation and assigns the values to the table with the instruction ET[N] = ,Y.

INSTRUCTION	INTERPRETATION
#SETUP	Label
EAX	Select X as master
EM 2000,1000	Cam cycles
EP 20,0	Master position increments
N = 0	Index
#LOOP	Loop to construct table from equation
P = N*3.6	Note 3.6 = 0.18*20
S = @SIN [P] *100	Define sine position
Y = N *10+S	Define slave position
ET [N] =, Y	Define table
N = N+1	
JP #LOOP, N<=100	Repeat the process
EN	

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: X = 1000 and Y = 500. This implies that Y must be driven to that point to avoid a jump.

This is done with the program:

INSTRUCTION	INTERPRETATION
#RUN	Label
EB1	Enable cam
PA,500	starting position
SP,5000	Y speed
BGY	Move Y motor
AM	After Y moved
AI1	Wait for start signal
EG,1000	Engage slave
AI - 1	Wait for stop signal
EQ,1000	Disengage slave
EN	End

ELECTRONIC CAM

COMMAND SUMMARY - ECAM MODE

COMMAND	DESCRIPTION
EA x,y,z,w	Specify ECAM master axis
EB n (n = 0 or 1)	Enable ECAM
EG x,y,z,w	ECAM go - Specifies position for engineering ECAM
EM x,y,z,w	Specify cam style
EP m,n	Specifies cam table interval and starting point
EQ x,y,z,w	Quit ECAM
ET [n]	Specify ECAM table entry

OPERAND SUMMARY - ECAM MODE

OPERAND	DESCRIPTION
_EB	Contains the state of ECAM mode (0= disabled, 1 = enabled)
_EGx	Contains ecasm status for specified axis (0 = engaged 1, = disengaged)
_EMx	Contains the cycle of the specified axis
_EP	Contains the status of the interval
_EQx	Contains the status of ECAM mode for specified axis

ELECTRONIC CAM EXAMPLE

The following example illustrates a cam program with a master axis, Z, and two slaves, X and Y.

INSTRUCTION	INTERPRETATION
#A;V1=0	Label; Initialize variable
PA 0,0;BGXY;AMXY	Go to position 0,0 on X and Y axes
EA Z	Z axis as the Master for ECAM
EM 0,0,4000	Change for Z is 4000, zero for X, Y
EP400,0	ECAM interval is 400 counts with zero start
ET[0]=0,0	When master is at 0 position; 1st point.
ET[1]=40,20	2nd point in the ECAM table
ET[2]=120,60	3rd point in the ECAM table
ET[3]=240,120	4th point in the ECAM table
ET[4]=280,140	5th point in the ECAM table
ET[5]=280,140	6th point in the ECAM table
ET[6]=280,140	7th point in the ECAM table
ET[7]=240,120	8th point in the ECAM table
ET[8]=120,60	9th point in the ECAM table
ET[9]=40,20	10th point in the ECAM table
ET[10]=0,0	Starting point for next cycle
EB 1	Enable ECAM mode
JGZ=4000	Set Z to jog at 4000
EG 0,0	Engage both X and Y when Master = 0
BGZ	Begin jog on Z axis
#LOOP;JP#LOOP,V1=0	Loop until the variable is set
EQ2000,2000	Disengage X and Y when Master = 2000
MF,, 2000	Wait until the Master goes to 2000
ST Z	Stop the Z axis motion
EB 0	Exit the ECAM mode
EN	End of the program

The above example shows how the ECAM program is structured and how the commands can be given to the controller.

Contour Mode

The SSC also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for 1 to 4 axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

SPECIFYING CONTOUR SEGMENTS

The Contour Mode is specified with the command, CM. For example, CMXZ specifies contouring on the X and Z axes. Any axes that are not being used in the contouring mode may be operated in other modes.

A contour is described by position increments which are described with the command, CD x,y,z,w over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2^n ms, where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each millisecond.

Consider, for example, the trajectory shown in Fig. 8.5. The position X may be described by the points:

Point 1 X=0 at T=0ms
Point 2 X=48 at T=4ms
Point 3 X=288 at T=12ms
Point 4 X=336 at T=28ms

The same trajectory may be represented by the increments:

Increment 1	*DX=48*	*Time=4*	*DT=2*
Increment 2	*DX=240*	*Time=8*	*DT=3*
Increment 3	*DX=48*	*Time=16*	*DT=4*

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

INSTRUCTION	INTERPRETATION
#ACMX	Specifies X axis for contour mode
DT 2	Specifies first time interval, 4 ms
CD 48;WC	Specifies first position increment
DT 3	Specifies second time interval, 8 ms

CD 240;WC	Specifies second position increment
DT 4	Specifies the third time interval, 16 ms
CD 48;WC	Specifies the third position increment
DT0;CD0	Exits contour mode
EN	

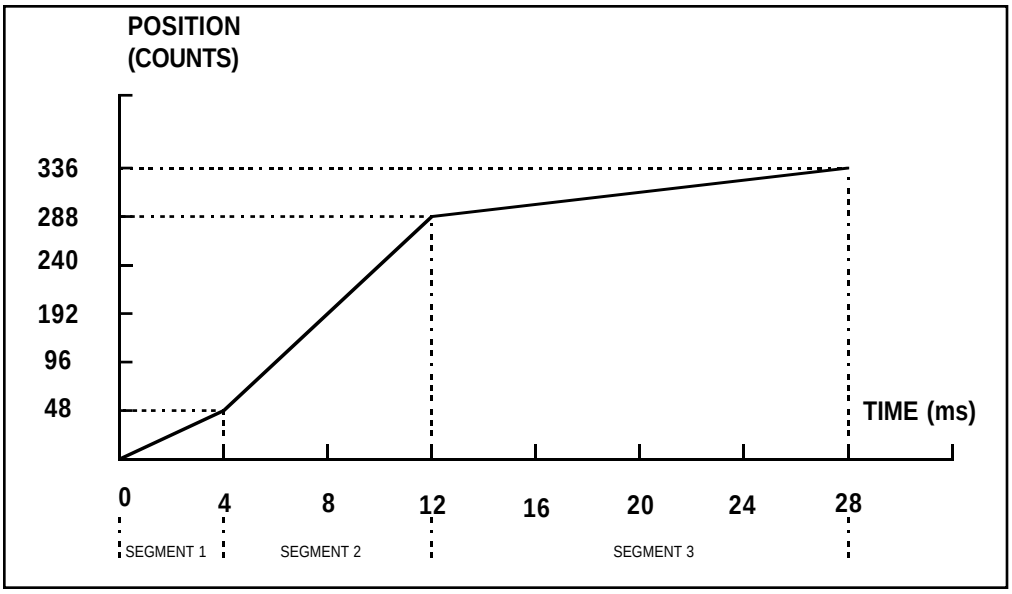


Figure 8.5 - The Required Trajectory

ADDITIONAL COMMANDS

The command, WC, is used as a trippoint "When Complete". This allows the SSC to use the next increment only when it is finished with the previous one. Zero parameters for DT or CD exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

The command _CS, the segment counter, returns the number of the segment being processed. This information allows the host computer to determine when to send additional data.

COMMAND SUMMARY - CONTOUR MODE

COMMAND	DESCRIPTION
CM XYZW	Specifies which axes for contouring mode. Any non-contouring axe may be operated in other modes
CDx,y,z,w	Specifies position increment over tim interval. Range is+/-32,00. Zero end contour mode
DT n	Specifies time interval 2 msec for position increment, where n is an integer between 1 and8. Zero ends contour mode. if n does not change, it does not need to be specified with each CD
WC	Waits for previous time intrval to be completed before next data record is processed

OPERAND SUMMARY - CONTOUR MODE

OPERAND	DESCRIPTION
_CS	Return segment number

CONTOUR EXAMPLE

The Contour Mode is ideal for generating any arbitrary velocity profiles. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Generating an Array

Consider the velocity and position profiles shown in Fig. 8.6. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

Note: ω is the angular velocity; X is the position;
and T is the variable, time, in milliseconds.

$$\omega = \frac{A}{B}(1 - \cos(2\pi / B))$$

$$X = \frac{AT}{B} - \frac{A}{2\pi}\sin(2\pi / B)$$

In the given example, A=6000 and B=120,
the position and velocity profiles are:

X = 50T - (6000/2π) sin (2π T/120)

Note that the velocity, ω , in count/ms, is:

ω = 50 [1 - cos 2π T/120]

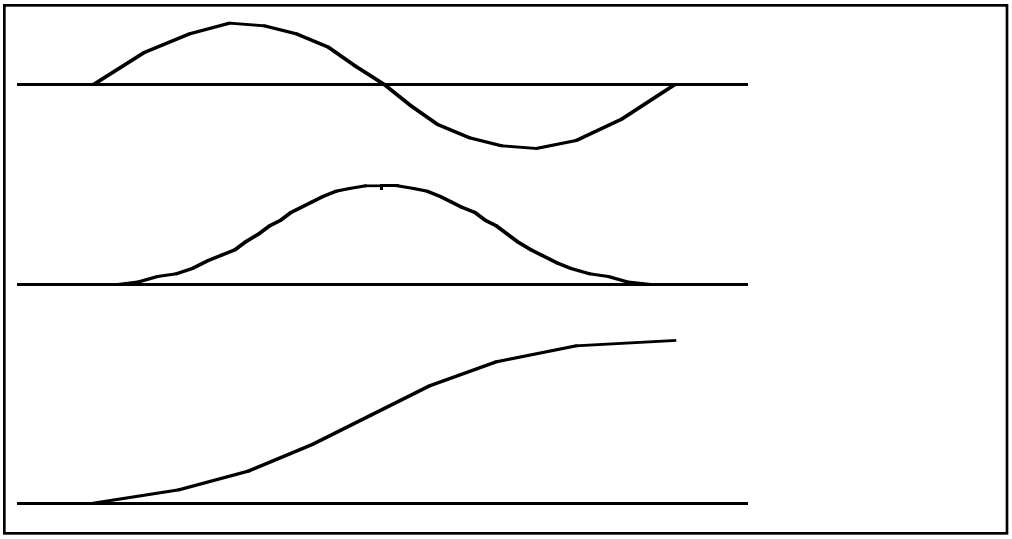


Figure 8.6 - Velocity Profile with Sinusoidal Acceleration

The SSC can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

X = 50T - 955 sin 3T

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

CONTOUR MODE

INSTRUCTION	INTERPRETATION
#POINTS	Program defines X points
DM POS[16]	Allocate memory
DM DIF[15]	
C=0	Set initial conditions, C is index
T=0	T is time in ms
#A	
V1=50*T	
V2=3*T	Argument in degrees
V3=-955*@SIN[V2]+V1	Compute position
V4=@INT[V3]	Integer value of V3
POS[C]=V4	Store in array POS
T=T+8	
C=C+1	
JP #A,C<16	
#B	Program to find position differences
C=0	
#C	
D=C+1	
DIF[C]=POS[D]-POS[C]	Compute the difference and store
C=C+1	
JP #C,C<15	
EN	End first program
#RUN	Program to run motor
CMX	Contour Mode
DT3	4 millisecond intervals
C=0	
#E	
CD DIF[C]	Contour Distance is in DIF
WC	Wait for completion
C=C+1	
JP #E,C<15	
DT0	
CD0	Stop Contour
EN	End the program

Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the SSC automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

INSTRUCTION	INTERPRETATION
DM C[n]	Dimension array
RA C[]	Specify array for automatic record (up to 8 arrays)
RD _TPX	Specify data for capturing (such as _TPX or _TPZ)
RC n,m	Specify capture time interval where n is 2^n msec, m is number of records to be captured
RC? or _RC	Returns a 1 if recording

RECORD AND PLAYBACK EXAMPLE:

INSTRUCTION	INTERPRETATION
#RECORD	Begin Program
DM XPOS[501]	Dimension array with 501 elements
RA XPOS[]	Specify automatic record
RD _TPX	Specify X position to be captured
MOX	Turn X motor off
RC2	Begin recording; 4 msec interval
#A;JP#A,_RC=1	Continue until done recording
#COMPUTE	Compute DX
DM DX[500]	Dimension Array for DX
C=0	Initialize counter
#L	Label
D=C+1	
DELTA=XPOS[D]-XPOS[C]	Compute the difference
DX[C]=DELTA	Store difference in array
C=C+1	Increment index
JP #L,C<500	Repeat until done
#PLAYBCK	Begin Playback
CMX	Specify contour mode
DT2	Specify time increment
I=0	Initialize array counter
#B	Loop counter
CD XPOS[I];WC	Specify contour data
I=I+1	Increment array counter
JP #B,I<500	Loop until done
DT 0;CD0	End contour mode
EN	End program

TEACH (RECORD AND PLAY-BACK)

For additional information about Automatic Array Capture, see Chapter 9, Arrays.

Dual Loop (Auxiliary Encoder)

The SSC provides an interface for a second encoder for each axis except for axes configured for stepper motor operation. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CE x,y,z,w where the parameters x,y,z,w each equal the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

USING THE CE COMMAND

M=	MAIN ENCODER	N=	SECOND ENCODER
0	Normal quadrature	0	Normal quadrature
1	Pulse and direction	4	Pulse and direction
2	Reverse quadrature	8	Reserved quadrature
3	Reverse pulse and direction	12	Reversed pulse and direction

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the X axis is

INSTRUCTION

CE 6

ADDITIONAL COMMANDS FOR THE AUXILIARY ENCODER

The command, DE x,y,z,w, can be used to define the position of the auxiliary encoders. For example,

INSTRUCTION
DE 0,500,-30,300

sets their initial values.

The positions of the auxiliary encoders may be interrogated with the command, DE?. For example

INSTRUCTION
DE ?,,?

returns the value of the X and Z auxiliary encoders.

The auxiliary encoder position may be assigned to variables with the instructions

INSTRUCTION
V1= _DEX

The command, TD XYZW, returns the current position of the auxiliary encoder.

The command, DV XYZW, configures the auxiliary encoder to be used for backlash compensation.

BACKLASH COMPENSATION

There are two methods for backlash compensation using the auxiliary encoders:

1. Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

DUAL LOOP (AUXILIARY ENCODER)

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems which require position accuracy only at the endpoint.

DUAL LOOP EXAMPLE

Continuous Dual Loop

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

INSTRUCTION

DV 1,1,1,1

activates the dual loop for the four axes and

INSTRUCTION

DV 0,0,0,0

disables the dual loop.

Note that the dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KP=0, KI=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

Sampled

In this example, we consider a linear slide which is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the X-axis and connect the linear encoder to the auxiliary encoder of X. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

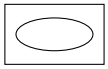
INSTRUCTION	INTERPRETATION
#DUALOOP	Label
CE 0	Configure encoder
DE0	Set initial value
PR 40000	Main move
BGX	Start motion
#Correct	Correction loop
AMX	Wait for motion completion
V1=10000-_DEX	Find linear encoder error
V2=-_TEX/4+V1	Compensate for motor error
JP#END,@ABS[V2]<2	Exit if error is small
PR V2*4	Correction move
BGX	Start correction
JP#CORRECT	Repeat
#END	
EN	

Motion Smoothing

The SSC controller allows the smoothing of the velocity profile to reduce the mechanical vibration of the system.

Trapezoidal velocity profiles have acceleration rates which change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

USING THE IT AND VT COMMANDS (S CURVE PROFILING) (SERVO MOTORS SMOOTHING):



When operating with servo motors, motion smoothing can be accomplished with the IT and VT command. These commands filter the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, known as S curve, has continuous acceleration and results in reduced mechanical vibrations.

MOTION SMOOTHING

The smoothing function is specified by the following commands:

INSTRUCTION	INTERPRETATION
IT x,y,z,w	Independent time constant
VT n	Vector time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and the command, VT, is used to smooth vector moves of the type VM and LM.

The smoothing parameters, x,y,z,w and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Fig. 6.6 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

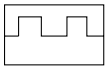
SERVO MOTOR

Smoothing Example

INSTRUCTION	INTERPRETATION
PR 20000	Position
AC 100000	Acceleration
DC 100000	Deceleration
SP 5000	Speed
IT .5	Filter for S-curve
BG X	Begin

Figure 6.6 - Trapezoidal velocity and smooth velocity profiles

USING THE KS COMMAND (STEP MOTOR SMOOTHING):



When operating with step motors, motion smoothing can be accomplished with the command, KS. The KS command smooths the frequency of step motor pulses. Similar to the commands, IT and VT, this produces a smooth velocity profile.

The step motor smoothing is specified by the following command:

KS x,y,z,w where x,y,z,w is an integer from 1 to 16 and represents the amount of smoothing

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and the command, VT, is used to smooth vector moves of the type VM and LM.

The smoothing parameters, x,y,z,w and n are numbers between 0 and 16 and determine the degree of filtering. The minimum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Larger values of the smoothing parameters imply heavier filtering and smoother moves.

Note that KS is valid only for step motors.

Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The velocity profile generated is shown in Fig. 8.7.

The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The command sequence HM and BG causes the following sequence of events to occur.

1. Upon begin, motor accelerates to the slew speed. The direction of its motion is determined by the state of the homing input. A zero (GND) will cause the motor to start in the forward direction; +5V will cause it to start in the reverse direction. The CN command is used to define the polarity of the home input.

2. Upon detecting the home switch changing state, the motor begins decelerating to a stop.

3. The motor then traverses very slowly back until the home switch toggles again.

4. The motor then traverses forward until the encoder index pulse is detected.

5. The SSC defines the home position (0) as the position at which the index was detected.

HOMING EXAMPLES

Home

INSTRUCTION	INTERPRETATION
#HOME	Label
AC 1000000	Acceleration Rate
DC 1000000	Deceleration Rate
SP 5000	Speed for Home Search
HM X	Home X
BG X	Begin Motion
AM X	After Complete
MG “AT HOME”	Send Message
EN	End

Find Edge

INSTRUCTION	INTERPRETATION
#EDGE	Label
AC 2000000	Acceleration rate
DC 2000000	Deceleration rate
SP 8000	Speed
FE Y	Find edge command
BG Y	Begin motion
AM Y	After complete
MG “FOUND HOME”	Print message
DP,0	Define position as 0
EN	End

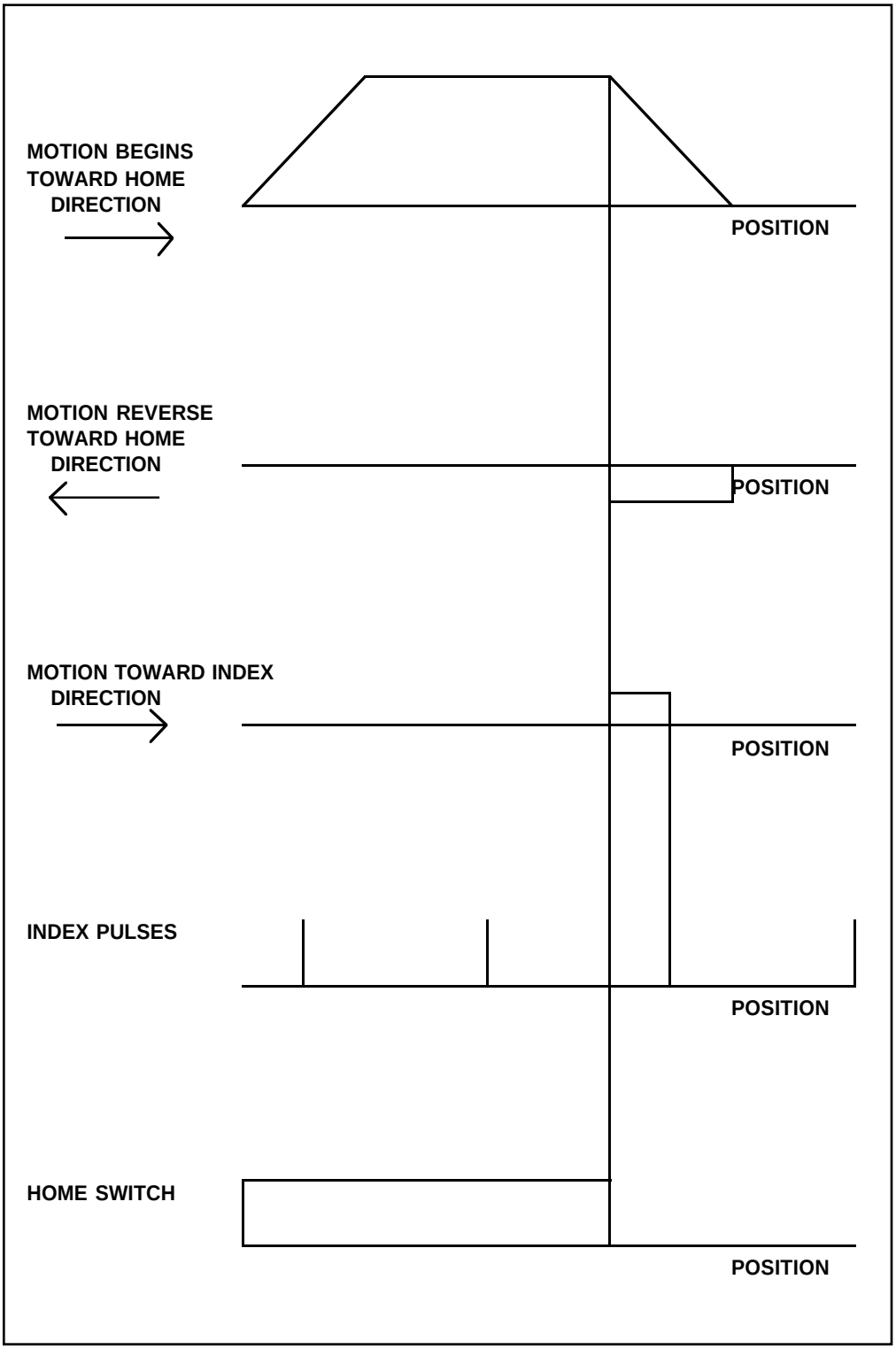


Figure 8.7 - Motion intervals in the Home sequence

High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. The SSC provides a position latch feature. This feature allows the position of X,Y,Z or W to be captured within 25 microseconds of an external low input signal. The general inputs 1 through 4 correspond to each axis.

IN1 X-axis latch
IN2 Y-axis latch
IN3 Z-axis latch
IN4 W-axis latch

Note: To insure a position capture within 25 microseconds, the input signal must be a transition from high to low.

The SSC Software commands, AL and RL, are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL XYZW command, to arm the latch for the specified axis or axes.
2. Test to see if the latch has occurred (Input goes low) by using the _AL X or Y or Z or W command. Example, V1=_ALX returns the state of the X latch into V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the RL XYZW command or _RL XYZW.

Note: The latch must be re-armed after each latching event.

HIGH SPEED POSITION EXAMPLE

INSTRUCTION	INTERPRETATION
#Latch	Latch program
JG,5000	Jog Y
BG Y	Begin motion on Y axis
AL Y	Arm Latch for Y axis
#Wait	#Wait label for loop
JP #Wait,_ALY=1	Jump to #Wait label if latch has not occurred
Result=_RLY	Set value of variable 'Result' equal to the report position of y axis
Result=	Print result
EN	En

Overview

The SSC provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the SSC memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed.

In addition to standard motion commands, the SSC provides commands that allow the SSC to make its own decisions. These commands include conditional jumps, event triggers and subroutines. For example, the command JP#LOOP, n<10 causes a jump to the label #LOOP if the variable n is less than 10.

For greater programming flexibility, the SSC provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss all aspects of creating applications programs.

Using the SSC Editor to Enter Programs

Application programs for the SSC may be created and edited either locally using the Tol-O-Motion SSC Editor or remotely using another editor and then downloading the program into the controller. (Tol-O-Motion's Editor Window provides an editor and UPLOAD and DOWNLOAD utilities).

The SSC provides a line Editor for entering and modifying programs. The Edit mode is entered with the ED instruction. The ED command can only be given when the controller is not running a program.

In the Edit Mode, each program line is automatically numbered sequentially starting with 000. If no parameter follows the ED command, the editor prompter will default to the last line of the program in memory. If desired, the user can edit a specific line number or label by specifying a line number or label following ED.

USING THE SSC EDITOR TO ENTER PROGRAMS

INSTRUCTION	INTERPRETATION
ED	Puts Editor at end of last program
ED 5	Puts Editor at line 5
ED #BEGIN	Puts Editor at label #BEGIN

The program memory space for the SSC is 1000 lines x 80 characters per line

Line numbers appear as 000,001,002 and so on. Program commands are entered following the line numbers. Multiple commands may be given on a single line as long as the total number of characters doesn't exceed the limits given above.

While in the Edit Mode, the programmer has access to special instructions for saving, inserting and deleting program lines. These special instructions are listed below:

EDIT MODE COMMANDS

<RETURN>
Typing the return or enter key causes the current line of entered instructions to be saved. The editor will automatically advance to the next line. Thus, hitting a series of <RETURN> will cause the editor to advance a series of lines. Note, changes on a program line will not be saved unless a <return> is given.

<cntrl>P
The <cntrl>P command moves the editor to the previous line.

<cntrl>I
The <cntrl>I command inserts a line above the current line. For example, if the editor is at line number 2 and <cntrl>I is applied, a new line will be inserted between lines 1 and 2. This new line will be labeled line 2. The old line number 2 is renumbered as line 3.

<cntrl>D
The <cntrl>D command deletes the line currently being edited. For example, if the editor is at line number 2 and <cntrl>D is applied, line 2 will be deleted. The previous line number 3 is now renumbered as line number 2.

<cntrl>Q
The <cntrl>Q quits the editor mode. In response, the SSC will return a colon.

After the Edit session is over, the user may list the entered program using the LS command. If no number or label follows the LS command, the entire program will be listed. The user can start listing at a specific line or label. A range of program lines can also be displayed. For example;

INSTRUCTION	INTERPRETATION
LS	List entire program
LS 5	Begin listing at line 5
LS 5,9	List lines 5 through 9
LS #A,9	List line label #A through line 9

Program Format

A SSC program consists of SSC instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program Flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

Each SSC instruction in a program must be separated by a delimiter. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line where the maximum number of instructions on a line is limited by 38 characters. A carriage return enters the final command on a program line.

USING LABELS IN PROGRAMS

All SSC programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

The maximum number of labels allowed on the SSC series controller is 254.

INSTRUCTION	INTERPRETATION
Valid Labels	
#BEGIN	
#SQUARE	
#X1	
#Begin1	
Invalid Labels	Problem
#1Square	Can not use number to begin a label
#SQUAREPEG	Can not use more than 7 characters in a label

PROGRAM FORMAT

LABEL EXAMPLE

INSTRUCTION	INTERPRETATION
#START	Beginning of the Program
PR 10000,20000	Specify relative distances on X and Y axes
BG XY	Begin Motion
AM	Wait for motion complete
WT 2000	Wait 2 sec
JP #START	Jump to label START
EN	End of Program

The above program moves X and Y 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

SPECIAL LABELS

The SSC has some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines.

INSTRUCTION	INTERPRETATION
#AUTO	Label for autoprogram start
#ININT	Label for Input Interrupt subroutine
#LIMSWI	Label for Limit Switch subroutine
#POSERR	Label for excess Position Error subroutine
#MCTIME	Label for timeout on Motion Complete trip point
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for communication interrupt subroutine

COMMENTING PROGRAMS

Using the Command, NO

The SSC provides a command, NO, for commenting programs. This command allows the user to include up to 37 characters on a single line after the NO command and can be used to include comments from the programmer as in the following example:

```
INSTRUCTION
#PATH
NO 2-D CIRCULAR PATH
VMXY
NO VECTOR MOTION ON X AND Y
VS 10000
NO VECTOR SPEED IS 10000
VP -4000,0
NO BOTTOM LINE
CR 1500,270,-180
NO HALF CIRCLE MOTION
VP 0,3000
NO TOP LINE
CR 1500,90,-180
NO HALF CIRCLE MOTION
VE
NO END VECTOR SEQUENCE
BGS
NO BEGIN SEQUENCE MOTION
EN
NO END OF PROGRAM
```

Note: The NO command is an actual controller command. Therefore, inclusion of the NO commands will require process time by the controller.

Using REM Statements with the Tol-O-Matic Terminal Software.

If you are using Tol-O-Matic Software to communicate with the SSC controller, you may also include REM statements. 'REM' statements begin with the word 'REM' and may be followed by any comments which are on the same line. The Tol-O-Matic terminal Software will remove these statements when the program is downloaded to the controller. For example:

PROGRAM FORMAT

```
INSTRUCTION
#PATH
REM 2-D CIRCULAR PATH
VMXY
REM VECTOR MOTION ON X AND Y
VS 10000
REM VECTOR SPEED IS 10000
VP -4000,0
REM BOTTOM LINE
CR 1500,270,-180
REM HALF CIRCLE MOTION
VP 0,3000
REM TOP LINE
CR 1500,90,-180
REM HALF CIRCLE MOTION
VE
REM END VECTOR SEQUENCE
BGS
REM BEGIN SEQUENCE MOTION
EN
REM END OF PROGRAM
```

These REM statements will be removed when this program is downloaded to the controller.

Executing Programs & Multitasking

The SSC can run up to four independent programs simultaneously. These programs are called threads and are numbered 0 through 3, where 0 is the main one. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. Only the main thread may use the input command, IN.
2. When input interrupts are implemented for limit switches, position errors or command errors, the subroutines are executed in thread 0.

To begin execution of the various programs, use the following instruction:

XQ #A, n

Where n indicates the thread number. If the XQ command is given with no parameters, the first program in memory will be executed in thread 0.

To halt the execution of any thread, use the instruction

HX n

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

MULTITASKING EXAMPLE

Waveform on Output 1 Independent of a Move.

INSTRUCTION	INTERPRETATION
#TASK1	Task1 label
AT0	Initialize reference time
CB1	Clear Output 1
#LOOP1	Loop1 label
AT 10	Wait 10 msec from reference time
SB1	Set Output 1
AT -40	Wait 40 msec from reference time, then initialize reference
CB1	Clear Output 1
JP #LOOP1	Repeat Loop1
#TASK2	Task2 label
XQ #TASK1,1	Execute Task1
#LOOP2	Loop2 label
PR 1000	Define relative distance
BGX	Begin motion
AMX	After motion done
WT 10	Wait 10 msec
JP #LOOP2,@IN[2]=1	Repeat motion unless Input 2 is low
HX	Halt all tasks

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread (ie. Thread 0). #TASK1 is executed within TASK2.

Debugging Programs

COMMANDS

The SSC provides trace and error code commands which are used in debugging programs. The trace command causes the controller to send each line in a program to the host computer immediately prior to execution.

Tracing is enabled with the command, TR1. TR0 turns the trace function off. Note: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

When there is a program error, the SSC halts the program execution at the point where the error occurs. The line number is then displayed.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message which describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

The SSC provides the capability to check the available program memory and array memory. The command, DM ?, will return the number of array elements currently available. The command, DA ?, will return the number of arrays currently available. For example, a standard SSC will have a maximum of 8000 array elements in up to 30 arrays. If an array of 100 elements is defined, the command DM ? will return the value 7900 and the command DA ? will return 29.

OPERANDS

The operand _ED will return the value of the last line executed and can be used to determine where an error occurred. For example, the command MG _ED will display the line number in the program that failed.
The operand _DL returns the number of available labels.
The operand _UL returns the number of available variables.
The operand _DA returns the number of available arrays.
The operand _DM returns the number of available array elements.

DEBUGGING EXAMPLE:

The following program has an error. It attempts to specify a relative movement while the X-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

INSTRUCTION	INTERPRETATION
:ED	Edit Mode
000 #A	Program Label
001 PR1000	Position Relative 1000
002 BGX	Begin
003 PR5000	Position Relative 5000
004 EN	End
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A
?003 PR5000	Error on Line 3
:TC1	Tell Error Code
?7 Command not valid while running.	Command not valid while running
:ED 3	Edit Line 3
003 AMX;PR5000;BGX	Add After Motion Done
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A

Program Flow Commands

The SSC provides instructions to control program flow. The SSC program sequencer normally executes program instructions sequentially. The program flow can be altered with the use of event triggers, trippoints, and conditional jump statements.

EVENT TRIGGERS & TRIPPOINTS

To function independently from the host computer, the SSC can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The SSC provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is “tripped”. For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the SSC can make decisions based on its own status or external events without intervention from a host computer.

PROGRAM FLOW COMMANDS

SSC EVENT TRIGGERS

COMMAND	FUNCTION
AM X Y Z W or S	Halts program until motion is complete on the specified axes or motion sequence (s). AM with no parameter tests for motion complete on all axes. This command is useful for separating motion sequences in a program.
AD X or Y or Z or W	Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time.
AR X or Y or Z or W	Halts program execution until after specified distance from the last\ AR or AD command has elapsed. Only one axis may be specifies at a time.
AP X or Y or Z or W	Halts program execution until after absolute position occurs. Only one axis may be specified at a time.
MF X or Y of r Z or W	Halts program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately. will function on geared axis.
MC X or Y or Z or W	Halt program execution until after the motion profile has been completed and the encoder =has entered or passed the specified position. TW x.y.z.w sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stepcode will be set to 99. An application program will jump to label #MCTIME.
AI +/-n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level. n=1 through 24.
AS X Y Z W S	Halts program execution until specified axis has reached its slew speed.
AT +/-n	Halts program execution until n msec from reference time. AT O sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time.
AV n	Halts program execution until specified distance along a coordinated path has occurred.
WT n	Halts program execution until specified time in msec has elapsed.

EVENT TRIGGER EXAMPLES:

Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

INSTRUCTION	INTERPRETATION
#TWOMOVE	Label
PR 2000	Position Command
BGX	Begin Motion
AMX	Wait for Motion Complete
PR 4000	Next Position Move
BGX	Begin 2nd move
EN	End program

Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

INSTRUCTION	INTERPRETATION
#SETBIT	Label
SP 10000	Speed is 10000
PA 20000	Specify Absolute position
BGX	Begin motion
AD 1000	Wait until 1000 counts
SB1	Set output bit 1
EN	End program

Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

INSTRUCTION	INTERPRETATION
#TRIP	Label
JG 50000	Specify Jog Speed
BGX;n=0	Begin Motion
#REPEAT	# Repeat Loop
AR 10000	Wait 10000 counts
TPX	Tell Position
SB1	Set output 1
WT50	Wait 50 msec
CB1	Clear output 1
n=n+1	Increment counter
JP #REPEAT,n<5	Repeat 5 times
STX	Stop
EN	End

PROGRAM FLOW COMMANDS

Start Motion on Input

This example waits for input 1 to go low and then starts motion. Note: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as JP #GO,@IN[1] = -1.

INSTRUCTION	INTERPRETATION
#INPUT	Program Label
AI-1	Wait for input 1 low
PR 10000	Position command
BGX	Begin motion
EN	End program

Set Output When At Speed

INSTRUCTION	INTERPRETATION
#ATSPEED	Program Label
JG 50000	Specify jog speed
AC 10000	Acceleration rate
BGX	Begin motion
ASX	Wait for at slew speed 50000
SB1	Set output 1
EN	End program

Change Speed Along Vector Path

The following program changes the feedrate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

INSTRUCTION	INTERPRETATION
#VECTOR	Label
VMXY;VS 5000	Coordinated path
VP 10000,20000	Vector position
VP 20000,30000	Vector position
VE	End vector
BGS	Begin sequence
AV 5000	After vector distance
VS 1000	Reduce speed
EN	End

Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

INSTRUCTION	INTERPRETATION
#MOVES	Label
PR 12000	Distance
SP 20000	Speed
AC 100000	Acceleration
BGX	Start Motion
AD 10000	Wait a distance of 10,000 counts
SP 5000	New Speed
AMX	Wait until motion is completed
WT 200	Wait 200 ms
PR -10000	New Position
SP 30000	New Speed
AC 150000	New Acceleration
BGX	Start Motion
EN	End

Creating an Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

INSTRUCTION	INTERPRETATION
#OUTPUT	Program label
AT0	Initialize time reference
SB1	Set Output 1
#LOOP	Loop
AT 10	After 10 msec from reference,
CB1	Clear Output 1
AT -40	Wait 40 msec from reference and reset reference
SB1	Set Output 1
JP #LOOP	Jump to location #LOOP and continue executing commands
EN	End of program

Conditional Jumps

The SSC provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location. Program execution will continue at the location specified by the JP or JS command if the conditional statement is satisfied. If no conditional statement is provided, the jump will occur automatically. See description of conditional statements below.

Conditional jumps are useful for testing events in real-time since they allow the SSC to make decisions without a host computer. For example, the SSC can begin execution at a specified label or line number based on the state of an input line.

USING THE JP COMMAND:

The JP command will cause the controller to execute commands at the location specified by the label or line number if the condition of the jump statement is satisfied. If no condition is specified, program execution will automatically jump to the specified line. If the condition is not satisfied, the controller will continue to execute the next commands in the program sequence.

USING THE JS COMMAND:

The JS command is significantly different from the JP command. When the condition specified by the JS command is satisfied, the controller will begin execution at the program location specified by the line or label number. If no conditional statement is given, the jump will always occur. However, when the controller reaches an end statement, EN, the controller will jump back to the location of the JS command and resume executing the next commands. This is known as jumping to a subroutine. For more information, see Subroutines on page 9-17.

Each jump to a subroutine causes the controller to save the line number of the jump statement. This information is saved in an area of program memory called the program stack. The program stack can save up to 16 line numbers allowing a program to nest up to 16 jumps to subroutines. If it is necessary to remove entries from the program stack, use the command ZS. For example, while executing a subroutine, the program can be kept from jumping back to the original program line by issuing the command ZS. This will remove all entries in the program stack and continue executing at the current line. See Stack Manipulation on page 9-18 and the Two Letter Command Reference in Chapter 12.

CONDITIONAL STATEMENTS

The conditional statement is satisfied if it evaluates to any value other than zero. The conditional statement can be any valid SSC numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions.

CONDITION	TYPE
V1=6	Number
V1=V7*6	Numeric Expression
@ABS[V1]>10	
V1<Count[2]	Array Element
V1<V2	Variable
_TPX=0	Internal Variable
_TVX>500	
V1>@AN[2]	I/O
@IN[1]=0	

Multiple Conditional Statements

The SSC will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true. Note: Each condition must be placed in parenthesis for proper evaluation by the controller. In addition, the SSC will execute operations from left to right. For further information on Mathematical Expressions and the bit-wise operators '&' and '|', see pg. 9-23.

For example, using variables named V1, V2, V3 and V4:
JP #TEST, (V1<V2) & (V3<V4)

In this example, this statement will cause the program to jump to the label #TEST if V1 is less than V2 and V3 is less than V4. To illustrate this further, consider this same example with an additional condition:

JP #TEST, ((V1<V2) & (V3<V4)) | (V5<V6)

This statement will cause the program to jump to the label #TEST under two conditions; 1. If V1 is less than V2 and V3 is less than V4. OR 2. If V5 is less than V6.

CONDITIONAL JUMPS

JUMP EXAMPLES

Using JP and JS

INSTRUCTION	INTERPRETATION
JP #Loop, COUNT<10	Jump to #Loop if the variable, COUNT, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[V2]>2	Jump to #BLUE if the absolute value of variable, V2, is greater than 2
JP #C,V1*V7<=V8*V2	Jump to #C if the value of V1 times V7 is less than or equal to the value of V8*V2
JP#A	Jump to #A

Using JP command:

Move the X motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

INSTRUCTION	INTERPRETATION
#BEGIN	Begin Program
COUNT=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGX	Begin move
AMX	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGX	Begin move
AMX	Wait for motion complete
WT 100	Wait 100 msec
COUNT=COUNT-1	Decrement loop counter
JP #LOOP,COUNT>0	Test for 10 times through loop
EN	End Program

COMMAND FORMAT - JP AND JS

FORMAT	DESCRIPTION
JS destination, logical condition	Jump to subroutine if logical condition is satisfied
JP destination, logical condition	Jump to location if logical condition is satisfied

The destination is a program line number or label where the program sequencer will jump if the specified condition is satisfied. Note that the line number of the first line of program memory is 0. The comma designates "IF". The logical condition tests two operands with logical operators.

LOGICAL OPERATORS:

OPERATOR	DESCRIPTION
<	less than
>	greater than
>	equal to
<=	less than or equal to
>=	greater than if equal to
<>	not equal

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

SUBROUTINE EXAMPLE

Subroutine to draw a square 500 counts on each side. The square starts at vector position 1000,1000.

INSTRUCTION	INTERPRETATION
#M	Begin main program
CB1	Clear Output Bit 1 (pick up pen)
VMXY	Specify vector motion between X and Y axes
VP 1000,1000;VE;BGS	Define vector position; move pen
AMS	Wait for after motion trippoint
SB1	Set Output Bit 1 (put down pen)
JS #Square;CB1	Jump to square subroutine
EN	End main program
#Square	Square subroutine
V1=500;JS #L	Define length of side, Jump to subroutine #L
V1=-V1;JS #L	Switch direction, Jump to subroutine #L
EN	End subroutine #Square
#L;PR V1,V1;BGX	Subroutine #L, Define relative position movement on X and Y; Begin motion
AMX;BGY;AMY	After motion on X, Begin Y, Wait for motion on Y to complete
EN	End subroutine #L

STACK MANIPULATION

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

AUTOMATIC SUBROUTINES FOR MONITORING CONDITIONS

Often it is desirable to monitor certain conditions continuously without tying up the host or SSC program sequences. The SSC can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, predefined label in the applications program. The pre-defined labels are:

SUBROUTINE	DESCRIPTION
#AUTO	Automatically start program on power up
#LIMSWI	Limit switch on any axis goes low
#ININT	Iput specified by II goes low
#POSERR	Position error exceeds limit specified by ER
#MCTIME	Motion complete timeout occurred. Timeout period set by TW command
#CMDERR	Bad command given
#COMINT	Commuication interrupt routine

For example, the #POSERR subroutine will automatically be executed when any axis exceeds its position error limit. The commands in the #POSERR subroutine could decode which axis is in error and take the appropriate action. In another example, the #ININT label could be used to designate an input interrupt subroutine. When the specified input occurs, the program will be executed automatically.

NOTE: An application program must be running for automatic monitoring to function.

MORE SUBROUTINE EXAMPLES

Limit Switch

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the SSC must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP #LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

INSTRUCTION	INTERPRETATION
#LOOP	Dummy Program
JP #LOOP;EN	Jump to Loop
#LIMSWI	Limit Switch Label
MG "LIMIT OCCURRED"	Print Message
RE	Return to main program
XQ #LOOP	Execute Dummy Program
JG 5000	Jog X axis at rate of 5000 counts / sec
BGX	Begin motion on X axis

Now, when a forward limit switch occurs on the X axis, the #LIMSWI subroutine will be executed.

NOTE: The RE command is used to return from the #LIMSWI subroutine. The #LIMSWI will continue to be executed until the limit switch is cleared (goes high).

Position Error

INSTRUCTION	INTERPRETATION
#LOOP	Dummy Program
JP #LOOP;EN	Loop
#POSERR	Position Error Routine
V1=_TEX	Read Position Error
MG "EXCESS POSITION ERROR"	Print Message
MG "ERROR=",V1=	Print Error
RE	Return from Error

While running the 'dummy' program, if the position error on the X axis exceeds that value specified by the ER command, the #POSERR routine will execute.

NOTE: The RE command is used to return from the #POSERR subroutine The #POSERR routine will continue to be executed until the position error is cleared (is less than the ER limit).

Input Interrupt

INSTRUCTION	INTERPRETATION
#A	Label
II1	Input Interrupt on 1
JG 30000,,,60000	Jog
BGXW	Begin Motion
#LOOP;JP#LOOP;EN	Loop
#ININT	Input Interrupt
STXW;AM	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
JG 30000,,,6000	Restore Velocities
BGXW;RI	Begin motion and Return to Main Program
EN	

NOTE: Use the RI command to return from #ININT subroutine.

Motion Complete Timeout

INSTRUCTION	INTERPRETATION
#BEGIN	Begin main program
TW 1000	Set the time out to 1000 ms
PA 10000	Position Absolute command
BGX	Begin motion
MCX	Motion Complete trip point
EN	End main program
#MCTIME	Motion Complete Subroutine
MG “X fell short”	Send out a message
EN	End subroutine

This simple program will issue the message “X fell short” if the X axis does not reach the commanded position within 1 second of the end of the profiled move.

Bad Command

INSTRUCTION	INTERPRETATION
#BEGIN	Begin main program
IN “ENTER SPEED”, SPEED	Prompt for speed
JG SPEED;BGX;	Begin motion
JP #BEGIN	Repeat
EN	End main program
#CMDERR	Command error utility
JP#DONE,_ED<>2	Check if error on line 2
JP#DONE,_TC<>6	Check if out of range

SUBROUTINES

MG “SPEED TOO HIGH”	Send message
MG “TRY AGAIN”	Send message
ZS1	Adjust stack
JP #BEGIN	Return to main program
#DONE	End program if other error
ZS0	Zero stack
EN	End program

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

Communication Interrupt

A SSC is used to move the X axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an auxiliary port terminal.

INSTRUCTION	INTERPRETATION
#BEGIN	Label for beginning of program
CC 9600,0,0,0	Setup communication configuration for auxiliary serial port
CI ,2	Setup communication interrupt for auxiliary serial port
MG {P2}”Type 0 to stop motion”	Message out of auxiliary port
MG {P2}”Type 1 to pause motion”	Message out of auxiliary port
MG {P2}”Type 2 to resume motion”	Message out of auxiliary port
RATE=2000	Variable to remember speed
SPX=RATE	Set speed of X axis motion
#LOOP	Label for Loop
PAX=10000	Move to absolute position 10000
BGX	Begin Motion on X axis
AMX	Wait for motion to be complete
PAX=0	Move to absolute position 0
BGX	Begin Motion on X axis
AMX	Wait for motion to be complete
JP #LOOP	Continually loop to make back and forth motion
EN	End main program
#COMINT	Interrupt Routine
CI0	Clear interrupt
JP #STOP,P2CH=”0”	Check for 0 (stop motion)
JP #PAUSE,P2CH=”1”	Check for 1 (pause motion)
JP #RESUME,P2CH=”2”	Check for 2 (resume motion)
EN1,1	Do nothing
#STOP	Routine for stopping motion
STX;ZS;EN	Stop motion on X axis; Zero program stack; End Program

#PAUSE	Routine for pausing motion
RATE=_SPX	Save current speed setting of X axis motion
SPX=0	Set speed of X axis to zero (allows for pause)
#RESUME	Routine for resuming motion
SPX=RATE	Set speed on X axis to original speed
EN1,1	Re-enable trip-point and Re-enable the communication interrupt

Mathematical Expressions

For manipulation of data, the SSC provides the use of the following mathematical operators:

OPERATOR	FUNCTION
+	Addition
–	Subtraction
*	Multiplication
/	Division
&	Logical And (bit-wise)
|	Logical or (n some computers, a solid line appears as a broken line)
( )	Parenthesis

The numeric range for addition, subtraction and multiplication operations is +/-2,147,483,647.9999. The precision for division is 1/65,000.

Mathematical operations are executed from left to right. Calculations within a parentheses have precedence.

MATHEMATICAL EXPRESSIONS

INSTRUCTION	INTERPRETATION
SPEED=7.5*V1/2	The variable, SPEED, is equal to 7.5 multiplied by V1 and divided by 2
COUNT=COUNT+2	The variable, COUNT, is equal to the current value plus 2.
RESULT=_TPX-(@COS[45]*40)	Puts the position of X - 28.28 in RESULT. 40 * cosine of 45° is 28.28
TEMP=@IN[1]&@IN[2]	TEMP is equal to 1 only if Input 1 and Input 2 are high

MATHEMATICAL EXPRESSIONS

BIT-WISE OPERATORS

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid SSC numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. The bit-wise operators may also be used with strings.

Bit-wise operators are useful for separating characters from an input string. When using the input command for string input, the input variable holds 6 bytes of data. Each byte is eight bits, so a number represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold a six character string. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

INSTRUCTION	INTERPRETATION
#TEST	Begin main program
IN “ENTER”,LEN{S6}	Input character string of up to 6 characters into variable ‘LEN’
FLEN=@FRAC[LEN]	Define variable ‘FLEN’ as fractional part of variable ‘LEN’
FLEN=$10000*FLEN	Shift FLEN by 32 bits (IE - convert fraction, FLEN, to integer)
LEN1=(FLEN&$00FF)	Mask top byte of FLEN and set this value to variable ‘LEN1’
LEN2=(FLEN&$FF00)/$100	Let variable, ‘LEN2’ = top byte of FLEN
LEN3=LEN&$000000FF	Let variable, ‘LEN3’ = first (lowest significant) byte of LEN
LEN4=(LEN&$0000FF00)/$100	Let variable, ‘LEN4’ = second byte of LEN
LEN5=(LEN&$00FF0000)/$10000	Let variable, ‘LEN5’ = third byte of LEN
LEN6=(LEN&$FF000000)/$1000000	Let variable, ‘LEN6’ = fourth byte of LEN
MG LEN6 {S4}	Display ‘LEN6’ as string message of up to 4 chars
MG LEN5 {S4}	Display ‘LEN5’ as string message of up to 4 chars
MG LEN4 {S4}	Display ‘LEN4’ as string message of up to 4 chars
MG LEN3 {S4}	Display ‘LEN3’ as string message of up to 4 charsMG
MG LEN2 {S4}	Display ‘LEN2’ as string message of up to 4 chars
MG LEN1 {S4}	Display ‘LEN1’ as string message of up to 4 chars
EN	

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding ‘$’).

To illustrate further, if the user types in the string “TESTME” at the input prompt, the controller will respond with the following:

T Response from command MG LEN6 {S4}
E Response from command MG LEN5 {S4}
S Response from command MG LEN4 {S4}
T Response from command MG LEN3 {S4}
M Response from command MG LEN2 {S4}
E Response from command MG LEN1 {S4}

Functions

FUNCTION	DESCRIPTION
@SIN[n]	Sine of n (n in degrees, resolution of 1/128, degrees, max +/- 4 billion)
@COS[n]	Cosine of n (n in degrees , resolution of 1/128, degrees max +/- 4 billion)
@COM[n]	1's compliment of n
@ABS[n]	Absolute value of n
@FRAC[n]	fraction portion of n
@INT[n]	Interger portion of n
@RND[n]	Round of n (rounds up if the fractional part of n is .5 or greater)
@SQR[n]	Square root of (accuracy is +/- .0004)
@IN[n]	Return status of digital input n
@OUT[n]	status of digital outp[ut n
@AN[n]	Return volatage measured at analog input n

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

USING FUNCTIONS

INSTRUCTION	INTERPRETATION
V1=@ABS[V7]	The variable, V1, is equal to the absolute value of variable V7.
V2=5*@SIN[POS]	The variable, V2, is equal to five times the sine of the variable, POS.
V3=@IN[1]	The variable, V3, is equal to the digital value of input 1.
V4=2*(5+@AN[5])	The variable, V4, is equal to the value of analog input 5 plus 5, then multiplied by 2.

Variables

The maximum number of variables available with a SSC is 254. These variables can be numbers or strings. Variables are useful in applications where specific parameters, such as position or speed, must be able to change. Variables can be assigned by an operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character, however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as SSC instructions. For example, PR is not a good choice for a variable name.

VALID VARIABLE NAMES

VARIABLE
POSX
POS1
SPEEDZ

INVALID VARIABLE NAMES

VARIABLE	PROBLEM
REALLONGNAME	Cannot have more than 8 characters
124	Cannot begin variable name with a number
SPEED Z	Cannot have spaces in the name

ASSIGNING VALUES TO VARIABLES:

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings;

Variables hold 6 bytes of data, 4 bytes of integer (231)followed by two bytes of fraction providing a range of values of +/-2,147,483,647.9999.

Numeric values can be assigned to programmable variables using the equal sign.

Any valid SSC function can be used to assign a value to a variable. For example, V1=@ABS[V2] or V2=@IN[1]. Arithmetic operations are also permitted.

To assign a string value, the string must be in quotations. String variables can contain up to six characters which must be in quotations.

Variable values may be assigned to controller parameters such as PR or SP.

EXAMPLE

INSTRUCTION	INTERPRETATION
POSX=_TPX	Assigns returned value from TPX command to variable POSX.
SPEED=5.75	Assigns value 5.75 to variable SPEED
INPUT=@IN[2]	Assigns logical value of input 2 to variable INPUT
V2=V1+V3*V4	Assigns the value of V1 plus V3 times V4 to the variable V2.
VAR="CAT"	Assign the string, CAT, to VAR
PR V1	Assign value of variable V1 to PR command for X axis
SP VS*2000	Assign VS*2000 to SP command

Displaying the Value of Variables at the Terminal

Variables may be sent to the screen using the format, variable=. For example, V1= , returns the value of the variable V1.

PROGRAM EXAMPLE

Using Variables for Joystick

The example below reads the voltage of an X-Y joystick and assigns it to variables VX and VY to drive the motors at proportional velocities, where

10 Volts = 3000 rpm = 200000 c/sec
Speed/Analog input = 200000/10 = 20000

INSTRUCTION	INTERPRETATION
#JOYSTIK	Label
JG 0,0	Set in Jog mode
BGXY	Begin Motion
#LOOP	Loop
VX=@AN[1]*20000	Read joystick X
VY=@AN[2]*20000	Read joystick Y
JG VX,VY	Jog at variable VX,VY
JP#LOOP	Repeat
EN	End

Operands

Operands allow motion or status parameters of the SSC to be incorporated into programmable variables and expressions. An operand contains data and must be used in a valid expression or function. Most SSC commands have an equivalent operand - which are designated by adding an underscore (_) prior to the SSC command. Commands which have an associated operand are listed in the Command Reference as “Used as an Operand” .. Yes.

Status commands such as Tell Position return actual values, whereas action commands such as GN or SP return the values in the SSC registers. The axis designation is required following the command.

EXAMPLES

Operand Usage

OPERAND	DESCRIPTION
_BGn	* Is equal to a 1 if ,motion on axis 'n' is complete, otherwise equal to 0.
_DA	*Is equal to the number of arrays available
_DL	*Is equal to the available memory
_HMn	*Is equal to status of Home Switch (equals 0 or 1)
_LFn	* Is equal to status of forward limit switch input of axis 'n' (equals 0 or 1)
_LRX	* Is equal to status of reverse limit switch input of axis 'n' (equals 0 or 1)
_UL	* Is equal to the number of available variables
TIME	Free-running real time clock (off by 2.4% - resets with power on). NOTE: TIME does not use an underscore character (_) as other keys.

* - These keywords have corresponding commands while the keywords _LF, _LR, and TIME do not have any associated commands. All keywords are listed in the Command Summary, Chapter 12.

Keywords

INSTRUCTION	INTERPRETATION
V1=_LFX	Assign V1 the logical state of the Forward Limit Switch on the X-axis
V3=TIME	Assign V3 the current value of the time clock
V4=_HMW	Assign V4 the logical state of the Home input on the W-axis

Arrays

For storing and collecting numerical data, the SSC provides array space for 8000 elements. The arrays are one dimensional and up to 30 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer (2^{31})followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application.

DEFINING ARRAYS

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an uppercase alphabetic character. The number of entries in the defined array is enclosed in [ ].

EXAMPLE

Using the Command, DM

INSTRUCTION	INTERPRETATION
DM POSX[7]	Defines an array names POSX with seven entries
DM SPEED[100]	Defines an array named speed with 100 entries
DM POSX[0]	Frees array space

ASSIGNMENT OF ARRAY ENTRIES

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the POSX array (defined with the DM command, DM POSX[7]) would be specified as POSX[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

NOTE: Arrays must be defined using the command, DM, before assigning entry values.

EXAMPLE

Assigning Values to Array Entries

INSTRUCTION	INTERPRETATION
DM SPEED[10]	Dimension Speed Array
SPEED[1]=7650.2	Assigns the first element of the array, SPEED the value 7650.2
SPEED[1]=	Returns array element value
POSX[10]=_TPX	Assigns the 10th element of the array POSX the returned value from the tell position command.
CON[2]=@COS[POS]*2	Assigns the second element of the array CON the cosine of the variable POS multiplied by 2.
TIMER[1]=TIME	Assigns the first element of the array timer the returned value of the TIME keyword.

USING A VARIABLE TO ADDRESS ARRAY ELEMENTS

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter. For example;

INSTRUCTION	INTERPRETATION
#A	Begin Program
COUNT=0;DM POS[10]	Initialize counter and define array
#LOOP	Begin loop
WT 10	Wait 10 msec
POS[COUNT]=_TPX	Record position into array element
POS[COUNT]=	Report position
COUNT=COUNT+1	Increment counter
JP #LOOP,COUNT<10	Loop until 10 elements have been stored
EN	End Program

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named POS. The variable, COUNT, is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described below.

Uploading and Downloading Arrays to On Board Memory

Arrays may be uploaded and downloaded using the QU and QD commands.

INSTRUCTION
QU array[],start,end,delim
QD array[],start,end

where array is an array name such as A[].

Start is the first element of array (default=0)

End is the last element of array (default=last element)

Delim specifies whether the array data is separated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

AUTOMATIC DATA CAPTURE INTO ARRAYS

The SSC provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to eight types of data can be captured and stored in four arrays. The capture rate or time interval may be specified. Recording can done as a one time event or as a circular continuous recording.

COMMAND SUMMARY - AUTOMATIC DATA CAPTURE

COMMAND	DESCRIPTION
RA n[],m[],o[],p[]	Selects up to four arrays for data capture. The arrays must be defined with the DM command.
RD type1, type2, type3, type4	Selects the type of data to be recorded, where type1, type2, type3, and type4 represent the various tyoe of data (see table below). The order of data type is important and corresponds with the order of m,n,o,p arrays in the arrays command.
RC n,m	The RC command begins data collection. Sets data capture time interval where n s an integer between 1 and 8 and designates 2n between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. n=0 stops recording.
RC?	Returns 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress.

Note: X may be replaced by Y,Z or W for capturing data on other axes.

OPERAND SUMMARY - AUTOMATIC DATA CAPTURE

OPERAND	DESCRIPTION
_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress.
_RD	Return address of next array element.

EXAMPLE

Recording into An Array

During a position move, store the X and Y positions and position error every 2 msec.

INSTRUCTION	INTERPRETATION
#RECORD	Begin program
DM XPOS[300],YPOS[300]	Define X,Y position arrays
DM XERR[300],YERR[300]	Define X,Y error arrays
RA XPOS[],XERR[],YPOS[],YERR[]	Select arrays for capture
RD _TPX,_TEX,_TPY,_TEY	Select data types
PR 10000,20000	Specify move distance
RC1	Start recording now, at rate of 2 msec
BG XY	Begin motion
#A;JP #A,RC=1	Loop until done
MG “DONE”	Print message
EN	End program
#PLAY	Play back
N=0	Initial Counter
JP# DONE,N>300	Exit if done
N=	Print Counter
X POS[N]=	Print X position
Y POS[N]=	Print Y position
XERR[N]=	Print X error
YERR[N]=	Print Y error
N=N+1	Increment Counter
#DONE	Done

DEALLOCATING ARRAY SPACE

Array space may be deallocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

Input of Data (Numeric and String)

The command, IN, is used to prompt the user to input numeric or string data. Using the IN command, the user may specify a message prompt by placing a message in quotations. When the controller executes an IN command, the controller will wait for the input of data. The input data is assigned to the specified variable or array element.

INSTRUCTION	INTERPRETATION
IN “Enter Length”, LENX	Use input command, IN, to query the user

In this example, the message “Enter Length” is displayed on the computer screen. The controller waits for the operator to enter a value. The operator enters the numeric value which is assigned to the variable, LENX.

CUT-TO-LENGTH PROGRAM EXAMPLE

In this example, a length of material is to be advanced a specified distance. When the motion is complete, a cutting head is activated to cut the material. The length is variable, and the operator is prompted to input it in inches. Motion starts with a start button which is connected to input 1.

The load is coupled with a 2 pitch lead screw. A 2000 count/rev encoder is on the motor, resulting in a resolution of 4000 counts/inch. The program below uses the variable LEN, to length. The IN command is used to prompt the operator to enter the length, and the entered value is assigned to the variable LEN.

INSTRUCTION	INTERPRETATION
#BEGIN	LABEL
AC 800000	Acceleration
DC 800000	Deceleration
SP 5000	Speed
LEN=3.4	Initial length in inches
#CUT	Cut routine
AI1	Wait for start signal
IN “enter Length(IN)”, LEN	Prompt operator for length in inches
PR LEN *4000	Specify position in counts
BGX	Begin motion to move material
AMX	Wait for motion done
SB1	Set output to cut
WT100;CB1	Wait 100 msec, then turn off cutter
JP #CUT	Repeat process
EN	End program

INPUT OF DATA (NUMERIC AND STRING)

OPERATOR DATA ENTRY MODE

The Operator Data Entry Mode permits data to be entered at anytime. In this mode, the input will not be interpreted as SSC commands and input such as ST or JG will not be recognized as commands. In this mode, the SSC provides a buffer for receiving characters. This mode may only be used when executing an applications program.

The Operator Data Entry Mode may be specified for either Port 1 or Port 2 or both. The mode may be exited with the \ or <escape> key.

NOTE: This is not used for high rate data transfer.

For Port 1: Use the third field of the CI command to set the Data Mode. A 1 specifies Operator Data Mode, a 0 disables the Data Mode.

For Port 2: Use the third field of the CC command to set the Data Mode. A 0 configures P2 as a general port for the Operator Data Mode.

To capture and decode characters in the Operator Data Mode, the SSC provides four special keywords for Port 1 (P1) and Port 2 (P2).
Port 1 (Main) Keyword Port 2 (Aux) Keyword Function

PORT 1 (MAIN) KEYWORD	PORT 2 (AUX) KEYWORD	FUNCTION
P1CH	P2CH	Contains the last character recieved
P1ST	P2ST	Contains the recieved string
P1NM	P2NM	Contains the recieved number
P1CD	P2CD	Contains the status code: -1 mode disabled 0 nothing recieved 1 recieved character, but not <enter> 2 recieved string, not a number 3 recieved number

Note: The value of P1CD and P2CD returns to zero after the corresponding string or number is read.

These keywords may be used in an applications program to decode data. They may be used in conditional statements with logical operators.

KEYWORD EXAMPLES:

INSTRUCTION	INTERPRETATION
JP #LOOP,P2CD< >3	Checks to see if status code is 3 (number received)
JP #P,P1CH="V"	Checks if last character received was a V
PR P2NM	Assigns received number to position
JS #XAXIS,P1ST="X"	Checks to see if received string is X

USING COMMUNICATION INTERRUPT

The SSC provides a special interrupt for communication allowing the application program to be interrupted by input from the user. The interrupt is enabled using the CI command. The syntax for the command is

INSTRUCTION	INTERPRETATION
CI m,n,o:	
m=0	Don't interrupt Port 1
1	Interrupt on <enter> Port 1
2	Interrupt on any character Port 1
-1	Clear any characters in buffer
n=0	Don't interrupt Port 2
1	Interrupt on <enter> Port 2
2	Interrupt on any character Port 2
-1	Clear any characters in buffer
o=0	Disable operator data mode for P1
1	Enable operator data mode for P1

The #COMINT label is used for the communication interrupt. For example, the SSC can be configured to interrupt on any character received on Port 2. The #COMINT subroutine is entered when a character is received and the subroutine can decode the characters. At the end of the routine the EN command is used. EN,1 will re-enable the interrupt and return to the line of the program where the interrupt was called, EN will just return to the line of the program where it was called without re-enabling the interrupt. As with any automatic subroutine, a program must be running in thread 0 at all times for it to be enabled.

INPUT OF DATA (NUMERIC AND STRING)

COMMUNICATION INTERRUPT PROGRAM EXAMPLE

Using the #COMINT Routine:

A SSC is used to jog the X and Y axis. This program automatically begins upon power-up and allows the user to input values from the main serial port terminal. The speed of either axis may be changed during motion by specifying the axis letter followed by the new speed value. An S stops motion on both axes.

COMMAND	INTERPRETATION
#AUTO	Label for Auto Execute
SPEEDX=10000	Initial X speed
SPEEDY=10000	Initial Y speed
CI 2,,1	Set Port 1 for Character Interrupt
JG SPEEDX,SPEEDY	Specify jog mode speed for X and Y axis
BGXY	Begin motion
#PRINT	Routine to print message to terminal
MG “TO CHANGE SPEEDS”	Print message
MG “TYPE X OR Y”	
MG “TYPE S TO STOP”	
#JOGLOOP	Loop to change Jog speeds
JG SPEEDX,SPEEDY	Set new jog speed
JP #JOGLOOP	
EN	End of main program
#COMINT	Interrupt routine
CI0	Clear interrupt
JP #A,P1CH=”X”	Check for X
JP #B,P1CH=”Y”	Check for Y
JP #C,P1CH=”S”	Check for S
ZS1;CI2;JP#JOGLOOP	Jump if not X,Y,S
#A;JS#NUM	
SPEEDX=VAL	New X speed
ZS1;CI2;JP#PRINT	Jump to Print
#B;JS#NUM	
SPEEDY=VAL	New Y speed
ZS1;CI2;JP#PRINT	Jump to Print
#C;ST;AMX;CI-1	Stop motion on S
MG{^8}, “THE END”	
ZS;EN,1	End-Re-enable interrupt
#NUM	Routine for entering new jog speed
MG “ENTER”,PICH{S},”AXIS }	Prompt for value
SPEED” {N	

```
#NUMLOOP; CI-1          Check for enter
#NMLP                   Routine to check input from terminal
JP #NMLP,P1CD<2         Jump to error if string
JP #ERROR,P1CD=2        Read value
VAL=P1NM
EN                      End subroutine
#ERROR;CI-1             Error Routine
MG "INVALID-TRY AGAIN"  Error message
JP #NMLP
EN                      End
```

INPUTTING STRING VARIABLES

String variables with up to six characters may be input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted. For example, IN "Enter X,Y or Z", V{S} specifies a string variable to be input.

The SSC, stores all variables as 6 bytes of information. When a variable is specified as a number, the value of the variable is represented as 4 bytes of integer and 2 bytes of fraction. When a variable is specified as a string, the variable can hold up to 6 characters (each ASCII character is 1 byte). When using the IN command for string input, the first input character will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations, see Bit-Wise Operators, page 9-24.

OUTPUT OF DATA (NUMERIC AND STRING)

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

SENDING MESSAGES

Messages may be sent out of the serial ports using the message command, MG. This command sends specified text and numerical or string data in ASCII format.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

INSTRUCTION

```
MG "The Final Value is", RESULT
```

INPUT OF DATA (NUMERIC AND STRING)

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
INSTRUCTION
MG "Analog input is", @AN[1]
MG "The Gain of X is", _GNX
```

Specifying the Serial Port for Messages:

By default, messages will be sent through port 1, the main serial port. The serial port can be specified with the specifier, {P1} for the main serial port and {P2} for auxiliary serial port thus:

```
INSTRUCTION
MG {P2} "Hello World"
```

Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 through 6. For example:

```
INSTRUCTION
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example:

MG "The Final Value is", RESULT {F5.2}

If the value of the variable RESULT is equal to 4.1, this statement returns the following:

The Final Value is 00004.10

If the value of the variable RESULT is equal to 999999.999, the above message statement returns the following:

The Final Value is 99999.99

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
INSTRUCTION
#A
JG 50000;BGX;ASX
MG "The Speed is", _TVX {F5.1} {N}
MG "counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

The speed is 50000 counts/sec

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

```
INSTRUCTION
MG {^07} {^255}
```

sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions:

FUNCTION	DESCRIPTION
" "	Surrounds text string
{ Fn.m }	Formats numeric values in decimal n digits to the right of the decimal point and m to the left.
{N}	Suppresses carriage return/line feed
{P1} or {P2}	Send message to main port or auxilliary port
{Sn}	Sends the first n characters a string variable, where n is 1 through 6
{$n.m}	Formats numeric vcalues in hexidecimal
{^n}	Sends ASCII character specified by interger n

DISPLAYING VARIABLES AND ARRAYS

Variables and arrays may be sent to the screen using the format, VARIABLE= or ARRAY[X]=. For example, V1= , returns the value of V1. These values may also be displayed using the message command, MG. If a variable was not previously defined, using the command, VARIABLE=, will cause the variable to be defined and the controller will not return an error. If the MG command is used to display a variable which has not been defined, the controller will return an error.

EXAMPLE

Printing a Variable and an Array element

INSTRUCTION	INTERPRETATION
#DISPLAY	Label
DM POSX[7]	Define Array POSX with 7 entries
PR 1000	Position Command
BGX	Begin
AMX	After Motion
V1=_TPX	Assign Variable V1
POSX[1]=_TPX	Assign the first entry
V1=	Print V1

FORMATTING THE RESPONSE OF INTERROGATION COMMANDS

The command, PF, can change format of the values returned by the interrogation commands:

INSTRUCTION
DP?
ER?
PA?
PR?
TE
TP

The numeric values may be formatted in decimal or hexadecimal* with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 through 10) and n is the number of digits to the right of the decimal point (0 through 4).

A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Examples:

INSTRUCTION	INTERPRETATION
DP21	Define position
TPX	Tell position
0000000021	Response from controller with default format
PF4	Change format to 4 places
TPX	Tell position
0021	Response from controller with new format
PF-4	Change to hexadecimal format
TPX	Tell Position
$0015	Response from controller in hexadecimal format
PF2	Format 2 places
TPX	Tell Position
99	Response from controller - returns 99 if position greater than 99

Global Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

INSTRUCTION
VF m.n

where m is the number of digits to the left of the decimal point (0 through 10) and n is the number of digits to the right of the decimal point (0 through 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4 Hex values are returned preceded by a $ and in 2's complement.

INPUT OF DATA (NUMERIC AND STRING)

INSTRUCTION	INTERPRETATION
V1=10	Assign V1
V1=	Return V1
0000000010.0000	Response from controller with default format
VF2.2	Change format
V1=	Return V1
10.00	Response from controller with new format
VF-2.2	Specify hex format
V1=	Return V1
$0A.00	Response from controller in hexadecimal format
VF1	Change format
V1=	Return V1
9	Response from controller - returns 9 if value greater than 9

Local Formatting of Variables

PF and VF commands are global format commands that effect the format of all relevant returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal. For example:

INSTRUCTION	INTERPRETATION
V1=10	Assign V1
V1=	Return V1
0000000010.0000	Response from controller with default format
V1={F4.2}	Specify local format
0010.00	Response from controller with new format
V1={$4.2}	Specify hex format
$000A.00	Response from controller in hexadecimal format
V1="ALPHA"	Assign string "ALPHA" to V1
V1={S4}	Specify string format first 4 characters
ALPH	Response from controller in string format

The local format is also used with the MG command.

CONVERTING TO USER UNITS

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The SSC position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec^2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

EXAMPLE

Converting to User Units

INSTRUCTION	INTERPRETATION
#RUN	Label
IN “ENTER # OF REVOLUTIONS”,N1	Prompt for revs
PR N1*2000	Convert to counts
IN “ENTER SPEED IN RPM”,S1	Prompt for RPMs
SP S1*2000/60	Convert to counts/sec
IN “ENTER ACCEL IN RAD/SEC2”,A1	Prompt for ACCEL
AC A1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

Programmable Hardware I/O

DIGITAL OUTPUTS

The SSC has an 8-bit uncommitted output port for controlling external events. Each bit on the output port may be set and cleared with the Software instructions SB (Set Bit) and CB(Clear Bit), or OB (define output bit).

EXAMPLES

Using Set Bit and Clear Bit Commands (SB, CB)

INSTRUCTION	INTERPRETATION
SB6	Sets bit 6 of output port
CB4	Clears bit 4 of output port

PROGRAMMABLE HARDWARE I/O

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Using the Output Bit Command (OB)

INSTRUCTION	INTERPRETATION
OB1, POS	Set Output 1 if the variable POS is non zero. Clear Output 1 if POS equals 0.
OB 2, @IN [1]	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
OB 3, @IN [1]&@IN [2]	Set Output 3 only if Input 1 and Input 2 are high.
OB 4, COUNT [1]	Set Output 4 if element 1 in the array COUNT is non-zero.

The output port can be set by specifying an 8-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 8-bit output port, where 2^0 is output 1, 2^1 is output 2 and so on. A 1 designates that the output is on.

Using the Output Port Command (OP)

INSTRUCTION	INTERPRETATION
OP6	Sets outputs 2 and 3 of output port to high. All other bits are 0.
OP0	Clears all bits of output port to zero
OP 255	Sets all bits of output port to one.

Using OP to Turn on Output After Move

INSTRUCTION	INTERPRETATION
#OUTPUT	Label
PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

DIGITAL INPUTS

The SSC has eight digital inputs for controlling motion by local switches. The @IN[n] function returns the logic level of the specified input 1 through 8. For example, a Jump on Condition instruction can be used to execute a sequence if a high condition is noted on an input 3. To halt program execution, the After Input (AI) instruction waits until the specified input has occurred.

EXAMPLES

Using the AI command:

INSTRUCTION	INTERPRETATION
JP #A,@IN[1]=0	Jump to A if input 1 is low
JP #B,@IN[2]=1	Jump to B if input 2 is high
AI 7	Wait until input 7 is high
AI -6	Wait until input 6 is low

Start Motion on Switch

Motor X must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor X must stop turning.

Solution: Connect panel switch to input 1 of SSC. High on input 1 means switch is in on position.

INSTRUCTION	INTERPRETATION
#S;JG 4000	Set speed
AI 1;BGX	Begin after input 1 goes high
AI -1;STX	Stop after input 1 goes low
AMX;JP #S	After motion, repeat
EN;	

INPUT INTERRUPT FUNCTION

The SSC provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. A 1 designates that input to be enabled for an interrupt, where 2^0 is bit 1, 2^1 is bit 2 and so on. For example, II,,5 enables inputs 1 and 3 ($2^0 + 2^2 = 5$).

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements.

IMPORTANT: Use the RI instruction (not EN) to return from the #ININT subroutine.

PROGRAMMABLE HARDWARE I/O

EXAMPLE

Input Interrupt

INSTRUCTION	INTERPRETATION
#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on X and Y axes
BG XY	Begin motion on X and Y axes
#B	Label #B
TP XY	Report X and Y axes positions
WT 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#ININT	Interrupt subroutine
MG “Interrupt occurred”	Display message
ST XY	Stops motion on X and Y axes
#LOOP	
JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG XY	Begin motion on X and Y axes
RI	Return from Interrupt subroutine

ANALOG INPUTS

The SSC provides seven analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 7. The resolution of the Analog-to-Digital conversion is 12 bits. Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

EXAMPLES

Position Follower (Point-to-Point)

Objective: The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

Method: Read the analog input and command X to move to that point.

INSTRUCTION	INTERPRETATION
#Points	Label
SP 7000	Speed
AC 80000;DC 80000	Acceleration
#Loop	
VP=@AN[1]*1000	Read analog input and compute position
PA VP	Command position
BGX	Start motion
AMX	After completion
JP #Loop	Repeat
EN	End

Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

INSTRUCTION	INTERPRETATION
#Cont	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGX	Start motion
#Loop	
VP=@AN[1]*1000	Compute desired position
VE=VP-_TPX	Find position error
VEL=VE*20	Compute velocity
JG VEL	Change velocity
JP #Loop	Change velocity
EN	End

Application Programming Examples

WIRE CUTTER

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals:

$$4000/2\pi = 637 \text{ count/inch}$$

APPLICATION PROGRAMMING EXAMPLES

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to input 1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Fig. 9.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

INSTRUCTION	FUNCTION
#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGX	Start Motion
AMX	After motion is complete
SB1	Set output bit 1
WT 20	Wait 20 ms
CB1	Clear output bit 1
WT 80	Wait 80 ms
JP #A	Repeat the process

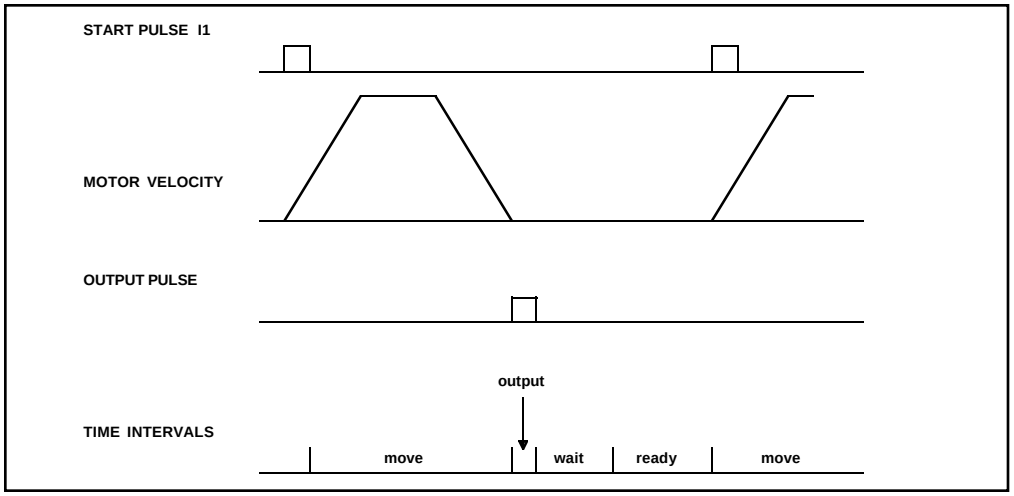


Fig. 9.1 - Motor Velocity and the Associated input/output signals

X-Y CONTROLLER

An X-Y-Z system must cut the pattern shown in Fig. 9.2. The X-Y axis moves the plate while the Z-axis raises and lowers the cutting tool.

The solid curves in Fig. 9.2 indicate sections where cutting takes place. Those must be performed at a feedrate of 1 inch per second. The dashed line corresponds to non-cutting moves and should be performed at 5 inch per second. The acceleration rate is 0.1 g.

The motion starts at point A, with the Z-axis raised. An X-Y motion to point B is followed by lowering the Z-axis and performing a cut along the circle. Once the circular motion is completed, the Z-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution. This results in the relationship:

1 inch = 40,000 counts
and the speeds of
1 in/sec = 40,000 count/sec
5 in/sec = 200,000 count/sec

an acceleration rate of 0.1g equals

0.1g = 38.6 in/s2 = 1,544,000 count/s2

Note that the circular path has a radius of 2” or 80000 counts, and the motion starts at the angle of 270° and traverses 360° in the CW (negative direction). Such a path is specified with the instruction

CR 80000,270,-360

Further assume that the Z must move 2” at a linear speed of 2” per second. The required motion is performed by the following instructions:

APPLICATION PROGRAMMING EXAMPLES

INSTRUCTION	INTERPRETATION
#A	Label
VM XY	Vector move for XY
VP 160000,160000	Positions
VE	End Vector Motion
VS 200000	Vector Speed
VA 1544000	Vector Acceleration
BGS	Start Motion
AMS	When motion is complete
PR,,-80000	Move Z down
SP,,80000	Z speed
BGZ	Start Z motion
AMZ	Wait for completion of Z motion
CR 80000,270,-360	Circle
VE	
VS 40000	Feedrate
BGS	Start circular move
AMS	Wait for completion
PR,,80000	Move Z up
BGZ	Start Z move
AMZ	Wait for Z completion
PR -21600	Move X
SP 20000	Speed X
BGX	Start X
AMX	Wait for X completion
PR,,-80000	Lower Z
BGZ	
AMZ	
CR 80000,270,-360	Z second circle move
VE	
VS 40000	
BGS	
AMS	
PR,,80000	Raise Z
BGZ	
AMZ	
VP -37600,-16000	Return XY to start
VE	
VS 200000	
BGS	
AMS	
EN	

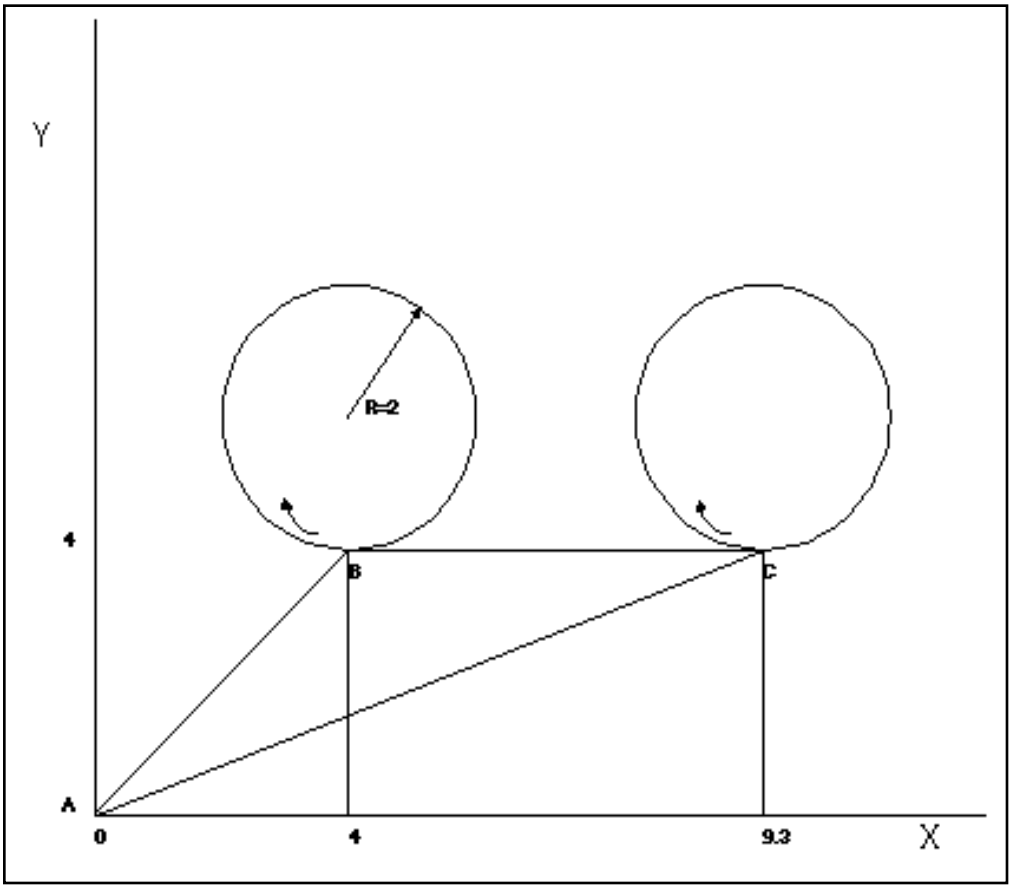


Figure 9.2 - Motor Velocity and the Associated input/output signals

SPEED CONTROL BY JOYSTICK

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as

Speed = 20000 x VIN

APPLICATION PROGRAMMING EXAMPLES

The corresponding velocity for the motor is assigned to the VEL variable.

INSTRUCTION	INTERPRETATION
#A	Label
JG0	Set jog speed of zero
BGX	Begin jogging (at speed zero)
#B	Label
VIN=@AN[1]	Set variable, VIN, to value of analog input 1
VEL=VIN*20000	Set variable, VEL to multiple of variable of VIN
JG VEL	Update jog speed to value of variable VEL
JP #B	Loop back to label, #B
EN	End

POSITION CONTROL BY JOYSTICK

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the joystick voltage is 5 Volts, corresponding to 1028 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

INSTRUCTION	INTERPRETATION
#A	Label
V3=5	Initial position ratio
DP0	Define the starting position
JG0	Set motor in jog mode as zero
BGX	Start
#B	
V1=@AN[1]	Read analog input
V2=V1*V3	Compute the desired position
V4=V2-_TPX-_TEX	Find the following error
V5=V4*20	Compute a proportional speed
JG V5	Change the speed
JP #B	Repeat the process
EN	End

BACKLASH COMPENSATION BY SAMPLED DUAL-LOOP

The continuous dual loop, enabled by the DV1 function is an effective way to compensate for backlash. In some cases, however, when the backlash magnitude is large, it may be difficult to stabilize the system. In those cases, it may be easier to use the sampled dual loop method described below.

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a leadscrew. Such a leadscrew has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The basic dilemma is where to mount the sensor. If you use a rotary sensor, you get a 4 micron backlash error. On the other hand, if you use a linear encoder, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, where we use two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash) whereas the linear sensor provides accurate load position information. The operation principle is to drive the motor to a given rotary position near the final point. Once there, the load position is read to find the position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of +/-2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

APPLICATION PROGRAMMING EXAMPLES

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position is only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below +/- 2 counts. Often, this is performed in one correction cycle.

EXAMPLE

Backlash Compensation by Sampled Dual Loop

INSTRUCTION	INTERPRETATION
#A	Label
DP0	Define starting positions as zero
LINPOS=0	
PR 1000	Required distance
BGX	Start motion
#B	
AMX	Wait for completion
WT 50	Wait 50 msec
LIN POS = _DEX	Read linear position
ER=1000-LINPOS-_TEX	Find the correction
JP #C,@ABS[ER]<2	Exit if error is small
PR ER	Command correction
BGX	Begin motion on X axis
JP #B	Repeat the process
#C	Label
EN	End program

Hardware & Software Protection

Introduction

The SSC provides several hardware and Software features to check for error conditions and to inhibit the motor on error. These features help protect the various system components from damage.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Since the SSC is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the SSC. Tol-O-Matic shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The SSC includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

OUTPUT PROTECTION LINES

Amp Enable - This signal goes low when the motor off command is given, when the position error exceeds the value specified by the Error Limit (ER) command, or when off-on-error condition is enabled (OE1) and the abort command is given. Each axis amplifier has separate amplifier enable lines. This signal also goes low when the watch-dog timer is activated, or upon reset. Note: The standard configuration of the AEN signal is TTL active low.

INPUT PROTECTION LINES

Abort - A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

Forward Limit Switch - Low input inhibits motion in forward direction. If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

HARDWARE PROTECTION

Reverse Limit Switch - Low input inhibits motion in reverse direction. If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

Software Protection

PROGRAMMABLE ERROR LIMIT

The SSC provides a programmable error limit for servo operation. The error limit can be set for any number between 1 and 32767 using the ER n command. The default value for ER is 16384.

Example:

ER 200,300,400,500 Set X-axis error limit for 200, Y-axis error limit to 300, Z-axis error limit to 400 counts, W-axis error limit to 500 counts

ER,1,,10 Set Y-axis error limit to 1 count, set W-axis error limit to 10 counts.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the SSC will generate several signals to warn the host system of the error condition. These signals include:

SIGNAL OR FUNCTION	INDICATION OF ERROR
# POSERR	Jumps to automatic excess position error subroutine
Error Light	Turns on when position error exceeds error limit
OE Function	Shuts motor off by setting AEN output line low if OE1.

The position error of X,Y,Z and W can be monitored during execution using the TE command.

PROGRAMMABLE POSITION LIMITS

The SSC provides programmable forward and reverse position limits. These are set by the BL and FL Software commands. Once a position limit is specified, the SSC will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example - Using Position Limits

INSTRUCTION	INTERPRETATION
DP0,0,0	Define Position
BL -2000,-4000,-8000	Set Reverse position limit
FL 2000,4000,8000	Set Forward position limit
JG 2000,2000,2000	Jog
BG XYZ	Begin
(motion stops at forward limits)	

OFF-ON-ERROR

The SSC controller has a built in function which can turn off the motors under certain error conditions. This function is know as “Off-On-Error”. To activate the OE function for each axis, specify 1 for X,Y,Z and W axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER

2. The abort command is given

3. The abort input is activated with a low signal.

Note: If the motors are disabled while they are moving, they may ‘coast’ to a stop because they are no longer under servo control.

To re-enable the system, use the Reset (RS) or Servo Here (SH) command.

Examples - Using Off-On-Error

INSTRUCTION	INTERPRETATION
OE 1,1,1,1	Enable off-on-error for X,Y,Z and W
OE 0,1,0,1	Enable off-on-error for Y and W axes and disable off-on-error for X and Z axes

SOFTWARE PROTECTION

AUTOMATIC ERROR ROUTINE

The #POSERR label causes the statements following to be automatically executed if error on any axis exceeds the error limit specified by ER. The error routine must be closed with the RE command. The RE command returns from the error subroutine to the main program.

NOTE: The Error Subroutine will be entered again unless the error condition is gone.

Example - Using Automatic Error Subroutine

INSTRUCTION	INTERPRETATION
#A;JP #A;EN	"Dummy" program
#POSERR	Start error routine on error
MG "error"	Send message
SB 1	Fire relay
STX	Stop motor
AMX	After motor stops
SHX	Servo motor here to clear error
RE	Return to main program

NOTE: An applications program must be executing for the #POSERR routine to function.

LIMIT SWITCH ROUTINE

The SSC provides forward and reverse limit switches which inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated and that axis motor is moving in that direction. The RE command ends the subroutine.

The state of the forward and reverse limit switches may also be tested during the jump-on-condition statement. The _LR condition specifies the reverse limit and _LF specifies the forward limit. X,Y,Z, or W following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Example - Using Limit Switch Subroutine

INSTRUCTION	INTERPRETATION
#A;JP #A;EN	Dummy Program
#LIMSWI	Limit Switch Utility
V1=_LFX	Check if forward limit
V2=_LRX	Check if reverse limit
JP#LF,V1=0	Jump to #LF if forward
JP#LR,V2=0	Jump to #LR if reverse
JP#END	Jump to end
#LF	#LF
MG “FORWARD LIMIT”	Send message
STX;AMX	Stop motion
PR-1000;BGX;AMX	Move in reverse
JP#END	End
#LR	#LR
MG “REVERSE LIMIT”	Send message
STX;AMX	Stop motion
PR1000;BGX;AMX	Move forward
#END	End
RE	Return to main program

NOTE: An applications program must be executing for #LIMSWI to function.

Notes:

Troubleshooting

Overview

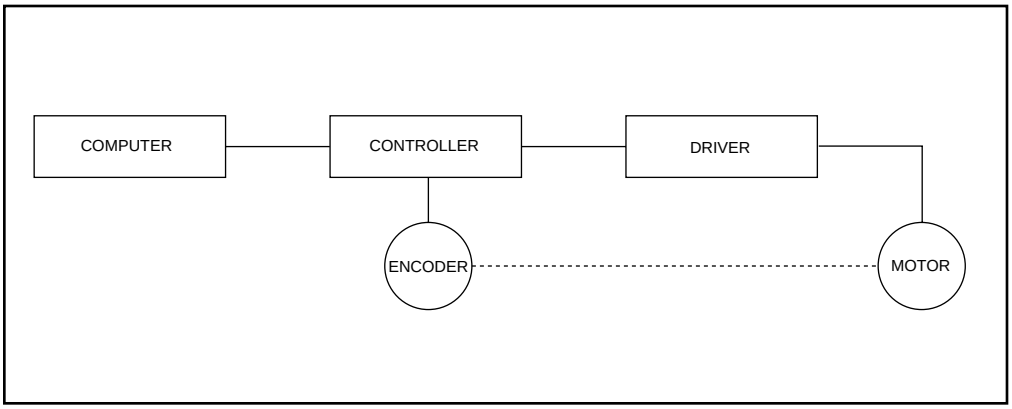
The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Communication
3. Stability and Compensation
4. Operation

The various symptoms along with the cause and the remedy are described in the following tables.

Installation



Refer Back to Basic schematics and Chapter 2: Set Up to ensure correct installation.

Communication

SYMPTOM	CAUSE	REMEDY
Using Tol-O-Motion, cannot communicate with controller.	Baud rate is not correctly configured.	Check Baud rate switch positions and registry settings for software setup
	Hardware handshaking is not enabled (must be enabled to use Tol-O-Motion software)	Set Hardware HSHK DIP switch
	Serial Cable is not correct	Use serial cable supplied by Tol-O-Matic or check pinouts for cable (see appendix)
	Incorrect port is specified	Change registry setting from com port

Stability

SYMPTOM	CAUSE	REMEDY
Motor runs away when the loop is closed	Wrong feedback polarity	Invert the polarity of the loop by inverting the motor leads (brush type) or the encoder
Motor oscillates	Too high gain or too little damping	Decrease KI and KP. Increase KD

Operation

SYMPTOM	CAUSE	REMEDY
Controller rejects command. Responded with a ?	Invalid command	Interrogate the TC or TC1.
Motor does not complete move.	Noise on limit switch stops the motor.	To verify cause, check the stop code (SC). If caused by limit switch, reduce noise.
During a periodic operation, motor drifts slowly.	Encoder noise	Interrogate the position periodically. If the controller states that the position is different locations it implies encoder noise. Reduce noise. Use differential encoder inputs.
Same as above	Programming error	Avoid resetting position error at end of move with SH command.