ELECTROMATE

Galil Programming Manual Version 3.0

 www.electromate.com • sales@electromate.com
TOLL FREE PHONE (877)SERVO98 • TOLL FREE FAX (877)SERVO99

ELECTROMATE

TABLE OF CONTENTS

Program Flow:

Motion Programming:

Application Example:

Appendices:

Disclaimer

This guide is designed to assist users in understanding the Galil motion controller programming language, and by no means, should serve to replace the factory published Hardware User Manuals. This Manual should be used in conjunction with all other factory supplied documentation.

The production of this Reference Guide is a culmination of 15 years of Galil programming experience by the staff of Electromate. Considering the multitude of potential programming solutions, this guide should by no means serve as a complete programming guide, nor do we guarantee it's syntax accuracy.

Numerous application examples throughout this Reference Guide are specific to certain advanced level controllers, so please refer to the Command Summary Appendices for appropriate controller selection.

Programming Instruction Format

- Two-letter, English-like commands

- Must be upper case (CAP LOCK on)

- Commas separate X, Y, Z, W parameters

- Multiple commands per line allowable if separated by semi-colon (i.e. PR50;SP10;BG;TPX)

- All Galil controllers perform quadrature decoding of the encoder input signal to provide 4 X interpolation IE. the command to move a motor with a 500 ppr resolution encoder is PR 2000

- **Example**:

TP X	Tell position of X axis only
ST XW	Stop X and W axes
SP 100,200	Set Speed for X and Y axes
PR,,,4000	Set position for W axis only
BG	Begin motion on all axes

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ELECTROMATE

Tuning Parameters

Designates the proportional constant in the controller filter	KP
Designates the derivative constant in the controller filter	KD
Sets the integral gain of the control loop	KI
Sets the acceleration feedfordward coefficient	FA
Sets the velocity feedforward coefficient	FV
Sets a bias voltage in the motor command output	OF
Smoothing time constant which filters the acceleration & deceleration functions in independent/vector moves	IT / VT
Sets the gain of the control loop	GN
Sets the compensating zero function in the control loop	ZR
Sets the integral gain of the control loop	KI

Rule of Thumb: Tuning is a 'trial and error' process-

For systems with high oscillation, decrease GN and increase ZR in tandem or decrease KP and increase KD in tandem

For systems with underdamped conditions, increase GN and decrease ZR in tandem or increase KP and decrease KD in tandem

For systems with poor accuracy, gradually increase KI

For systems with high frequency ringing, increase KD

A typical tuning parameter set-up will have a KD of 5 to 10 times KP, and a KI =1 (or KI=2)

Motion Programming: Tuning Your Servo System

Adjusting the tuning parameters are required when using servo motors (standard or sinusoidal commutation). The system compensation provides fast and accurate response and the following presentation suggests a simple and easy way for compensation. More advanced design methods are available with software design tools from Galil, such as the Servo Design Kit (SDK software)

The filter has three parameters: the damping, KD; the proportional gain, KP; and the integrator, KI. The parameters should be selected in this order.

To start, set the integrator to zero with the instruction

Instruction	Interpretation
KI 0 (CR)	Integrator gain

and set the proportional gain to a low value, such as

Instruction	Interpretation
KP 1 (CR)	Proportional gain

Instruction	Interpretation
KD 100 (CR)	Derivative gain

For more damping, you can increase KD (maximum is 4095). Increase gradually and stop after the motor vibrates. A vibration is noticed by audible sound or by interrogation.
If you send the command

Instruction	Interpretation
TE X (CR)	Tell error

a few times, and get varying responses, especially with reversing polarity, it indicates system vibration. When this happens, simply reduce KD.

Next you need to increase the value of KP gradually (maximum allowed is 1023). You can monitor the improvement in the response with the Tell Error instruction

Instruction	Interpretation
KP 10 (CR)	Proportion gain

Instruction	Interpretation
TE X (CR)	Tell error

As the proportional gain is increased, the error decreases.

Again the system may vibrate if the gain is too high. In this case, reduce KP. Typically, KP should not be greater than KD/4 to KD/10 (Only when the amplifier is configured in the current mode).

Finally, to select KI, start with zero value and increase it gradually. The integrator eliminates the position error, resulting in improved accuracy. Therefore, the response to the instruction

Instruction	Interpretation
TE X (CR)	Tell error

becomes zero. As KI is increased, its effect is amplified and it may lead to vibrations. If this occurs, simply reduce KI. Repeat tuning for the Y, Z and W axes.

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Motion Programming: System Set-up

Here are a few examples for tuning and using your controller. These examples have remarks next to each command - these remarks must not be included in the actual program.

This example assigns the system filter parameters, error limits and enables the automatic error shut-off.

Instruction	**Interpretation**
KP10,10,10,10,10,10,10,10	Set gains for A, B, C, D, E, axes
KP10,10,10,10,10,10,10,10	Set gains for A, B, C, D, E, and H axes
KP*=10	Alternate method for setting gain on all axes
KPX=10	Alternate method for setting X (or A) axis gain
KPA=10	Alternate method for setting A (or X) axis gain

The X, Y, Z and W axes can also be referred to as the A, B, C, and D axis.

Instruction	**Interpretation'**
OE 1,1,1,1,1,1,1,1	Enable automatic Off on Error function for all axes
ER*=1000	Set error limit for all axes to 1000 counts
KP10,10,10,10,10,10,10,10	Set gains For A, B, C, D, E, and H axes
KP*=10	Alternate method for setting gain on all axes
KPX=10	Alternate method for setting X (or A) axis gain
KPA=10	Alternate method for setting A
KPZ=10	Alternate method for setting Z axis gain

KPD=10	Alternate method for setting D axis gain
KPH=10	Alternate method for setting H axis gain

ELECTROMATE

Motion Programming: Using the IT and VT Commands(S curve profiling):

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.

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 on the next page illustrates the effect of smoothing.

Note that the smoothing process results in longer motion time.

Example - Smoothing

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

ELECTROMATE

Motion Programming: Operation Under Torque Limit

The magnitude of the motor command may be limited independently by the instruction TL.

Instruction	**Interpretation**
TL 0.2	Set output limit of X axis to 0.2 volts
JG 10000	Set X speed
BG X	Start X motion

In this example, the X motor will probably not move since the output signal will not be sufficient to overcome the friction. If the motion starts, it can be stopped easily by a touch of a finger.

Increase the torque level gradually by instructions such as

Instruction	**Interpretation**
TL 1.0	Increase torque limit 1 volt
TL 9.98	Increase torque limit to maximum, 9.98 volts

The maximum level of 10 volts provides the full output torque.

Programming Environments

A variety of device drivers are available for most Galil controllers via the source code subdirectory located on the
COM-Disk, specifically:

- Visual Basic
- C, C++
- DOS
- Pascal, Turbo Pascal
- BASIC
- Windows 3.X/ 95/ 98/ NT/ 2000 and CE
- QNX
- OCX/ OLE Controls
- DLL'S
- Labview, Wonderware, Think And Do
- Delphi, HP VEE, Linux

A variety of translator software packages are also supported, including:

CAD (any DXF file) File Conversion
HPGL Code Conversion
Visual Basic/ Active X
ECAM
DDE

Motion Control Modes

Independent move	PR, PA
Jog Mode	JG
Vector Mode - Linear and Circular	VM
Linear Interpolation	LM
Contour Mode	CM
Electronic Gearing	GA
Electronic CAM	EA
Elliptical Move	ES
Gantry Mode	GM
Tangential Move	TN
Homing Commands	HM, FE, FI
S-Curve Move	IT, VT

ELECTROMATE

Motion Programming: Independent Axis Positioning

- Motion between specified axes is independent
- User specifies:

PR or PA	Relative or Absolute Position [counts]
SP	Slew speed [counts/sec]
AC	Acceleration [counts/sec^2]
DC	Deceleration
IT	S-Curve filter
BG	Begin

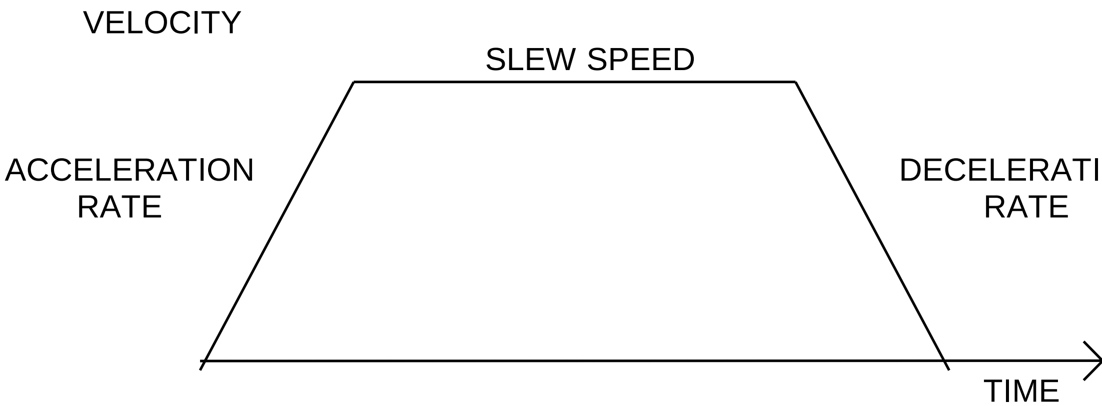
- BG command begins motion
- Can change SP, AC, DC during motion
- Can change direction during motion

- Example:

Instruction	**Interpretation**
PR 1000,4000	Specify relative position
SP 2500,2500	Specify slew speed
AC 100000,100000	Specify acceleration
DC 50000,50000	Specify deceleration
BG XY	Begin motion

Motion Programming: Independent Motion Commands

A motion controller can be "told" by a host computer to perform a move by any of the controlled motors. The most simple move is one with a trapezoidal velocity profile as illustrated below. This move is characterized by the parameters: distance, slew velocity, and acceleration and deceleration rates.



Trapezoidal Velocity Profile

The most basic form of specifying these motion parameters is by "units of position resolution." For example, let the encoder resolution be 4000 counts per turn and suppose the objective is to rotate the motor one revolution along a trapezoidal velocity, with a total motion time of 0.3 sec with acceleration and deceleration times of 0.1 sec each. Simple calculation indicates that the slew velocity equals 5 revolutions per second and that both acceleration and deceleration rates are 50 revolutions/sec^2.
The motion parameters can be expressed in terms of units of resolution as a distance of 4000 counts, a slew velocity of 20,000 counts/sec, and acceleration and deceleration rates of 200,000 counts/sec^2. The motion parameters must be transmitted from the host computer to the motion controller. The special instructions used by the controller, along with their interpretation, are given below.

Instruction	Interpretation
PR 4000	Relativedistance
SP 20000	Speed rate
AC 200000	Acceleration rate
DC 200000	Deceleration rate
BGX	Start motion of X motor

All that is needed is that the host computer sends the characters indicated by the program shown above and the motion starts immediately.

Another type of simple motion is the jog move. Here the motor is commanded to run indefinitely at a specified speed. The motion parameters in this case are limited to the speed, acceleration, and deceleration. These parameters can be expressed in units of resolution as illustrated below.

Consider a system where the resolution of the encoder is 4000 counts/turn and where the motor is required to run at a speed of 600 rpm, or 10 revolutions/sec, and must accelerate to that speed over 100 msec. Simple calculations show that the speed rate is 40,000 counts/sec and that the acceleration rate is 400,000 counts/sec^2.

The instructions used to generate such a motion, along with their interpretations, are shown in following example.

Instruction	**Interpretation**
JG 40000	Jog speed
AC 400000	Acceleration rate
BGX	Start X motion

Motion Programming: Stored Programs

The instructions to execute the motion can be issued directly from the host computer or external operator interface, resulting in an immediate move. This mode, the immediate mode, requires the continuous involvement of the interface. This requirement is often undesirable because the host may have to perform other functions simultaneously. An alternative method is to combine several motion commands into a complete application program which is downloaded to the controller memory. This method reduces or eliminates the involvement of the host computer. The controller can receive its instructions from stored programs. To start the motion, the host sends a short command, such as “Execute Program A.” The controller will then receive the instructions from its memory without the intervention of the host computer.

To illustrate the concept, consider the move described in the previous example. To perform the same move from a stored program, the host modifies the program by adding a label, #A, for example, and an end statement, EN. The resulting program is as follows:

Instruction	Interpretation
#A	Program label
PR 4000	Distance
SP 20000	Speed
AC 200000	Acceleration
DC 200000	Deceleration
BGX	Start X motion
EN	End of program

The host downloads this program to the controller memory, where the program remains inactive. To execute the program, the host sends the command:

XQ#A

which causes the controller to execute the program labeled #A.

The controller can receive commands from both the host computer and the stored program and execute these instructions simultaneously.

ELECTROMATE

Motion Programming: Jogging

• Changes motion 'on the fly'

• User specifies:

JG	Jog speed and direction
AC	Acceleration
DC	Deceleration
IT	S-Curve filter
BG	Begin
ST	Stop

• Can change JG, AC, DC during motion

• Example:

Instruction	Interpretation
JG - 1000	Specify jog speed
AC 200000	Specify acceleration rate
DC 200000	Specify deceleration rate
BG X	Begin motion

Motion Programming: Linear Interpolation

- For coordinated motion between 2 to 8 axes on advanced level controllers

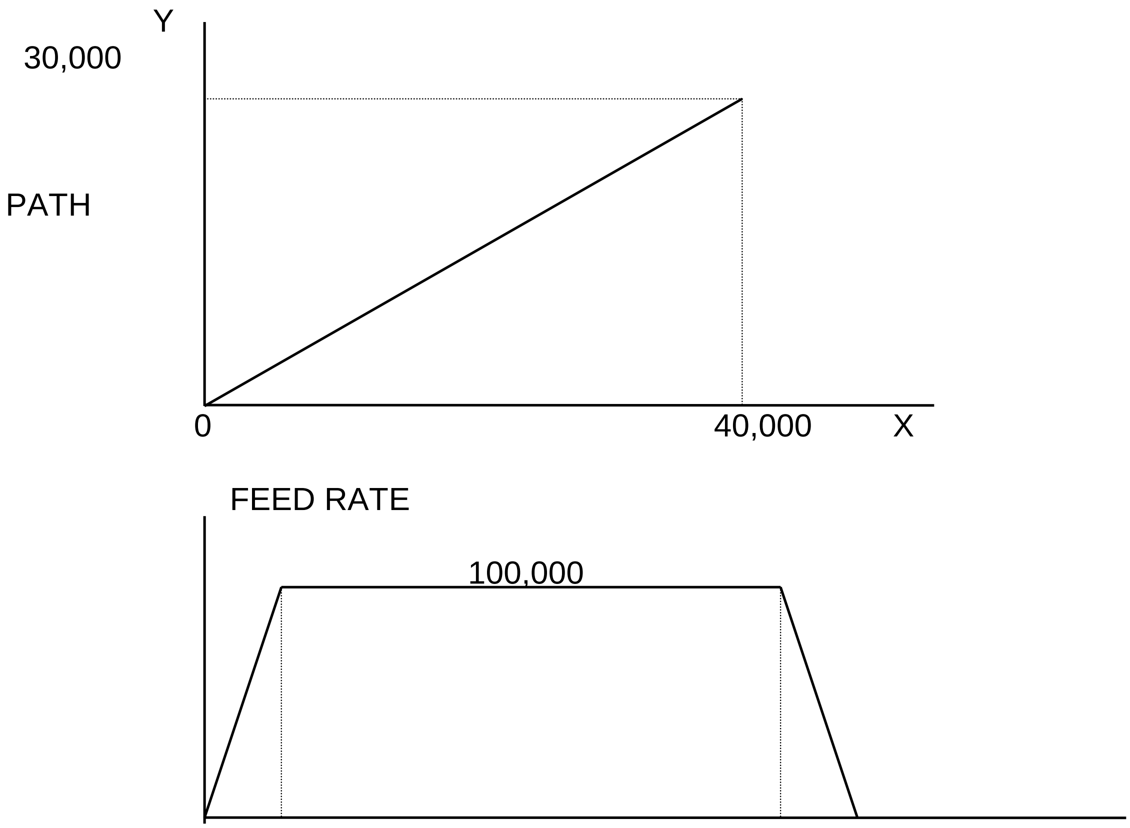
- Allows infinite paths

- User specifies:

LM	Linear Interpolated Axes
LI	Linear segments – incremental distances
VS	Vector speed
VA	Vector acceleration
VD	Vector deceleration
VT	S-curve filter
LE	End segment
BGS	Begin sequence

- Example:

Instruction	Interpretation
LM XYZW	Linear Interpolate on X,Y,Z,W
VS 5000	Specify vector speed
VA 100000	Specify vector acceleration
VD 100000	Specify vector deceleration
LI 100,200,300,400	Linear segment
LI -500,-200,0,300	Next linear segment
LE	End
BGS	Begin motion sequence



Application Example: Linear Interpolation

• Linear Interpolation Program:

VP 40000,30000	Specify vector path
VS 100000	Define feed rate vector velocity
VA 1000000	Define vector acceleration
VD 1000000	Define vector deceleration
VE	End of motion
BGS	Start motion

DMC-1200	**DMC-1600**	**DMC-1700**
DMC-1800	**DMC-2000**	

Motion Programming: Controlling Linear Interpolation from a Host Program

When operating in linear interpolation mode, up to 511 linear interpolation segments can be specified before execution. These segments are saved in the sequence buffer. As segments are executed, the sequence buffer allows for additional segments to be added. Using array space on the controller, it is possible to extend the linear interpolation buffer. The following sample program demonstrates how to use array space to hold linear interpolation points in 2 sets of arrays, called set A and set B. The maintenance program, #FILLPTS, monitors the sequence buffer. As space becomes available in the sequence buffer, the monitor program adds additional points from the arrays. The monitor program uses the A set of arrays first, then the B set of arrays. The set of arrays which are not being used for filling the sequence buffer can be updated with additional points by a host program running on the computer. This method allows for minimum supervision by the host program and quick information exchange by use of the download array function, QD.

This program uses 3 axes of linear interpolation as an example. This method could also be extended for additional axes or vector mode.

This monitor program resides in the controller. LI commands would be sent to start the move. The monitor program sets the state variables, AWAIT and BWAIT to 0 to let the host program know when it can download additional segments. The command QD can be used by the host program to download array elements to the controller while the move is executing. (First to the A arrays: XPOSA, YPOSA, ZPOSA then to the B arrays: XPOSB, YPOSB, ZPOSB). The last segment can be signified by including a final segment with length that is out of the normal operating range such as 1000000. When the program sees this value, it gives the last LI element and ends the program.

#FILLPTS	Buffer maintenance program.
DM XPOSA[1000],YPOSA[1000],ZPOSA[1000]	Define arrays
DM XPOSB[1000],YPOSB[1000],ZPOSB[1000]	
#MONITOR	Monitor Routine
JP#MONITOR, (_SCX<>100)	Loop until Vector Mode Executing
JP#MONITOR, (_SCY<>100)	Loop until Vector Mode Executing

JP#MONITOR, (_SCZ<>100)	Loop until Vector Mode Executing
CTR=0	Define counter variable
AWAIT=-1	
BWAIT=-1	Define buffer state variables
#FILLA	Routine to fill LI Buffer W/ arrays "A"
JP#FILLA,AWAIT=-1	Wait until buffer has been filled
AWAIT=1	Set buffer state variable

Motion Programming: Coordinated Motion Sequences

- For linear and circular interpolation on any set of two axes
- Motion between path segments is continuous
- User specifies:

VM	Plane of motion
VP	Linear segment
CR	Arc segment
VS	Vector speed
VA	Vector acceleration
VD	Vector deceleration
VT	S-Curve filter
VE	End segment
BGS	Begin sequence

- Up to 511 segments can be given prior to motion
- Can send additional segments during motion
- Can change VS and VA during motion

Coordinated Motion

Motion controllers can generate various types of motion. One of the most common types is the coordinated motion between two axes, for example, X and Y. In this instance, the controller generates motion of both motors in a manner that results in straight lines and circular arcs. This motion is quite common in computer numeric control (CNC) and other industrial applications.

Coordinated motion is defined by the path and by the velocity profile along the path. The first step is to select the two axes that define the motion plane using the VM instruction. For example, VMXY defines the plane of coordinated motion as the XY plane.

The second step defines the motion path, which consists of a collection of straight lines and circular arcs. Straight lines are defined with the instruction

VP m,n

where (m,n) are the coordinates of the endpoint. Circular arcs are defined with the instruction

CR R, θ, δ

where R indicates the radius, θ defines the starting angle, and δ defines the width of the arc.

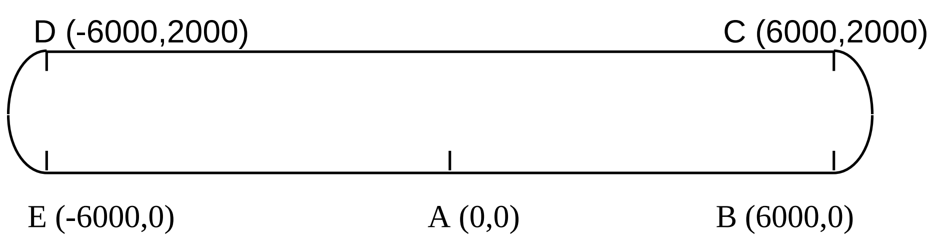
The complete path may consist of a collection of such motion segments, as illustrated by the following XY Racetrack example. There is no limit to the number of segments that can be specified and additional segments can be sent during motion. This allows the controller to control motion along very long paths without stopping.

In addition to the motion path, the user can specify the vector speed (feedrate). In most applications, the feedrate is set to a constant value. However, the velocity can also be reduced around corners, for example.

Similarly, the acceleration and deceleration rates along the motion can be specified. The instructions for the vector velocity, acceleration, and deceleration are VS, VA, and VD respectively.

The generation of a coordinated move is illustrated by the following example.

Consider the motion path described by the figure below and write a program to generate it. The motion is in the XY plane, the radius of the corners is 1000 counts, the vector speed is 20,000 count/sec, and the vector acceleration and deceleration rates are both 100,000 count/sec^2.



Application Example: XY Racetrack

The instructions and their interpretations are shown below.

Instruction	Interpretation
#M	Label
VM XY	Specify XY plane
VP 6000,0	Move to Point B
CR 1000,270,180	Move to Point C
VP -6000,2000	Move to Point D
CR 1000,90,180	Move to Point E
VP 0,0	Return to Point A
VE	End of path
VS 20000	Vector speed
VA 100000	Vector acceleration
VD 100000	Vector deceleration
BGS	Start motion
EN	End program

Motion Programming: Converting to User Units

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

The 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.

A sample program is found on the following page.

Application 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

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ELECTROMATE

Motion Programming: In User Units

• Position and speed scale factors are allowable for ease of programmability. User defined variables can also be used for the same function.

Example 1:

Load is coupled to a motor through a 10 pitch (10 turns per inch) leadscrew. Desired motion is 1 inch at slew speed of 3.75 inches per sec and acceleration of 100 inches/sec 2. Assume a 1000 line encoder.

PS = (1000 lines/rev)x(4 counts/line)x(10 rev/inch) = 40,000 counts/user_unit

SS = (3.75 in/sec)x(1000 lines/rev)x(4 counts/line)x(10 rev/inch) = 150,000 (counts/sec)/(user_unit/sec)

Motion Program
PS 40000
SS 150000
PR 1
SP 3.75
AC 100
BG

Example 2:

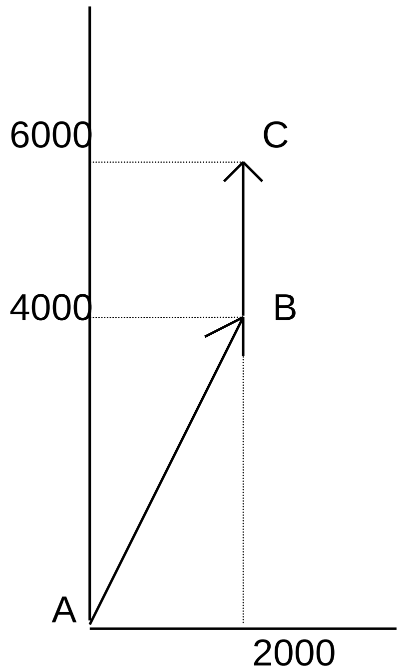
Run a motor a distance of 30 revolutions at a slew speed of 500 RPM and acceleration of 300 rev/sec^2 . Assume a 500 lines per rev encoder.

PS = (500 lines/rev)x(4 counts/line) = 2,000 counts/user_unit
SS = (500 lines/rev)x(4 counts/line)/(60 sec/min) = 33 (counts/sec)/(user_unit/min)

Motion Program
PS 2000
SS 33
AC 300
SP 500
PR 30
BG

ELECTROMATE

Motion Programming: Specifying Linear Segments

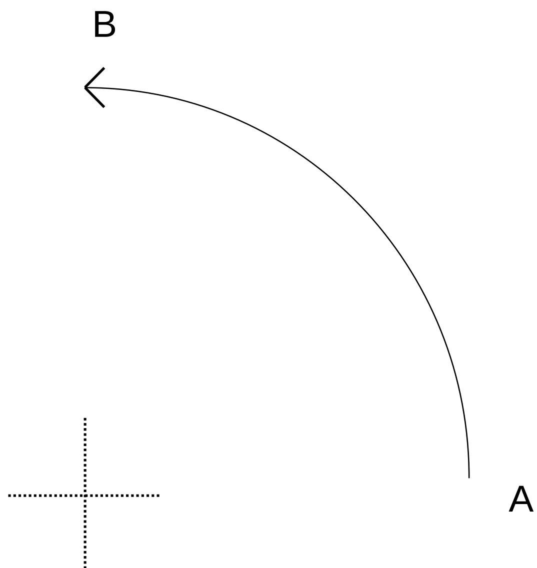


• Specify linear segments as destination coordinate with respect to start of move

Segment AB	VP 2000,4000
Segment BC	VP 2000,6000

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Motion Programming: Specifying Arc Segments

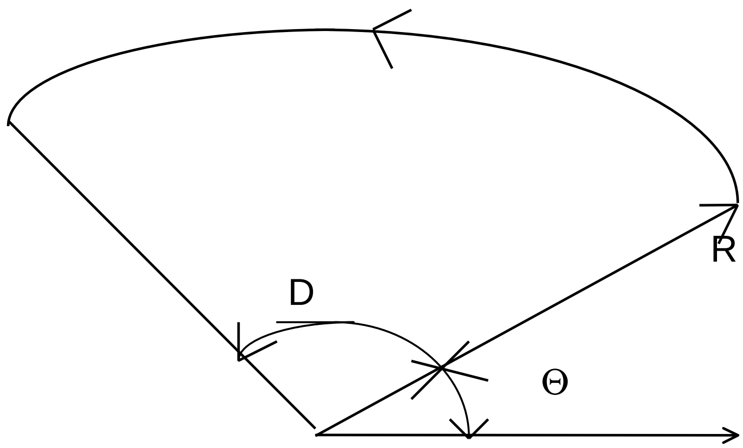


• Specify circular segment as radius, starting angle and travel angle. Clockwise rotation is negative travel.

Segment AB ⇒ CR 1000,0,90

Circular Interpolation

To Specify a circular arc:



R = radius
Θ = angle of radius at starting point
D = traversed angle

Example:

Instruction	Interpretation
CR 1000,45,120	Define circle of radius 1000 counts
X,Y,V,M	Specify Vector Motion Path
BGS	Begin sequence

ELECTROMATE

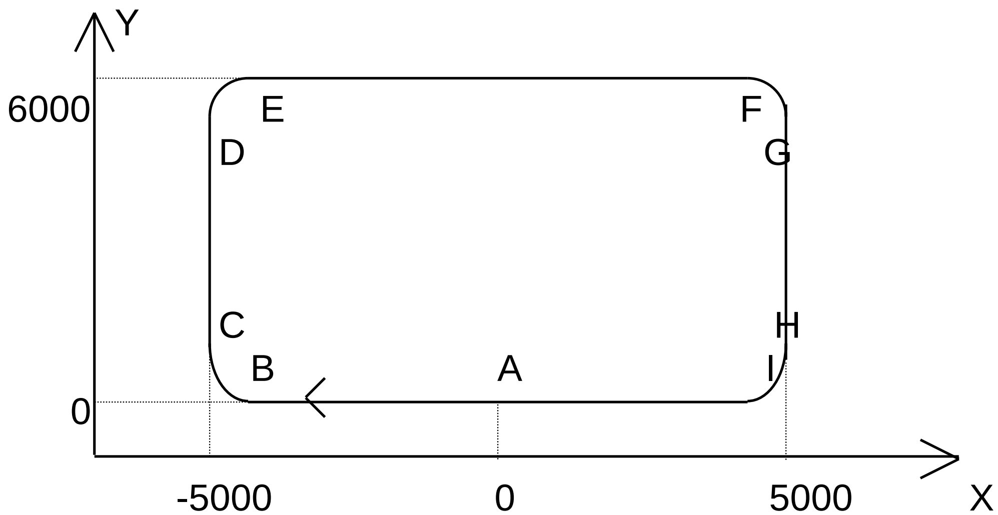
Application Example: Circular Interpolation

A to B
X_f = -4000, Y_f = 0
VP -4000,0

B to C
R = 1000
θ = 270°
D = -90°
CR 1000,270,-90

C to D
X_f = -5000
Y_f = 5000
VP -5000,5000

Instruction
VP -4000,0
CR 1000,270,-90
VP -5000,5000
CR 1000,180,-90
VP 4000,6000
CR 1000,90,-90
VP 5000,1000
CR 1000,0,-90
VP 0,0
VS 5000
VA 100000
VD 100000
VE
BGS



The Required Path

Radius of corners = 1000 counts

Motion Programming: Contour Mode

The contouring mode is the ideal vehicle for generating motion that is expressed as a function of time. Suppose, for example, that the motion involves three axes, X, Y, and Z, and that the time-dependent position functions are X(T), Y(T), and Z(T).

The procedure for generating motion starts with specifying the time interval, DT, and by evaluating the functions X, Y, and Z at those times. Later, the position increments, DX, DY, DZ, are computed and specified.

Note that the computations outlined above can be performed in the host computer, or optionally in advanced level controllers. The computation process is illustrated below.

The design objective is to drive an XY table along a spiral trajectory according to the following functions expressed in resolution counts.

$X(T) = T \cos 0.03T$
$Y(T) = T \sin 0.03T$
for $0<T<12,000$ ms

The time, T, is in milliseconds and the argument, .03T, is in degrees. As a result, the largest argument is 360° corresponding to one revolution.
To generate the motion, we start with the selection of the time interval, DT. Note that in the contour mode we approximate the path by straight line segments. If the time interval is DT ms, the width of each segment in degrees is

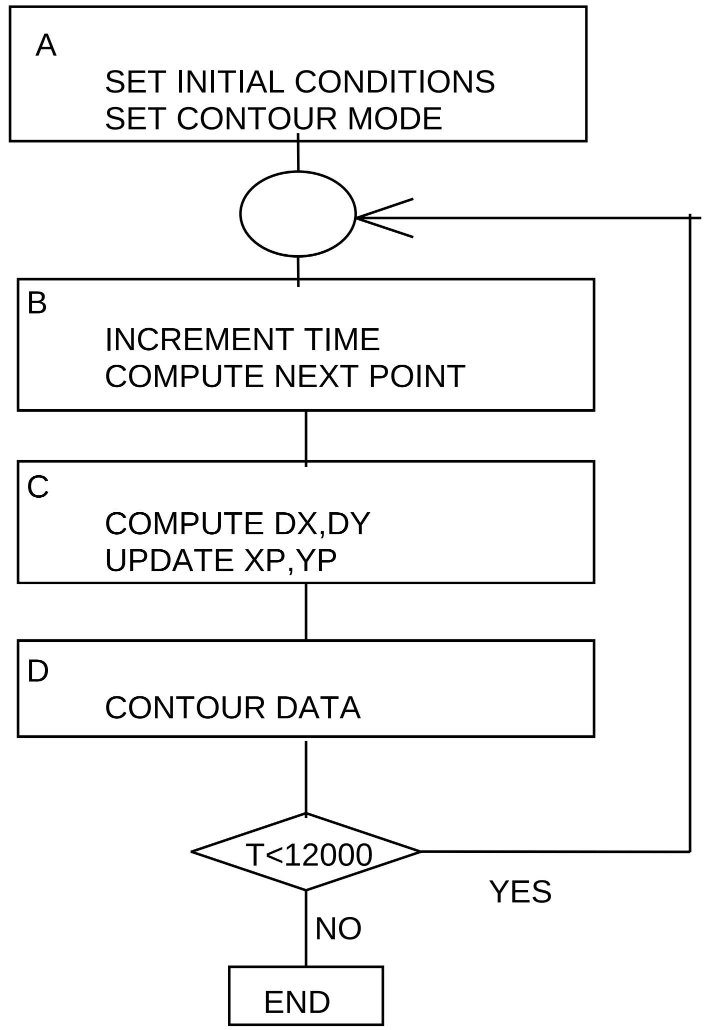
$$\alpha = 0.03DT$$

For example, if DT = 32 ms, the width of each segment is 0.96°, which is very precise. Since the total motion time is 12000 ms, the total number of increments is 375.

Now both X and Y are computed at the times 0,32,64,96...ms, to determine the required positions. Later, the increments between these points are computed.

Some motion controllers have the capability to perform the computation independently without host intervention. To illustrate the process, consider the operation by a DMC-1700 controller, as shown in the flowchart below and the following motion program.

The first block A sets the initial time, T = 0, and initial positions of X and Y. It also sets the controller in the contour mode with a time interval of 32 ms.



Flowchart for Spiral Trajectory Example

The second block, B, increments the time and computes X and Y. The values of X and Y may include integers and fractions. If the fractions are rounded differently in different parts of the operation, it may produce an error.

To avoid that possibility, it is advisable to limit the operation to the integer part of X and Y, which are denoted XC and YC.
The block C determines the increment as the difference between the current values, XC and YC, and the previous values, XP and YP. The increments are denoted DX and DY.

The computed increments are then specified as a motion command and the process is repeated until the time reaches its limit. The actual program and the interpretation are given on the following page.

ELECTROMATE

Application Example: Spiral Trajectory

Instruction	Interpretation
#A	Label
T = 0	Initial time
XP = 0	Initial value of X
YP = 0	Initial value of Y
CMXY	Set contour mode
DT 5	Time interval is 2^5 = 32 ms
#LOOP	Label
T = T+32	New time
A = T*0.03	Argument in degrees
X = @ cos[A] *T	Compute X
Y = @ sin[A] *T	Compute Y
XC = @ INT[X]	Integer part of X
YC = @ INT[Y]	Integer part of Y
DX = XC-XP	Increment of X
DY = YC-YP	Increment of Y
CD DX,DY	Command motion
WC	End of segment
XP = XC	Update XP
YP = YC	Update YP
JP#LOOP,T<12000	Repeat if necessary
DT 0; CD0	End contour
EN	End of program

ELECTROMATE

DMC-1200	DMC-1600	DMC-1700
DMC-1800	DMC-2000	

Application Example:
Controlling Contour Mode from a Host Program

This notes discusses the use of a host program to manage the execution of contour mode. The term host program refers to a program which is executed on the host computer and interacts with the controller.

Background

The contour buffer holds 1 element. To begin contour motion, issue the first contour point. To keep the controller from beginning the next contour point, issue the command WC ("wait for contour"), and then the next contour point. The second point will reside in the contour buffer until the first contour point is complete. The user can monitor the contour buffer by issuing the command CM ? or by checking the value of the operand, _CM. 1 means the buffer is full, 0 means the buffer is empty.

Example

This example demonstrates 2 axes of contour motion. Contour mode is begun by issuing the following:

CMXY	Contour mode for X and Y axes
DT 8	Time between points = 256 msec
CD 10000,10000	1 ST contour point
WC	Wait for next contour point
CD 10000,10000	2 ND contour point

The controller begins executing the first contour point and the second point has been placed in the contour buffer. The value returned by CM ? will be 1 until the first point is complete and the second point is being executed. If the command WC is issued after the second "CD" command, the controller will not respond until the first point is complete and there is room in the buffer. The host program will be able to send additional contour points when it receives a response from the controller. This is the simplest method for controlling the issuance of contour points. Another method is available which maintains open communications to the controller*. In this case, a monitor program that executes on the controller can be used to notify the host program when it is time to send the next contour point. Here is an example of such a program:

Monitor Program

#MONITOR	Monitor Program
ACK=0	Variable to set the state of
the host program	
JP#MONITOR, (_SCX<>50)&(_SCY<>50)	Loop until Contour Mode Executing
#LOOP	Loop while contour buffer
is 1 (full)	
JP#LOOP,_CM=1	
UI10	Issue a user interrupt
(10 was chosen at random)	
#WAIT	Wait for host program to set ACK variable
JP#WAIT,ACK=0	
JP#MONITOR,ACK=-1	Jump back to beginning of program
ACK=0	Set ACK back to 0
JP#MONITOR	
EN	

The monitor program waits for contour motion to begin and then monitors the contour buffer for availability. When room is available, the controller will generate an interrupt. The program on the host computer would need to respond to this interrupt by sending the commands:

```
WC
CD <value>,<value>
ACK=1 or ACK = -1
```

The variable ACK lets the monitor program know that the interrupt was accepted. When ACK is set to 1 the host program has additional contour points and when ACK is set to -1, the host program does not have additional contour points. When done, the monitor program will jump back to the main loop and wait for contour mode to start again.

---

*The controller will not be able to accept additional commands if a WC command is issued while the contour buffer is full. When the contour buffer becomes empty, the controller will respond.

Application Example: Creating a Spiral Motion Path

There are two methods for producing spiral motion with a Galil controller. The first method, demonstrated in the example program below, utilizes gearing and the CR command. Notice that the X and Y are "dummy" axes in the sense that they're only used to initiate the vector mode. The Z and W axes are incrementally geared off the X and Y commanded positions to produce the spiral motion. The gear ratios start at 1 and increment by 0.5 every 50 ms. The resulting profile is show in Figure 1.

```
#A
DP*=0
VMXY
CR500,0,1800
VE
GA,,CX,CY
GR,,1,1
N=1
BGS
#B
WT50
N=N+.5
GR,,N,N
JP#C, _CS=1
JP#B
#C
EN
```

ELECTROMATE

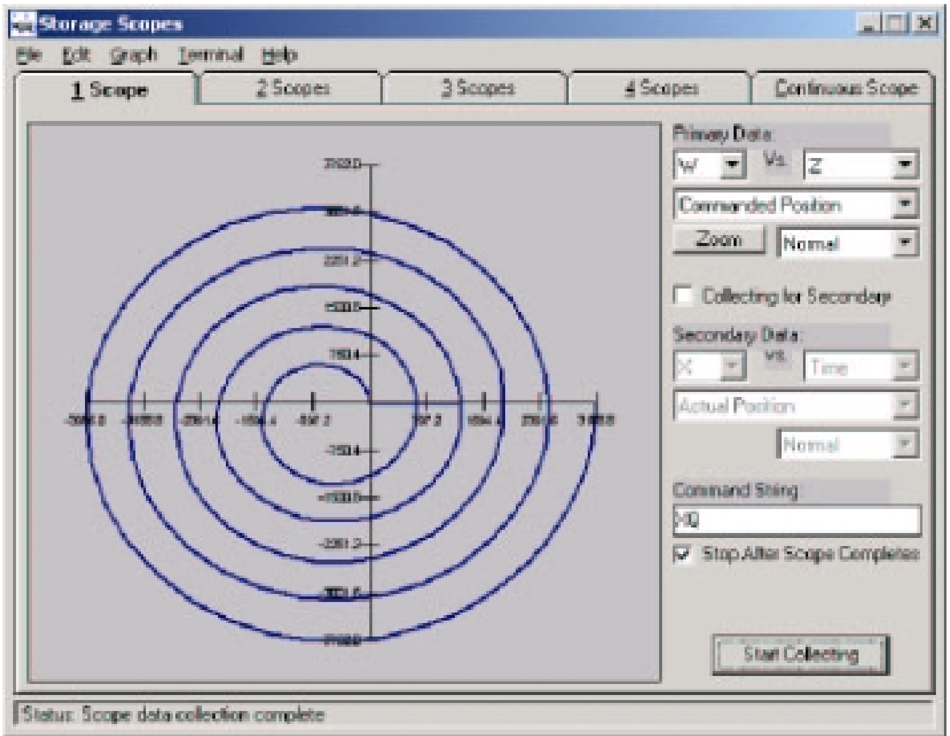


Figure1. View from WSDK of the first Spiral Method

Unfortunately, most users cannot spare “dummy” axes for the first method of spiral motion. The alternative method is to use very small VP segments to produce the profile. The example program below contains an iterative loop that calculates the coordinates for each vector segment. The radius of the spiral is incremented by 2 counts, and the X and Y positions are calculated using sine and cosine. Notice that the angles are incremented by 12 degrees. This is because smaller angle increments create vector segments that are too short for the controller to calculate. In order to allow for smaller angle increments, the starting radius must be larger.

Below is some example code for the second method of spiral motion. Also, Figure 2 shows the resulting profile in a WSDK screen shot.

```
#A
DP*=0
DM XVAL[1000],YVAL[1000],R1[1000]
R1[0]=200
YVAL[0]=0
XVAL[0]=200
CSS
CAS
VMXY
VP XVAL[0],YVAL[0]
M=1
```

```
BGS
#B
R1[M]=200+(2*M)
XVAL[M]=R1[M]*@COS[(12*M)]
YVAL[M]=R1[M]*@SIN[(12*M)]
VP XVAL[M],YVAL[M]
M=M+1
JP#B,M<1000
VE
EN
```

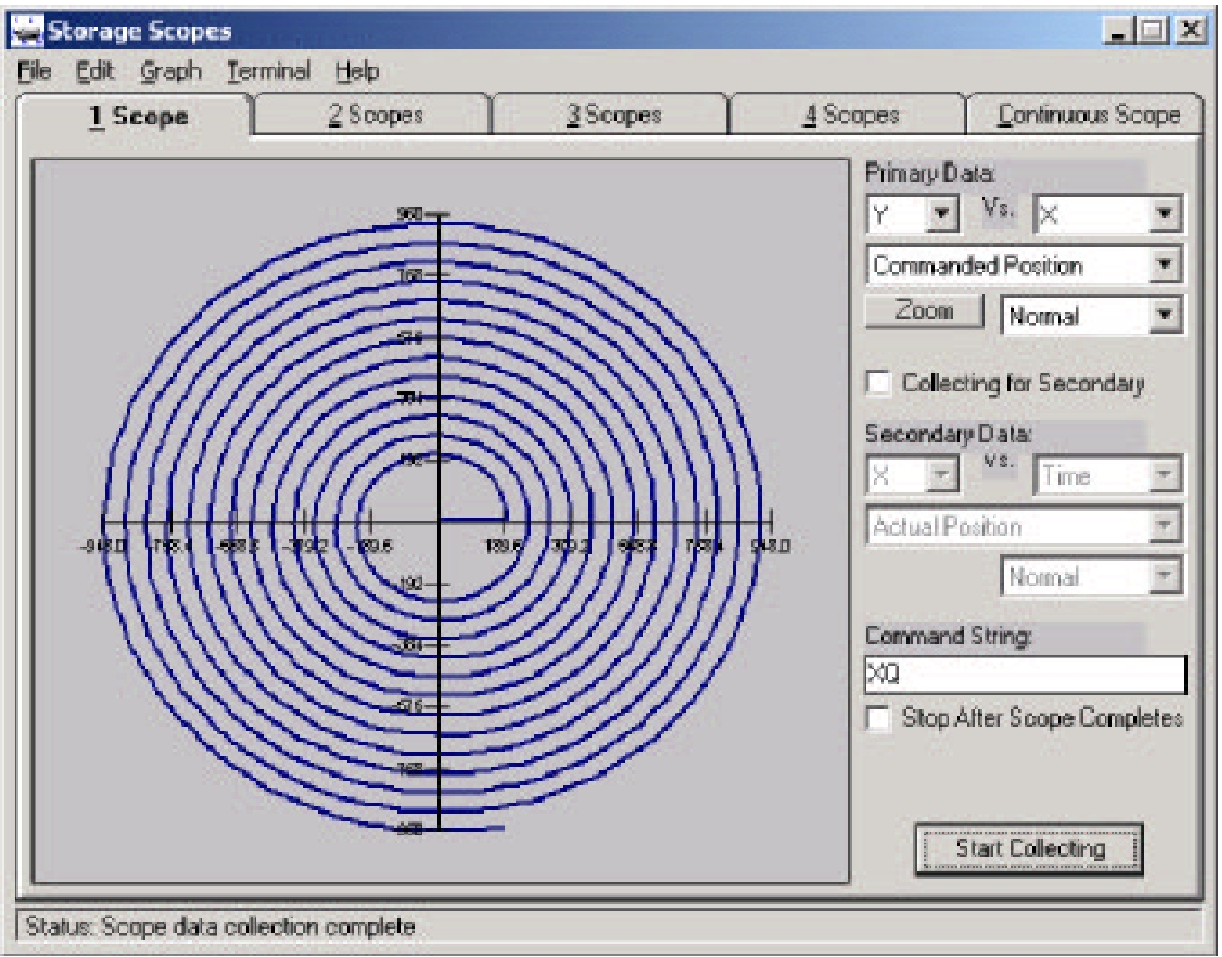
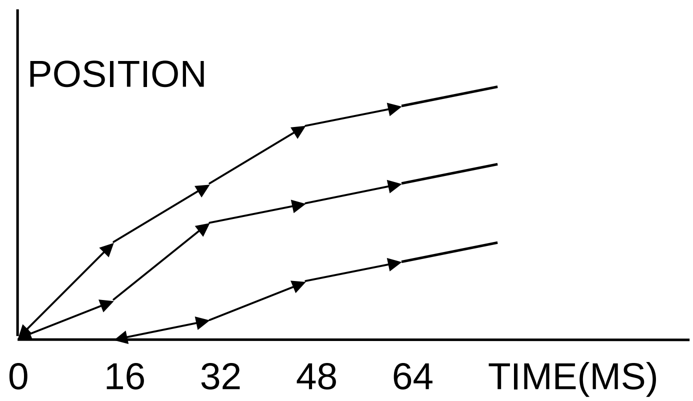


Figure 2. Screen Shot for the second method of Spiral Motion

Contouring

1. To generate arbitrary position trajectories
2. To create precise synchronization between axes



T	16	16	16	16
DX	16	12	8	4
DY	8	12	4	4
DZ	0	4	8	4

<u>Program</u>

```
CM XYZ
DT 4
CD 16,8,0
WC
CD 12,12,4
WC
CD 8,4,8
WC
CD 4,4,4
WC
•
•
•
DT 0
CD 0,0,0
EN
```

Application Example: 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]+V1V4=@INT[V3]	Compute position
POS[C]=V4	Integer value of V3
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

ELECTROMATE

Application Example: Elliptical Motion

Example

Generate motion along an ellipse

X = 4000 cos 360 T/1200

Y = 3000 sin 360 T/1200

period is 1200 ms

Method:

Divide the motion time into 16 ms intervals and use contouring

Instruction	Interpretation
# CONTOUR	Label
T = 0	Initial time
XP = 4000	Initial positions
YP = 0	
CMXY	Set mode
DT 4	T = 16 ms
#LOOP	
T = T+16	
A = T*0.3	Compute phase
X = @cos[A]*4000	Compute X
Y = @sin[A]*3000	Compute Y
IX = @INT[X]	Find integer part of X
IY = @INT[Y]	Find integer part of Y
DX = IX-XP	Compute increment X
DY = IY-YP	Compute increment Y
XP = IX	Update previous value
YP = IY	
CD DX, DY	Contour command
WC	Wait for contour data
JP #LOOP, T<1200	Repeat process
CD0,0	End contour
DT0	
EN	

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ELECTROMATE

Application Example: Elliptical Motion with Gearing

- Generate an ellipse motion using gearing.
- Note the equations

    Circle $X^2+Y^2=R^2$

    Ellipse $X^2+y^2/N^2=R^2$
- Generate a circle with XN axes.
- Define N as the master.
- Define Y as a slave with a programmable gear ratio.
- Example:

Generate an ellipse where X moves between +/-10000 and Y moves between +/-8000.

Program	**Interpretation**
#ELLIPSE	Label
VMXN	Vector mode
GAY=N	Set master
GR, 0.8	Define ratio
CR 10000,0,360	Command a circle
VE	Vector end
VS20000	Begin sequence
EN	End program

Application Example: Spiral Motion with Gearing

1. Generate a spiral motion of two turns.
2. Starting radius 10000 counts.
3. Radius increases 2000 counts per turn.
4. Vector speed must be constant.

Procedure:

1. Set vector mode with ZN. Command a circle with ZN.
2. Gear X to Z and Y to N, same ratio.
3. Start with gear ratio 1.
4. Increase the gear ratio linearly per rotation.
5. Update 200 times per turn or every 314 counts.
6. Reduce vector speed, VS, as reciprocal of gear ratio.

```
#SPIRAL              Label Program
VMZN                 Vector mode
GA Z,N               Gearing
GR1,1                Initial ratio
CR 10000,0,720       Define circle
VS 2000              Vector speed
VA 100000            Vector acceleration
VD 100000            Vector deceleration
VE                   Vector end
BGS                  Begin sequence
G=1.0                Initial gear ratio
#L                   Label subroutine
AV314                At every 1.8°
G=G+0.01             Increase gear ratio

GR G,G               Update gear ratio
V=2000/G             Calculate Speed
VSV                  Update speed
JP#L, G<1.4          Repeat 400 times or 2 turns
EN                   End program
```

DMC-1200	DMC-1600	DMC-1700
DMC-1800	DMC-2000	

Motion Programming: Gearing Resolution

Specifying Values for Gearing:

When using gearing, the gear ratio is specified as a decimal value in the range of +/-127.9999 with a minimum value of .0001. The actual gear ratio is represented in the binary number system. The fractional value has a minimum resolution of 1 part in 65536 (2 16). This means that the actual gear ratio will have a minimum resolution of 1.52587890625e-5. All fractional values of gearing will be converted to the nearest factor of 1 part in 65536.

To convert decimal values to actual value used by controller:

Step 1. Determine the decimal value to be used for gearing. Gearing values which are specified as decimal will be truncated to 4 places - for example .12345 will be truncated to .1234. Gearing values which are specified as ratios will be calculated to 5 places - for example, 1/3 will be calculated as .33333

Step 2. Multiply the decimal value by 65536.

Step 3. Round the result to the nearest integer value.

Step 4. The actual gear ratio will be the resultant value divided by 65536.

Example: User inputs gear ratio of .4

The actual gear ratio would be:

.4*65536 =26214.4

The rounded value of the result is 26214

The actual value would be 26214/65536 = 0.3999938964844

Special Features Mathematical and Functional Expressions

• Arithmetic, algebraic and trigonometric functions supported.

+	Addition
-	Subtraction
*	Multiplication
/	Division
&	And
|	Or
()	Parenthesis
@SIN	Sine
@COS	Absolute value
@FRAC	Fraction portion
@INT	Integer portion
@RND	Rounds number
@SQR	Square root
@IN	Digital input
@AN	Analog input
@COM[X]	1's compliment of X
@ASIN[X]	Arc sine of X
@ACOS[X]	Arc cosine of X
@ATAN[X]	Arc tangent of X
@OUT[X]	State of digital output X

ELECTROMATE

65 Interrogating Status Commands

TP	Tell Position
TE	Tell Error
TV	Tell Velocity (Average velocity)
TC	Tell error code (Reason for ?)
RP	Report Command Position (Useful for open loop step motors)
KP?	Returns value of parameter
V1=	Returns value of variable

ELECTROMATE

Program Flow: Conditional Statements

• Special commands that cause program to branch on specified condition.

• Format:

```
JP #DESTINATION,CONDITION
```

• Destination: Any valid label up to seven characters

• Conditions:

<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal

• Examples:

JP #STOP,ERROR>100	Jump to #STOP if ERROR is greater than 100
JS #B,V1+V2<0	Jump to subroutine #B if V1+V2 is less than zero

Program Flow: Loops

- Use Jump instruction, JP
- Use variable for loop counter
- Example:

Instruction	Interpretation
n=0	Initialize loop counter
#LOOP	Label
MG "LOOP COUNT=",n	Print
n=n+1	Increment counter
JP #LOOP,n<10	Loop 10 times
EN	End program

Program Flow: Trippoints

Motion control programs often include several functions. These functions are scheduled by delaying the execution of each function until a certain condition occurs. These conditions are called trippoints.

For example, the "after distance" (AD) trippoint delays the execution of a function until after the motor moves a certain distance. Similarly, the execution of a function may be delayed until a move has been completed (AM), an input signal changes states (AI), or by a pure time delay (WT).

- Special commands that halt program execution until a specified event occurs.

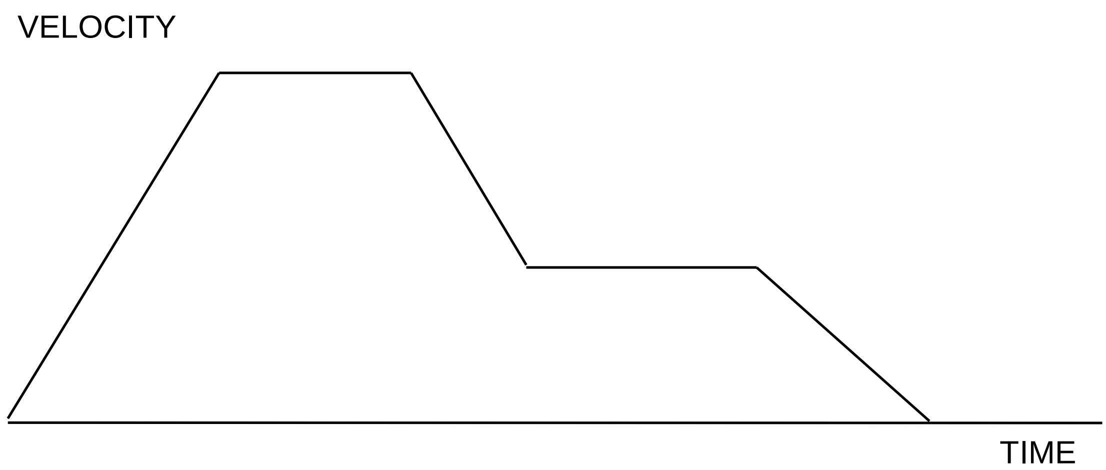
AD	After distance
AR	After relative distance
AP	After absolute position
AI	After input
AM	After motion complete
WT	After time elapsed (wait)
AT	After time relative to previous AT
AS	After speed
AV	After vector distances
WC	Wait for contour data

- Example:

Instruction	**Interpretation**
#A	Label
PR 1000	Position 1000
BGX	Begin
AM X	Wait for motion done
PR -1000	Position -1000
BGX	Begin
EN	End Program

Application Example: Trippoints

For complex motion profile:



Execution time is specified by trippoints

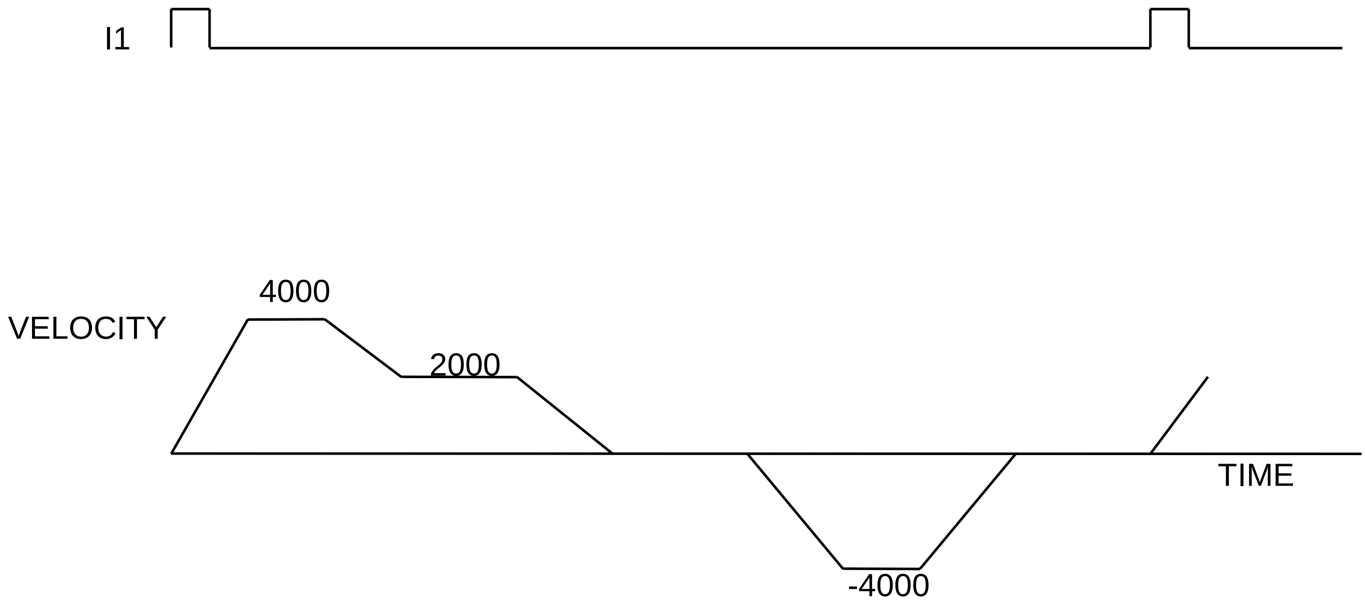
AD n	After distance
AP n	At position
AMX	After move of X
WT 500	Wait 500 ms
AI 3	After Input 3

Combine trippoints with:

JP - Jump on Condition
JP #A,_TEY = 0
JP #B,_TPX>100
JP #C,_TPX-_TPZ<2

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Example:



Requirements
After start pulse, move forward a distance of 4000 counts at
an initial speed of 4000 ct/s. After a distance of 2000, lower the speed to 2000. After the completion of the forward motion, wait for 100 ms and move backwards at a speed of 4000 ct/s.

Trippoints
AI1
AD 2000
AMX
WT 100

Instruction	**Interpretation**
#A	Label
PR 4000	Distance
SP 4000	Speed
AC 10000	Acceleration
DC 10000	Deceleration
AI1	Wait for Input 1
BGX	Start X motion
AD 2000	Wait until distance = 2000
SP 2000	New Speed
AMX	Wait until move is complete
WT 100	Wait 100 ms

SP 4000	New speed
PR -4000	Distance
BGX	Start motion
AMX	Wait for motion completion
JP #A	Start the cycle again
EN	End

Motion Programming: I/O Interface

It is often desirable to synchronize motion with input/output (I/O) events, which enables the controller to control a complete process. As such, the controller can read input signals, both digital and analog, and can generate digital or analog output signals. Inputs may be control signals from digital push-buttons or analog potentiometers. Outputs can be used to activate solenoids or valves or to turn on indicator lights.

Input signals can be read by the controller and their values can be stored in control variables. These may be used later in motion programs.

The reading of digital inputs may be performed with the instruction:

```
DIGITAL = @ IN[2]
```

which reads the digital input #2 and stores its content, 0 or 1, in the variable DIGITAL. Analog signals are similarly read with the instruction:

```
ANALOG = @ AN[3]
```

which reads analog input #3.

Digital output signals are generated by setting or clearing a bit with the instructions:

```
SB 3
CB 3
```

which sets and clears output bit #3.

The interface with the inputs and the generation of the output signals allows the controller to perform complete process control without a host computer intervention. This is illustrated by the following example.

Consider the simple case where the motion of X must be delayed until the start pulse is given (applied to input 1). When the motion is completed, an output signal (output 1) must be given for one second.

Instruction	Interpretation
PR 7000	Distance
SP 5000	Speed
AI 1	Wait for start signal
BGX	Start motion
AMX	Wait for completion
SB 1	Set output 1 high
WT 1000	Wait 1 second
CB 1	Clear output 1
EN	End of program

Input/Output Interface

Digital Inputs

Sensors, Switches
Start/Stop Signals

To Read Inputs

V1 = @IN [1]	Define variable
AI2	Wait for input #2 to go high
JP#B,@IN[3]=1	Conditional jump
II4	Input interrupt

Outputs

To activate devices
Status reporting

To Change Outputs

SB1	Set Output Bit 1
CB2	Clear Output Bit 2
OB3,I1&I2	Programmable output

Analog Inputs can be Unipolar/Bipolar ±10v Type 12 Bit resolution (16 Bit optional)

To read tension, force, temperature, potentiometers, etc.

TENSION=@AN[2]

I/O Uncommitted Inputs

• 8 or more isolated digital inputs (n=1 through 8), expandable with expansion I/O modules

• +5 Volt to +24 Volt inputs permissible

• Available commands:

AI n	Wait for input n high
AI -n	Wait for input n low
JP #A,@IN[n]=1	Jump if input n high
II n	Interrupt if input n occurs

• After Input Example:

#JOG	Jog program
JG1000	Jog speed
AI 1	After input 1 high
BG X	Begin motion
AI -1	After Input 1 low
ST X	Stop motion
MG@IN[1]	Display state of digital input #1

NOTE:

AI halts program sequencer until designated input condition occurs.

Motion Programming: Input Interrupt Functions/Subroutines

The controller 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 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.

ELECTROMATE

Application 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	Loop until Interrupt cleared
JP #LOOP,@IN[1]=0	Conditional jump
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG XY	Begin motion on X and Y axes
RI 1	Return from Interrupt subroutine

Application Example: Starting Motion From A Switch Input

Motor X must turn at 4000 count/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 controller. 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;	

Motion Programming: Using IF, ELSE and ENDIF Constructs

Instruction	Interpretation
#TEST	Begin Main Program "TEST"
II,,3	Enable input interrupts on input 1 and input 2
MG "WAITING FOR INPUT 1, INPUT 2"	Output message
#LOOP	Label to be used for endless loop
JP#LOOP	Endless loop
EN	End of main program
#ININT	Input Interrupt Subroutine
IF (@IN[1]=0)	IF conditional statement based on input 1
IF (@IN[2]=0)	2nd IF conditional statement executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message to be executed if 2nd If conditional is true
ELSE	ELSE command for 2nd IF conditional statement
MG"ONLY INPUT 2 IS ACTIVE"	Message to be executed if 2nd IF conditional statement
ENDIF	End of 1st conditional statement
#WAIT	Label to be used for a loop
JP#WAIT,(@IN[1]=0) | (@IN[2]=0)	Loop until both input 1 and input 2 are not active
RI0	End input Interrupt Routine without restoring trippoints

I/O Uncommitted Outputs

• 8 or more digital outputs (n=1 through 8), expandable with expansion I/O modules

• Up to +28 Volt 500 mA isolated outputs available

• Defined by:

SB n	Set Bit n
CB n	Clear Bit n
OP n	Define all 8 bits
OB n, expression	Make output n equal to value of expression (0 or1)

• Example:

Instruction	Interpretation
#POS	Label
PR 1000	Position
BG X	Begin motion
AM X	After motion
SB1	Set Bit 1
OB 2,I1&I3	Output 2 is equal to Input 1 and Input 3
EN	End program

ELECTROMATE

I/O Analog Inputs

- 7 Analog inputs (n=1 through 7), expandable with I/O expansion modules
- ±10 Volt resolved over 12 bits (16 bit resolution optional)
- Available commands:

V1=@AN[n]	Read analog input (in volts)

- Analog Input Example:

Instruction	**Interpretation**
#JOYSTK	Joystick
JG0	Jog mode
#LOOP	Loop
V1=@AN[1]	Read analog input
JG V1*1000	Change jog speed
JP #LOOP	Repeat
EN	End program

I/O Input of Data

- IN command used to prompt operator to enter data
- Example:

IN “Enter Length”,L1 Entered value assigned to L1.	Sends prompt.
IN “Enter Name”,N{s} Entered characters assigned to string variable,N.	Sends prompt.

- Input Prompt Example:

Instruction	**Interpretation**
#ROTATE	Program label
IN “ENTER # OF REVS”,R	Prompt for revs
IN “ENTER SPEED (RPM)”,S	Prompt for speed
PR R*4000	Position command
SP S*4000/60	Speed command
BG	Begin motion
EN	End program

I/O Output of Data

- MG command used to send a message
- Examples:

MG “HELLO THERE” Displays message

MG “POSITION =”,_TPX Displays Message

and prints actual position

MG “Variable =”,V1{F6.1} Displays message

and value of variable V1 with format 6 integers and one decimal

Control Variables

Many motion applications include variable parameters. For example, a cut-to-length application often requires that the cut length be variable. The motion process is the same, but the length is changing. To accommodate these applications, advanced controllers provide symbolic variables. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. Variables allow the motion controller to perform certain mathematical functions and to make decisions accordingly. This capability increases the performance of the motion controller and allows it to perform some (background PLC) supervisory functions in addition to the simple motion control.

A variable can be defined as a constant or can be equated to a controller parameter. For example, the instruction:

    V = 3

sets the value of the variable V to 3. The instruction:

    P = _TPX

reads the position of the X motor and equates the variable P to that value. Variables can be defined in a variety of ways. For example, the instruction:

    YERROR = _TEY

equates the variable YERROR to the position error of the Y motor.

Once the variable is defined, it can be used in mathematical operations. The controller can perform mathematical functions which typically include algebraic, trigonometric, and logical operations.

When the operation is completed, the controller can use the computation result to adjust the system parameters. For example, the results can be used to change the speed, set the distance, or change the filter gain. The use of variables is best illustrated by the following example.

The following program shows an example where the X motor follows the position of the Y motor. This is done by driving the X motor at a speed proportional to the position difference.

Instruction	**Interpretation**
#FOLLOW	Program name
DP 0,0	Define X,Y position as zero
JG 0	Set initial X speed to zero
BGX	Start X
#LOOP	Label
VE=_TPY-_TPX	Find the position difference
VEL=VE*10	Compute the speed
JG VEL	Modify the speed
JP #LOOP	Repeat the process
EN	End of program

Control variables allow motion controllers to perform mathematical functions and change the motion parameters according to the computation results.

Examples:

V1 = 3.7	Constant
VP =_TPX	Position of X
VE =_TEY	Position error of Y
INPUT=@IN[5]	Digital Input 5
ANALOG=@AN[2]	Analog Input 2

Manipulation

```
VA = VB*3+VC
RESULT = @SIN[V6]
```

Assignment

```
PR V4
JG VEL
```

Variables

Allows writing a motion program in generic form and specifying parameters in a given case.

PROGRAM

```
#MOVE
PR DIST
BGX
EN
```

To run:

```
DIST = 3000
XQ# MOVE
```

Allows mathematical functions and correction

```
ERROR = 1000 - _DEX
PR ERROR * 2
BGX
```

Variables can also serve as “mailboxes” for communication between a host computer and the controller

PROGRAM	HOST
#A	
JG 100	
BGX	←V = 3000
#LOOP	
JGV	←E =
E=@AN[1]	→23
JP#LOOP	

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Motion Programming: Control Variables and Offset

Objective: Illustrate the use of variables in iterative loops and use of multiple instructions on one line.

Instruction	**Interpretation**
#A	Set initial values
DP0	Define home position
V1=8; V2=0	Initializing variables to be used by program
#B	Program label #B
OF V1	Set offset value
WT 200	Wait 200 msec
V2=_TP	Set variable V2 to the current position
JP#C,@ABS[V2]<2	Exit if error small
MG V2	Report value of V2
V1=V1-1	Decrease Offset
JP #B	Return to top of program
#C;EN	End

This program starts with a large offset and gradually decreases its value, resulting in decreasing error.

Special Features Symbolic Variables

- For inputting or changing parameters in a program
- Up to 254 user-defined variables
- Each variable defined by a name, up to eight characters
- Range: ±2,147,483,647.9999 for numeric variables
  Six characters for string variables
- Examples:

POSX=500.98	Assigns value 500.98 to the variable, POSX
PR POSX*2	Multiply variable, POSX, by 2 and assign to X axis position command
VAR="CAT"	Assign the string, CAT, to the variable named VAR

Program Flow: Automatic Subroutines

- Most controllers can monitor certain conditions in the background (i.e. secondary PLC functions)

- Automatic monitoring is enabled by using the following special labels:

#LIMSWI	Limit switch active
#ININT	Designated input goes low
#POSERR	Position error exceeds ER limit
#CMDERR	Bad command given

- An applications program must be running for automatic monitoring.

ELECTROMATE

Motion Programming: Special Labels

Galil controllers have 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 turn point
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for communication interrupt subroutine

DMC-1200	DMC-1600	DMC-1700
DMC-1800	DMC-2000	

Motion Programming: Displaying Binary Numbers

The following program displays an 8 bit number in binary format. In this example, the first 8 general inputs are displayed as binary.

```
#TELL
M=0
N=_TI0
O=0
P=0
Q=128
#LOOP
O=(N&Q)
P=(10*P)+@INT[(O/Q)]
Q=Q/2
JP#LOOP,Q>=1
MGP {F8.0}
JP#TELL
EN
```

Motion Programming: Limit Switch Routines

Galil controllers provide 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.

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Application Example: Bouncing off of a limit switch

When using a pulse type switch with the HM command, the motor will always begin motion in the same direction, irrespective of the position of the motor. One method for using a pulse type or momentary home switch is to implement a routine to "bounce off" of the limit switches. In this method, the motor is commanded to HOME. If the motor is on the 'wrong' side of the home switch, the motor will eventually hit the limit switch. If this happens, the controller will move the motor to the other side of the home switch and re-issue the HOME command.

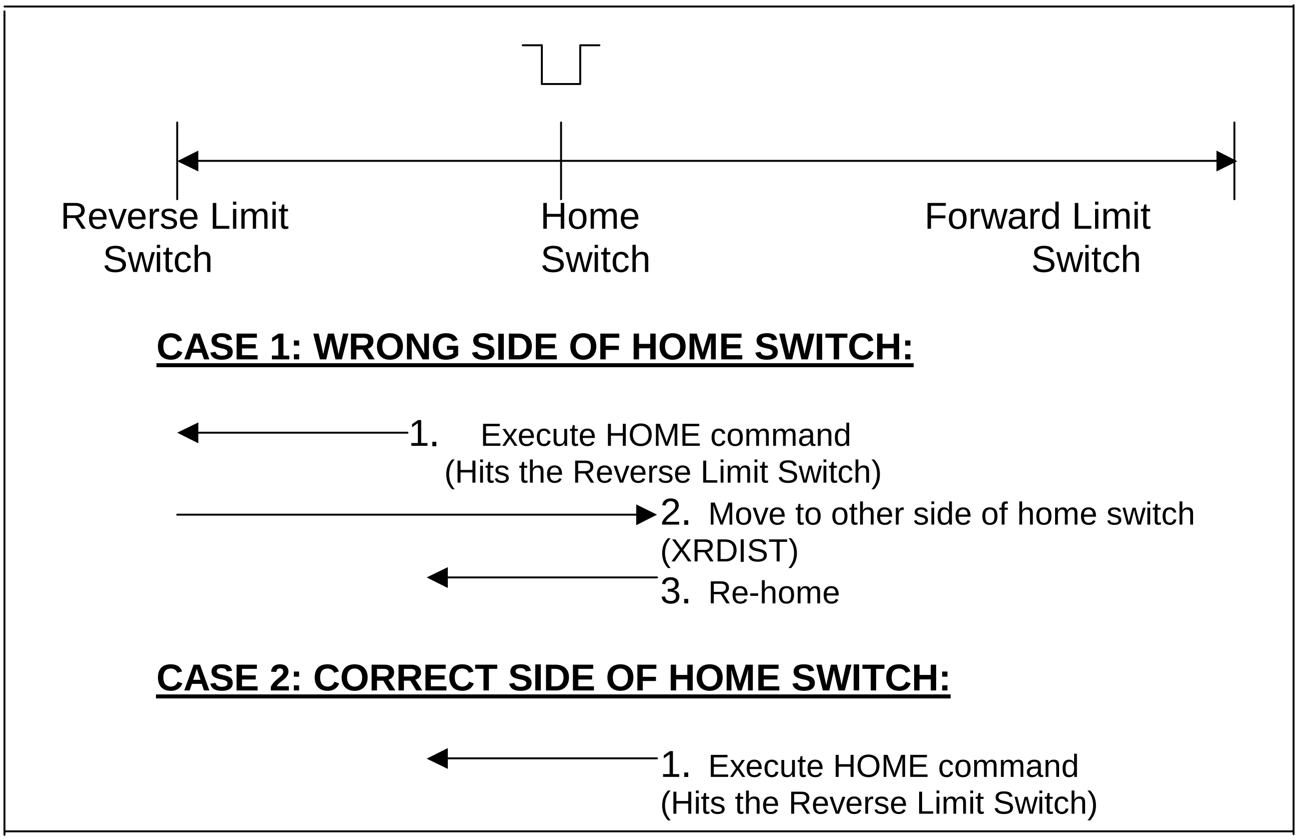


Figure 1. This figures shows the method of bouncing off of the limit switch.

The Program:

```
#BOUNCE
HSTATE = 0                          ;Variable to identify
                                    status of homing
XFDIST=<value to be inserted>       ;See Note below.
XRDIST=<value to be inserted>       ;See Note below.
#HOMEX                              ;Home X routine
HMX;BGX;HSTATE=1                    ;Begin motion, set
                                    status variable
```

```
AMX;HSTATE=0                      ;After motion complete,
                                  clear status variable
EN                                ;End of homing routine
#LIMSWI                           ;Limit switch subroutine
JP#FWDX,(_LFX=0)&(HSTATE=1)       ;Bounce off forward
                                  limit switch if homing
JP#REVX,(_LRX=0)&(HSTATE=1)       ;Bounce off reverse
                                  limit switch if homing
MG "HIT LIMIT SWITCH"             ;Otherwise, return a
                                  message
AB0                               ; and abort motion
#FWDX                             ;Forward Bounce
                                  Routine
AMX                               ;Wait for motion to stop
                                  after hitting limit
PRX=XFDIST                        ;Move off of limit to other
                                  side of home
BGX                               ;Begin motion
AMX                               ;Wait for motion to
                                  complete
HMX                               ;Re-home X axis

BGX                               ;Begin motion on X
                                  axis
RE1                               ;Return from routine
                                  & re-enable trippoint
#REVX                             ;Reverse Bounce
                                  Routine
AMX                               ;Wait for motion to
                                  stop after hitting
                                  limit
PRX=XRDIST                        ;Move off of limit to
                                  other side of home
BGX                               ;Begin motion
AMX                               ;Wait for motion to complete
HMX                               ;Re-home X axis
BGX                               ;Begin motion on X axis
RE1                               ;Return from routine
                                  & re-enable trippoint
```

Notes:

This program includes two routines for bouncing off of both limit switches. Once the system is configured the homing routine will only require one 'bounce' routine (#FWDX or #REVX). This program can be reduced once the required routine has been identified. The values for XFDIST and XRDIST must be opposite polarity

For example, XFDIST might be - 3000 CTS, and XRDIST might be 10000 CTS.

Application Example: Making Jumps Out of Automatic Subroutines

The automatic subroutines, LIMSWI, POSERR and ININT must be properly terminated to be re-enabled. LIMSWI and POSERR must be ended with the command RE or a ZA command must be given. The subroutine #ININT must be ended with the command RI. RE and RI are used to end the subroutine just as EN is used to end other subroutines.

To make and unconditional jump from #ININT, there are two methods for re-enabling the interrupt capability; 1)Re-issue the command II. 2) Use a ‘null’ routine. This routine allows for the execution of the RI command before the unconditional jump.

#ININT Example:

#TEST	Test routine
II1	Set Input Interrupt on input 2
#LOOP1	Simple loop function
MG”WAITING FOR INTERRUPT”	Message
WT1000	Wait 1000msec
JP#LOOP1	Jump to subroutine
EN	End program
#DONE	Routine to execute when interrupt cleared
MG”DONE WITH INTERRUPT”	Message
JP#LOOP1	Jump to subroutine
EN	End
#ININT	Input Interrupt routine
MG”INTERRUPT ON INPUT”	Display message
#WAIT	Wait for input to be cleared
JP#WAIT,@IN[1]=0	Conditional jump statement
JS#RESETI	Call’null’ subroutine
ZS	Zero stack
JP#DONE	Jump to #DONE routine
EN	End of ININT subroutine
#RESETI	Null routine
RI1	ININT ending command (re-enables #ININT)

Application Example: Stop at a Mechanical Limit

- Drive the motor at initial speed of 10000 ct/s, with acceleration and deceleration set at 100000 ct/s^2
- Expect a mechanical stop after 20,000 counts.
- Slow down to a speed of 2000 ct/s before the mechanical stop.
- Start deceleration at 20000 –480=19,520.
- Detect the mechanical stop by observing the following error.
- When stop is detected apply a constant force corresponding to 2 volts motor command. (Amplifier is in current mode).
- Stop the motion of motor.

Program	Interpretation
#STOP	Label
AC 100000	Acceleration
DC 100000	Deceleration
JG 10000	Speed
BGX	Start motion
AD 19520	Wait for point
JG 2000	Deceleration
#L	
JP#L,_TEX<100	Check for error
TL 2	Limit output
STX	Stop Motion
EN	End

Application Example: Pause Motion

Task: When input is triggered, decelerate vector motion to a stop, when input is de-activated, resume motion

- Create #ININT subroutine in Program
- Use General Use Input to generate interrupt
- Set Vector Speed, VS, to zero
- Pause while interrupt remains active

Instruction	**Interpretation**
II1	Set X axis only for vector motion
#LOOP	Label program
VP –4000,0	Define vector move
CR 1500,270,-180	Define arc segment
VP 0,3000	Define vector move
CR 1500,90,-180	Define arc segment
VE	Vector end
BGS	Begin sequence
AMS	After move sequence
JP#LOOP	Repeat motion when done
EN	End program
#ININT	Input Interrupt routine
N=_VS	Save Current Vector Speed
VS 0	Set Speed to 0
#PAUSE	Wait while input is active
JP#PAUSE, @IN[1]=0	Conditional jump
VSN	Resume
RI1	Retune from interrupt

ELECTROMATE

Application Example: Motion Complete Timeout

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.

Instruction	Interpretation
#BEGIN	Begin main program
TW 1000	Set the time out to 1000ms
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 without restoring trippoint

Application Example: Correcting Wrong Operator Input Data

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
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
EN1	End program and restore trippoint

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.

	DMC-2000	

Application Example: Using the Term-H/P Pendant

Here is an example that uses the hand held terminal to monitor when a key is being held down. In this example, the function keys F1 - F4 are being used to increase and decrease the X and Y jog speed. As the key is held down, the jog speed (s) is continuously changed until the key is released. The F5 key is used to stop the motion on both the X and Y axes. The terminal has to be set up for fast repeat function which causes the terminal to send out characters every 50msec. When holding down a key, the first character is not repeated until after approximately 2 seconds, so there is a special case for when the key is first held down. When the key is held down, the communication interrupt jumps to the Increase, Decrease, or STOP routine. In the each routine, the controller clears the communication buffer (P2CH), sets a new speed if F1, F2, F3 or F4 was depressed and waits for 50msec. If the communication interrupt does not take control within the 50msec, then the key must have been released and the next command causes the system to stop (ST command).

The terminal was configured with the Key Click Disabled to avoid the sound generated when holding the key down.

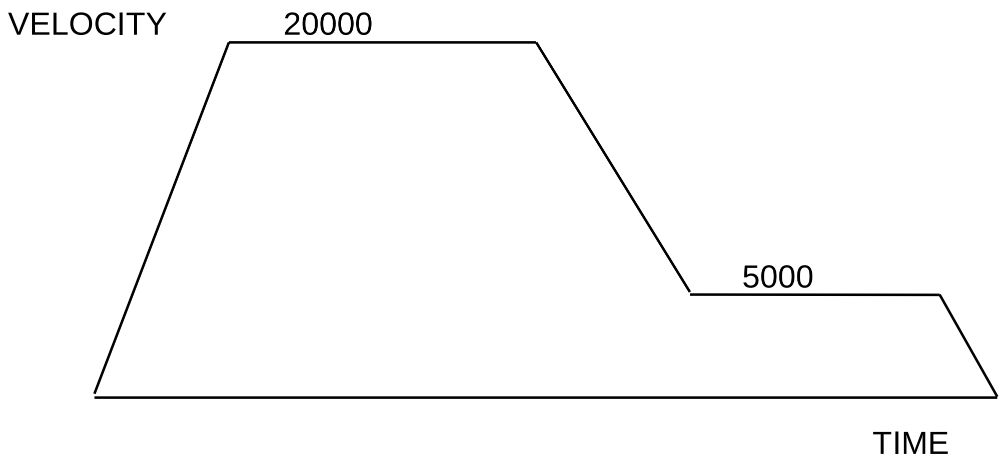
```
#F4TEST
FIRST=1
CC 9600,0,0,0
MG {P2},{^27},"V"
SPEEDX=10000
SPEEDY=10000
JG SPEEDX,SPEEDY
BGXY
CI 2
#LOOP
WT 1000
MG "PRESS F4 TO HOLD, THEN STOP"
JP#LOOP
EN
#COMINT
ZS
JP#STOP1,(P2CH=F4) &
(FIRST=1)
JP#STOP,(P2CH=F4)
```

```
JP#LOOP
EN
#STOP1
FIRST=0
CI -1
CI 2
WT2000
ST
EN
#STOP
CI -1
CI 2
WT100
ST
EN
```

Application Example: Point-to-Point Move

- Requirements: Drive a motor toward a mechanical stop and apply a constant force. The distance to the stop is between 8000 and 8500 counts.

- Apply a force of 20 lbs. This equals, for example, 1.6 Amp and 0.8 Volts of amplifier command.



Instruction	**Interpretation**
#PUSH	Label
PR 9000	Distance
SP 20000	Speed
AC 100000	Acceleration
DC 100000	Deceleration
BGX	Start motion
AD 6000	Wait until distance = 6000
SP 5000	Lower the speed
AD 8000	Ready for contact
#LOOP	Define subroutine
JP #LOOP,_TEX<100	Wait for contact
TL 0.8	Limit the output
EN	End program

Application Example: Pick and Place

Pick and Place
An automated IC insertion machine is used to pick up a part at an X,Y location and move it to the proper location on the circuit board. A PC-based controller needs to move an XY table along a straight line to the specified locations. The pick-up head is controlled by the Z axis which raises the head during movement and lowers the head during placement.

Requirements
System resolution: 0.1 micron
Accuracy: 1micron
Speed: 40,000 counts/sec
PC-based

Operation
The motion requirement is to pick up a part at coordinate X1, Y1 and to place it at coordinate X2, Y2. Once the coordinates are specified, the controller drives the XY table on a straight line to the pick-up location. Once there, the pick-up head, which is controlled by the Z axis, is lowered and the holding solenoid is activated. Next, the pick-up head is raised, the table is driven to the new location, and the pick-up head is lowered. Finally, the solenoid is released and the pick-up head is raised again.

The motion program includes two parts. The first, #INITIAL, is performed once to initialize the system. Consecutive moves are executed with the program #PICK.
Specifically, the controller computes the differences, DX, DY, between the starting position, X0, Y0, and the pick-up position, X1, Y1. It then commands the XY axes to move on a straight line with the VP DX, XY instruction. Upon completion, the Z axis is lowered and then the output bit 1, which activates the solenoid, is energized. The process is repeated to move the motor to the new coordinate. The instructions are given in the following program.

Instruction	Interpretation
#INITIAL	Label
HMXY	Drive X and Y to home
BGXY	Start motion
AMXY	Wait until completion
X0=0	Define starting position as zero
Y0=0	Define starting position as zero
#PICK	Label
DX=X1-X0	Find X difference
DY=Y1-Y0	Find Y difference
VP DX,DY	Command motion

VS 40000	Vector speed
VA 200000	Vector acceleration
VD 200000	Vector deceleration
VE	End of move
BGS	Start XY motion
AMS	Wait for motion completion
PR,,-50000	Move head down (Z-axis)
SP,,20000	Z speed
AC,,80000	Z acceleration
DC,,80000	Z deceleration
BGZ	Start Z motion
AMZ	Wait for Z motion completion
SB 1	Set output bit -- solenoid
WT 20	Wait 20 ms
PR,,50000	Raise head
BGZ	Start Z motion
DX=X2-X1	Compute the X difference
DY=Y2-Y1	Compute the Y difference
VP DX,DY	Motion command
VE	End of move
AMZ	Wait for Z completion
BGS	Start XY motion
AMS	Wait for XY completion
PR,,-50000	Lower head
BGZ	Start head motion
AMZ	Wait for Z motion completion
CB 1	Clear output bit -- release solenoid
WT 20	Wait 20 ms
PR,,50000	Raise head
BGZ	Start Z motion
X0=X2	Update starting X position
Y0=Y2	Update starting Y position
EN	End program

Application Example: Press Fitting Machine

A linear slide moves a machined component under an automated tool that inserts a bearing into the part. The objective is to ensure the insertion force falls within a specified range. The actual force required will be recorded for each insertion with an analog strain gage. If the force required falls outside the range, the assembly must be rejected. A PC will be used as the operator station using Visual Basic for the operator interface.

Requirements
Force range: 1.00-25.50 lbs (programmable limits)
Accuracy: ±0.0002 inch
Resolution of motion: 40000 counts per inch
(zero backlash ball nut on ball screw)

Solution
An advanced level controller is used to control the motion of the system. One axis is used to position the linear slide under the tool holding the bearing while the second axis controls the vertical position of the insertion press. Force measurements are made from a strain gage that outputs an analog voltage. This voltage will be read by one of the analog inputs of the controller and used to monitor the insertion pressure. If the force exceeds the upper limit at any time during the insertion, the motion will be aborted and the parts rejected. Likewise, if the insertion depth is
reached and the force remains below the minimum value the parts must be rejected.

Visual Basic is used to produce a user interface that displays the motion of the system and system status, as well as position and insertion pressure values. This program communicates to the controller through the VBX custom control produced by Galil Motion Control. A description of portions of the Visual Basic code follows:

When a form is first loaded, the code within the FORM_LOAD procedure is automatically executed. In this case communication is established and then the commands to be sent to the card during each polling interval are defined. Finally, the polling interval is set and the polling is enabled.

```
Sub Form_Load ()
‘sets up the communication
dmcshell1.dmcaddress = “1000”
dmcshell1.dmcconnect = True
‘allows polling of the card for the position and
 pressure values
dmcshell1.dmcinterrogate(0) = “TPX”
dmcshell1.dmcinterrogate(1) = “TPY”
dmcshell1.dmcinterrogate(2) = “MG @AN[1]”
‘sets the time between polls and begins polling
dmcshell1.dmcpollinterval = 200
```

```
dmcshell1.pollcontroller = True
End sub
```

Three status variables are polled every 200 milliseconds: X position, Y position, and the value of analog input #1. Each response is placed into the correct panel by the following code.

```
Sub DMCShell1_DMCInterrogate (index As Integer,
response As String)
 'routine is run each time a poll is completed
 If index = 0 Then
 press_pos_panel.caption = Val(response)
 End if
 If index = 1 Then
 slide_pos_panel.caption = Val(response)
 End if
 If index = 2 Then
 pressure_panel.caption = Val(response)
 End if
 End sub
```

If the button labeled “START” is pressed commands must be sent to download a DMC program and then execute it within the control card:

```
 Sub start_button_Click ()
 'routine downloads the DMC file “press.dmc” to controller
 then executes it
 dmcshell1.filename = “press.dmc”
 dmcshell1.dmcfileoperation = 1
 dmcshell1.dmccommand = “XQ”
 End sub
```

Cut to Length

A plastic strip is pulled from a feedroll and must be cut at a specified length. Also, the number of pieces must be programmable. The operation is done by advancing the web to a specified distance and activating the cutter.

The operation must be stand-alone. The operator has a hand-held terminal for selection the length and number of cuts.

Requirements

Range of cut: 6" to 36"
System resolution: 0.0002" or 5000 counts per inch
Slew speed: 12 inches/sec
Acceleration/deceleration: 200 in/s^2
Cutting time interval: 200 ms
Required accuracy: ±0.001"
Stand-alone

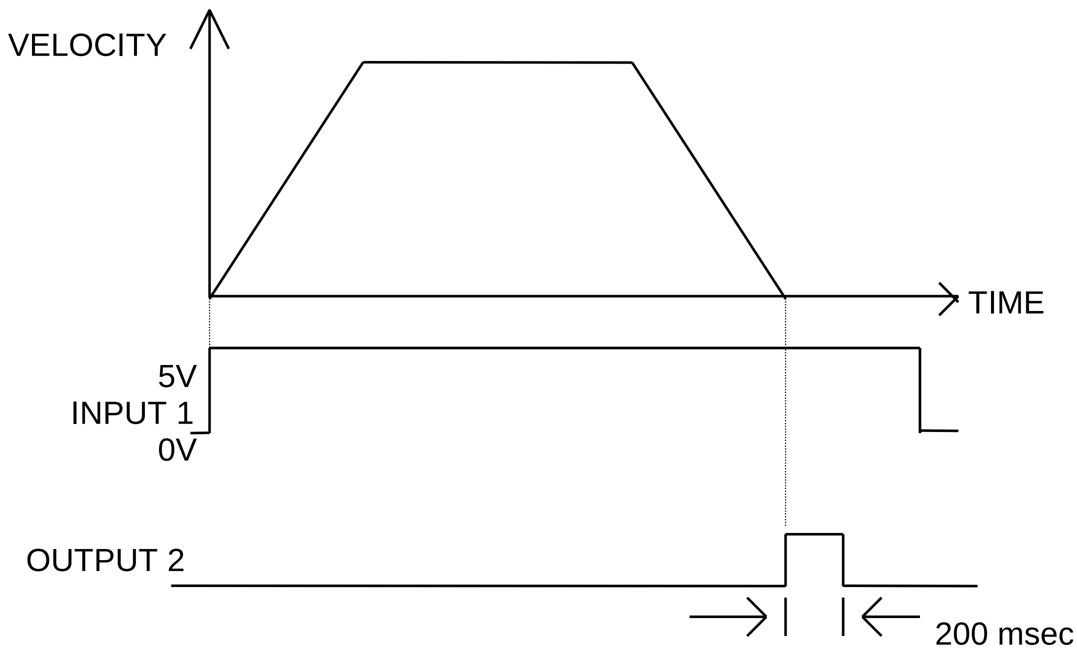
Operation

An advanced level stand-alone controller prompts the operator to enter the length in inches, L, and the number of cuts, N.

To start the operation, a switch connected to input 1 is turned on. The cutting cycle includes the motion interval followed by the cutting interval. The actual cutting tool is controlled by toggling output 2 on the controller.

The operation ends when the number of cuts is completed or when input 1 is turned off, whichever occurs first.

As the operation is stand-alone with a single axis of motion, the DMC-2010 controller and Term-H-P2 hand-held pendant were selected. The control program below is downloaded to the controller via the RS232 or RS422 port and stored in non-volatile memory permitting stand-alone operation.



Motion Profile for Cut to Length Application:

Instruction	**Interpretation**
#AUTO	Label for autostart
CC9600,0,0,1	Configure for Term-H input
DPO,0	Define home position
KI1;KD64;KP10	Set tuning parameters
IN {P2} "ENTER CUT LENGTH IN INCHES",L	Prompt for operator
IN {P2} "ENTER NUMBER OF CUTS",N	Prompt for operator
#WAIT	Label for wait
JP #WAIT, @IN[1]=0	Wait until input 1 is high
C=0	Initialize cut counter
#LOOP	Label for loop
PR L*5000	Convert inches to counts
SP 60000	Speed in counts/sec
AC 1000000	Acceleration in counts/sec^2
BGX	Begin motion
AMX	Wait for motion complete
SB2	Activate cutter, set output 2 high
WT 200	Wait 200 msec
CB2	Deactivate cutter, clear output 2
C=C+1	Increment cut counter
JP #E,C=N	Exit if done

```
JP #LOOP,@IN[1]=1        Repeat if input 1 is still high
#E                       Exit
EN                       End program
```

Application Example: Position Follower I

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

Approach 1 - Point-to-Point

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

Instruction	**Interpretation**
#POINTS	Label
SP 7000	Speed
AC 80000	Acceleration
DC 80000	Deceleration
#LOOP	Label branch program
VP=@AN[1]*1000	Read analog input, compute position
PA VP	Command position
BGX	Start motion
AMX	After completion
JP #LOOP	Repeat
EN	End

ELECTROMATE

Application Example: Position Follower II

Objective: The motor X follows the motor Y at a ratio of one-to-one.

Approach

Method: Force the follower to run at a speed that is proportional to the following error.

Instruction	Interpretation
#FOLLOW	Label
DP_TPY	Set initial positions
AC 100000	Acceleration
DC 100000	Deceleration
JGO	Set in jog mode
BGX	Start motion
#LOOP	
E=_TPY-_TPX	Following error
JG E*20	Update speed
JP #LOOP	Repeat
EN	End

Application Example: Modified Position Follower III

Method: Modify the program to eliminate steady state following errors.

Instruction	**Interpretation**
#FOLLOW2	Label
DP_TPY	Set initial conditions
P=20	Proportional constant
I=2	Integral constant
VI=0	Initial value of integrator
AC 100000	Acceleration
DC 100000	Deceleration
JG0	Set X in jog mode
BGX	Start motion
#LOOP	Define subroutine
E=_TPY-_TPX	Following error
VI=E*I+VI	Integral term
S=E*P+VI	Total speed
JGS	Update speed
JP#LOOP	Repeat
EN	End

Application Example: Continuous Move

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

Instruction	Interpretation
#CONT	Label
AC 80000	Acceleration rate
DC 80000	Deceleration rate
JG 0	Start jog mode
BGX	Start motion
#LOOP	Define subroutine
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

ELECTROMATE

DMC-1200	DMC-1600	DMC-1700
DMC-1800	DMC-2000	

Application Example: Generating a Helical Motion Profile

Background:

It is possible to create a helical motion profile using any 3 (or more) axis controller. This is accomplished by commanding a coordinated circular move between 2 axes and gearing a third axis to the vector motion of the coordinated axes.

Program Example:

This example shows how to implement helical motion:

```
#HELIX
REM VARIABLES USED:
REM PITCH                        ENCODER CTS
                                 TRAVERSED/CIRCLE
REM SPEED                        VECTOR SPEED OF CIRCLE
REM RATIO                        GEAR RATIO
REM ANGLE                        SWEPT ANGLE OF CIRCLE
REM LOOPCNT USED IF              ANGLE>32000 FOR LARGE
TRAVERSES
REM LASTLOOP                     ANGLE FOR LAST LOOP
#SETUP
PITCH= 360
RADIUS= 5000
SPEED= 50000
VS SPEED
RATIO= PITCH/(2*3.14159*RADIUS)
GA,,S                            Specify Vector Speed as Master for Z axis
GR,,RATIO
IN "ENTER THE TRAVERSE DISTANCE (ENCODER CTS):",DIST
ANGLE=360 * DIST/PITCH
LOOPCNT=@INT[ANGLE/32000]
LASTLOOP= 0
JP#MOVE,ANGLE<32000
LASTLOOP=ANGLE - (LOOPCNT*32000)
ANGLE=32000
#MOVE
CNT=0
```

```
#LOOP
VMXY
CR RADIUS,0,ANGLE
VE
BGS
AMS
CNT=CNT+1
JP#LOOP,CNT<LOOPCNT
ANGLE=LASTLOOP
JP#LOOP,(CNT<(LOOPCNT+1)) & (LASTLOOP > 0)
EN
```

In the above example, a circle is created using the X and Y axes of the controller. The Z axis is geared to the coordinated motion of the X and Y axes to create the helix. The gear RATIO is determined by the PITCH of the helix and the circumference of the circle.

One unavoidable limitation of the circle (CR) command is that the maximum swept ANGLE of a circle must be less than 32,000 degrees. As a result, the program executes multiple passes of #LOOP for any calculated angle greater than 32,000 degrees.

Method to Increase Accuracy of Helical Motion

For the highest possible accuracy, it is recommended that the gear ratio be calculated using the formula **ratio=pitch/(2*pi*radius)** on a standard calculator. The resolution of the controller is limited to approximately four decimal places so the ratio you enter should be the gear ratio you calculate truncated to four decimal places. You will now need to calculate the amount of error you accumulate during the helical motion due to rounding so that the #CORRECT routine can compensate for it.

First, you need to determine the value of the gear ratio as internally represented by the controller. The controller stores decimal values as fractions with a resolution of 1/65,536. To determine the internal value of the gear ratio, multiply the entered gear ratio by 65,536, truncate any fractional part of the result, and divide by 65,536. You can also determine this directly by typing in a gear ratio (ex. GR ,,.0114) and then multiplying the _GRn operand by 10,000 and displaying the result (ex. MG _GRZ*10000). This result divided by 10,000 should give you the same result as the calculation method.

Next, determine the amount of error that results from the rounding of the gear ratio. To do this, you must calculate the distance which the geared axis will travel and compare it to the distance you specified using DIST. The actual distance traveled is equal to the internal ratio divided by the exact value of the ratio you calculated times DIST. Round this number to the nearest integer. Subtract this value from DIST and enter as DELTA. There is one more parameter to calculate: INTERVAL. INTERVAL is the number of counts it takes for the geared axis to accumulate one count of error. It is equal to the actual distance calculated above divided by DELTA. This value should be rounded down to the nearest integer.

Example:

The following example shows the calculations performed for the sample program above:

PITCH=360
RADIUS=5000
DIST=3999

ANGLE = 360*DIST/PITCH = 3999
EXACT RATIO = PITCH/(2*PI*RADIUS) = 0.011459155 \ Calculated value
RATIO = 0.0114 \Value entered in program
INTERNAL RATIO: RATIO*65536 = 747.1104 \ Truncate decimal part and divide by 65536

747/65536 = 0.011398315

ACTUAL DISTANCE = INTERNAL RATIO/EXACT RATIO * DIST
= 0.011398315/0.011459155 * 3999 = 3977.768 \round to nearest integer
= 3978
DELTA = DIST-ACTUAL DISTANCE = 3999-3978 = 21
INTERVAL = ACTUAL DISTANCE/DELTA = 3978/21 = 189.4286 \ round down
= 189

Application Example: Helical Motion

Example:

Perform a helical move with 20 turns with 1000 count radius, and a vertical length of 2000 counts.

Note that the total length in the XY plane is 20 • $2\pi R$=125,664
The Z move is 2000
Therefore, the gear ratio is 0.0159

Instruction	Interpretation
VM XY	Specify vector plane
CR 1000,0,7200	Define arc segment
VE	Vector End
VS 5000	Vector speed
VA 10000	Vector acceleration
VD 10000	Vector deceleration
GAS	Define vector motion as master
GR,,0.0159	Define 2 axis gear ratio
BGS	Begin sequence
EN	End program

ELECTROMATE

Motion Programming: Electronic Gearing

- Allows 1 to 8 axes per controller to be electronically geared to master encoder
- Eliminates mechanical gears
- User specifies:

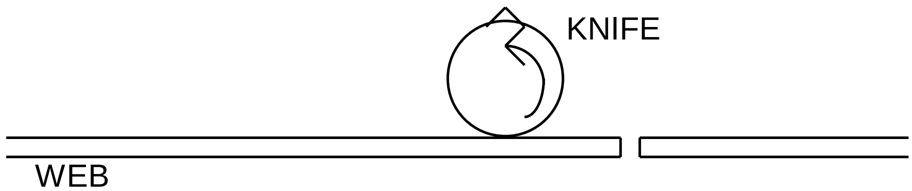
GA n where n=X	for main encoder as master n=CX for command position as master n=S for vector motion as master
GR	Gear Ratio(±127.9999 range)

- Can change Gear Ratio during motion
- Example:

Instruction	**Interpretation**
GA Y	Y is master
GR 5,,-2.5	X ratio = 5, Z ratio = -2.5
PR,1000	Master position
SP,100000	Master speed
AC,250000	Master acceleration
DC,250000	Master deceleration
BG Y	Begin Motion

Application Example: Rotating Knife

A rotating knife system, illustrated below, is used to cut webs of paper or plastics at a given length. The web moves independently and the knife is synchronized so that the blade moves at the same linear speed as the web during the cutting interval. When the cutting is completed, the knife is advanced or retarded in a manner that produces a required cut length.



Rotating Knife System

The method illustrated here applies with slight modifications to applications of printing on moving webs or applying labels.

Requirements

Web speed: Zero to 120 inches per second, independent motion (master)
Knife circumference: (π x diameter) = 20 inch
Required cut length: 16 to 40 inches, programmable
Resolution of web encoder: 1000 counts/inch
Resolution of knife: 10000 counts/rev or 500 counts/inch
Required accuracy: 0.005 inch noncumulative error

The knife is driven by a motor and an external amplifier. An encoder senses the position of the moving web.

Operation

The knife motion is synchronized with the web motion by electronic gearing. This, by itself, produces a cut length of exactly 20 inches. To achieve other cut lengths, the knife must be advanced or retracted an additional distance. This is achieved by generating some secondary motion on top of the electronic gearing. For example, to achieve a cut length of 16 inches, the knife is advanced a distance of 4 inches with a programmable speed and acceleration, on top of the geared motion.

The cutting cycle is performed by the following program. It assumes that the operation starts with the web at rest and the knife at the "up" position. Further, it is assumed that the cutting interval is $\pm 45°$ wide.

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

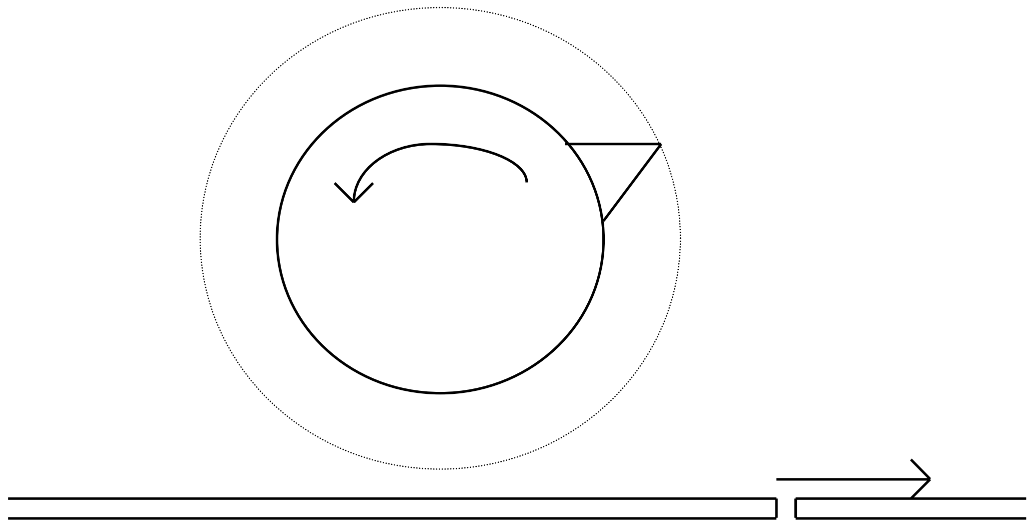
The correction motion is generated after the completion of the cutting cycle. The speed of the correction, S, equals 90% of the web speed in order to ensure that even under negative corrections, the knife never moves backward. The acceleration rate of the correction, A, is proportional to the square of the web speed to create an effect of cam-like motion.

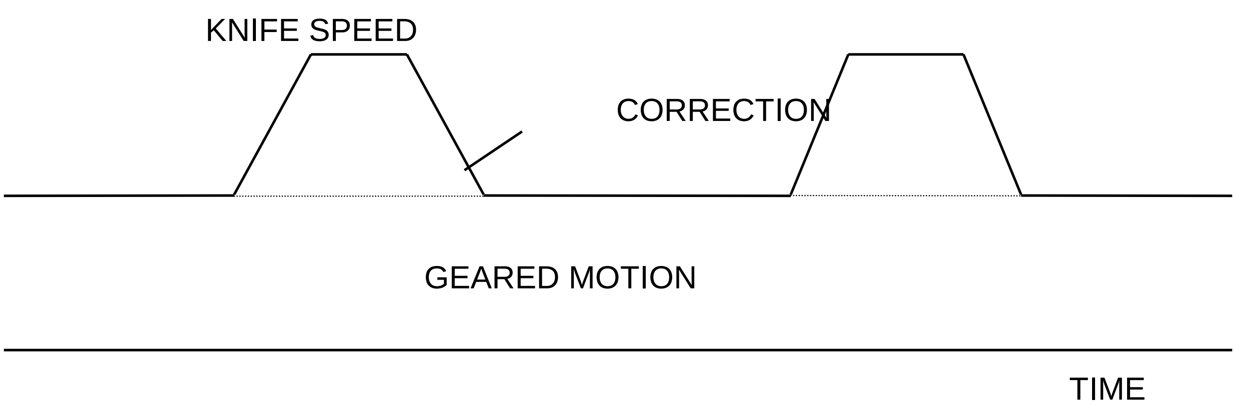
In the given system, the web encoder is connected to the Y axis and the knife is controlled by the X axis.

The process continues as long as the input I1 is high. When I1 is turned off, the knife gearing is terminated and the knife is decelerated to a gradual stop at the up position.

Instruction	**Interpretation**
#KNIFE	
L=23	Required cut length in inches
DP 0	Starting knife position = 0
GAY	Set Y as master
GR 0.5	Electronic gearing ratio
POINT = 6250	Point to start correction
C = (20-L)*500	Correction move in counts
MF POINT	Wait for correction point
V = _TVY/200	Read web speed
PRC	Correction distance
SP V*90	Correction speed
AC V*V*2.5	Correction acceleration
DC V*V*2.5	Correction deceleration
BGX	Start correction
AMX	Wait for completion of correction
POINT = POINT + 10000	Update next correction point
JP #LOOP, @IN[1]=1	Repeat if I1 = 1
PR 3750	Stopping distance
AC 60000000	Acceleration to stop mode
DC 600000	Stopping deceleration
SP _TVY/2	Transition speed
BGX	Start transition
GR0	Stop gearing
EN	End program

Application Example: Rotating Knife 2





- Knife circumference = 20"
- Desired cut length = 16"
- Knife encoder resolution = 4000 counts
- Conclusion, need to advance the knife 800 counts
- Contact interval = 90º = 1000 counts
- Starting condition -- Knife is up
- Distance to clear contact -- 2500 counts

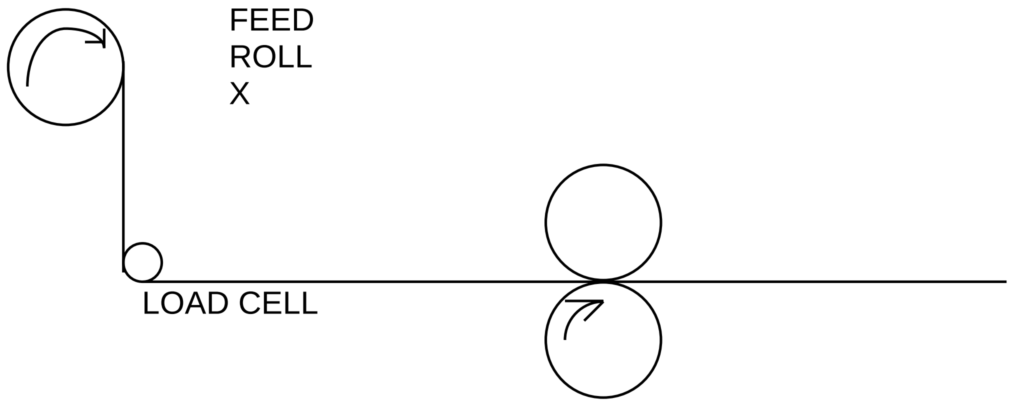
ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Instruction	Interpretation
DP 0	Define X position as zero
GAY	Set paper (Y) as master
GR 1	Set gear ratio 1:1
V=2500	Define variable
#WAIT	Define subroutine
JP #WAIT,_TPX<V	Wait to clear contact
#MOVE	Move correction distance
PR 800	Define distance
SP 20000	Define speed
BGX	Begin motion
AMX	After motion of X axis
V=V+4000	Update contact end point
JP #WAIT	Repeat cycle
EN	End program

ELECTROMATE

Application Example: Web Tension Control

Web processing applications often require tension control. In the most demanding case, the web is pulled by an independent process, which may be continuous or start/stop. The tension is sensed by a load cell and the requirement is to supply the web from a feed roll under constant tension.



The same process applies to winding the web on a take-up roll under constant or variable tension.

Requirements

Speed of master: 0-20 inches/sec, start/stop
Master encoder resolution: 400 counts/in
Feed roll diameter: 3.5 - 16 inches
Feed roll encoder resolution: 10,000 counts/rev
Load sensor output: 0-10V for 0-20 oz
Tension accuracy: ±0.5 oz

Operation

The length of the web pulled by the master process is monitored by an encoder with a resolution of 400 counts/in. The feed roll is driven by a motor with a tachometer and an encoder of 10,000 counts/rev. The motor is driven by an external amplifier. The amplifier is configured in the velocity mode (closing inner velocity tach loop) for added stability in view of the heavy inertial load.

The feed roll motion is divided into two parts: coarse and fine motion. The coarse motion is achieved by gearing the feed roll to the master process and continuously estimating the gear

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ratio. Since the ratio is not known precisely, this mode performs most of the required motion but not all of it.

The fine motion are correction moves performed on top of the electronic gearing in response to variations in tension. Here the errors in tension are monitored and the motor is driven at a proportional speed on top of the gearing.

It is assumed that the initial gear value, G, is known. The initial value can be computed or measured directly. The gear ratio may be continuously estimated by determining the ratio of the frequencies of the two encoders and filtering that ratio.

In the following program the feed-roll motor is controlled by the X axis and the master is monitored by the Y axis. The program consists of two parts: #GEAR, which estimates the gear ratio and performs the coarse mode, and #TRIM, which performs the fine move. The two programs are executed simultaneously by multitasking.

The load cell signal is applied to the analog input #1, and the required sensor level is 4 V. Accordingly, E, the difference between the sensor output and 4, is the error in tension. The motor is required to jog at a speed that is equal to
20 times E.

Instruction	**Interpretation**
#INITIAL	Label
GAY	Set Y as master
GRG	Initial gear ratio
JG0	Initial jog speed
BGX	Begin motion
XP = _TPX	Read initial X position
YP = _TPY	Read initial Y position
#GEAR	Label subroutine
Y = _TPY	Read new Y
X = _TPX	Read new X
JP #GEAR, Y = YP	Repeat if no Y motion
DX = X-XP	Compute X increment
DY = Y-YP	Compute Y increment
XP = X	Update X position
YP = Y	Update Y position
RATIO = DX/DY	Ratio of increments
G = (G*7+RATIO)/8	Estimate gear
GRG	Update gear
JP#GEAR	Repeat cycle
EN	End program
#TRIM	Label
E = @AN[1]-4	Tension error

JG E*20	Fine jog speed
JP #TRIM	Repeat
EN	End program

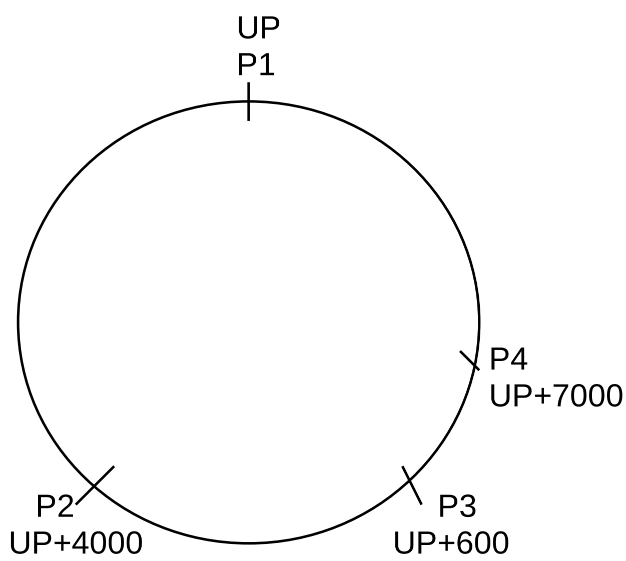
Application Example: Web Processing

Assume that the web is a plastic strip with printed labels. The objective is to cut the strip between the labels. The nominal length of each label is 40 cm. However, due to the elasticity of the web, the length may vary. To ensure cuts at the correct position, a periodic mark as added to indicate the exact cutting position. This mark is detected by a special sensor located exactly 40 cm ahead of the cutting knife.

To illustrate the control procedure, consider first The Blade Position Figure on the next page which defines positions along the rotating knife. Define point P_1 as the "up" position, which is defined initially as 0. The points P_2 and P_3 define the mark detection interval.

Under nominal conditions, where the length of the label is exactly 40 cm, the mark will be detected when the blade is exactly at the "down" position of P_1 + 5000. In that case, the correction is 2000 counts. This correction is made after the motor has reached the position P4. If the mark is detected at an earlier point, say P_1 + 4900, it indicates that the next label is short and, therefore, the correction move must be longer or, more specifically, 2100 counts. If no mark is detected between points P_2 and P_3, the nominal correction of 2000 counts is used. The actual program is given below.

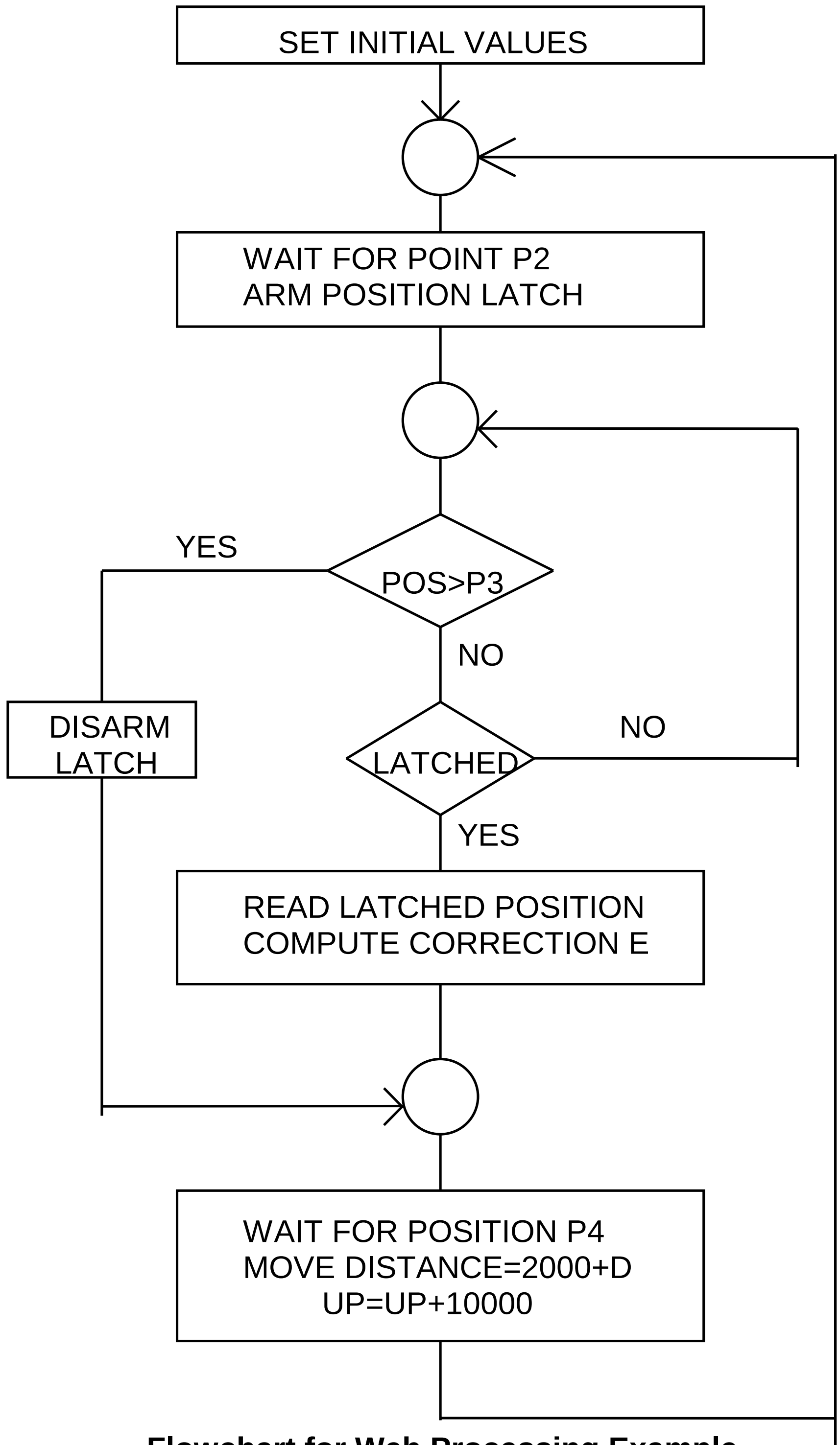
The reading of the blade position at the mark detection point is done with the "position latch" function. With this function, the motion controller can capture the position within a few microseconds, resulting in a precise reading. The program is given below along with the flowchart below



Definition of Blade Positions

Instruction	Interpretation
#MARK	Label
DP0	Define starting position as 0
UP = 0	Initial value UP position
GAY	Set Y as master
GR 0.1	Gear ratio 0.1
SP 50000	Speed of correction
AC 400000	Acceleration of correction
DC 400000	Deceleration of correction
#P2	Label subroutine
JP#P2,_TPX<UP+4000	Wait for point P2
ALX	Arm position latch
#LATCH	Label subroutine
JP#NOMARK,_TPX>UP+6000	If position >P3, skip
JP#LATCH,_ALX = 1	Wait for latch
E = UP+5000-_RLX	Second correction,E
JP#P4	Jump to subroutine
#NOMARK	Label subroutine
E = 0	Set E = 0
#P4	Label subroutine
JP#P4,_TPX<UP+7000	Wait for Point P4
D = 2000+E	Compute motion distance
PR D	Position correction
BGX	Start Move
AMX	Wait for end of correction
UP = UP+10000	Update position at UP
JP#P2	Repeat the process
EN	End of program

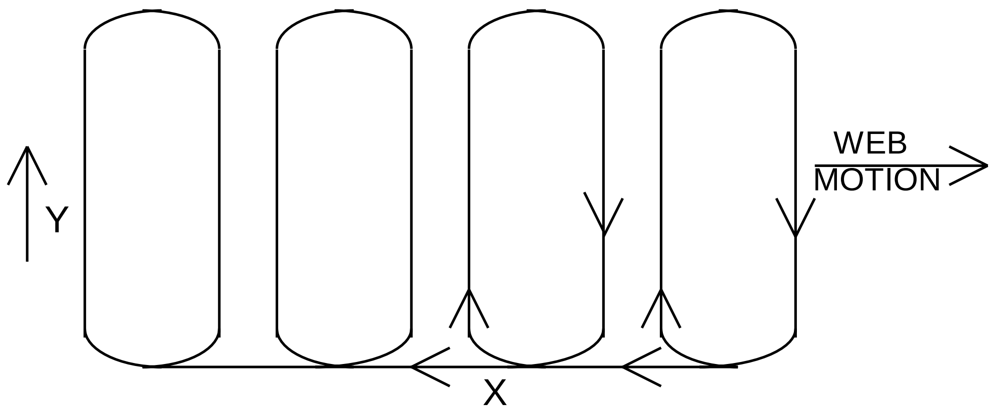
ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM



Flowchart for Web Processing Example

Application Example: Continuous Cutting of Moving Webs by Waterjet

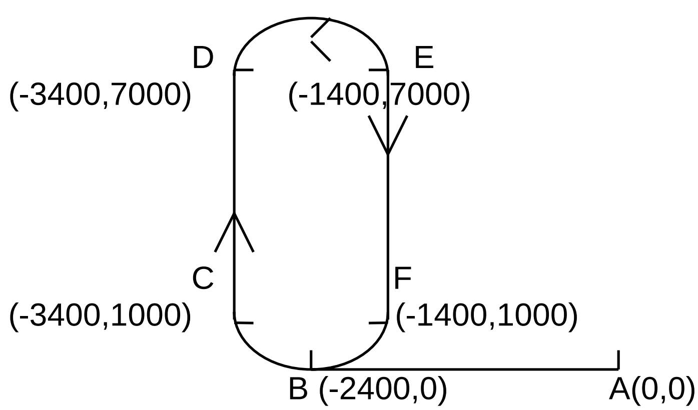
A moving sheet of plastic must be cut with a water jet along the pattern shown below. The process is continuous and the web does not stop.



Continuous Cutting Example

Operation

The unique feature of this system is that the plastic sheet moves continuously and independently at a variable speed. As a result, the motion of the X motor must equal the sum of two parts: motion 1 to follow the cutting pattern, and motion 2 to follow the moving web. This can be accomplished easily with the controller by separating the two moves. Part 2 is done by electronic gearing and part 1 is done by coordinated motion. The web motion is monitored by an independent encoder whose output is applied to the Z axis. Assuming that the resolution of all 3 encoders are the same, the X motor is then required to follow Z at a ratio of 1 to accomplish part 1 of the motion. The cutting cycle is shown below.



Cutting Cycle

The motion starts at the point A with the waterjet turned off (output bit 1 = 0). It moves toward point B, where the waterjet is turned on, and continues to C, D, E, F, and back to B which becomes the starting point of the next cycle.

In order to keep the waterjet location over the correct position, we must synchronize the cutting speed with the web speed. Note that the length of one cutting cycle, including the path from A to all the points, equals 20,680 counts. At the same time, the web advances 2400 counts. This implies that the vector speed, VS, must be equal to 8.616 times the speed of Y. This method, however, is not precise and will create some errors that can cause position drift of X. To eliminate this possibility, we define the starting point of the first cycle as X = 0. Later, we check the position of X at the start of all following cycles. If the starting position of X is positive, the cutting speed is slow and the speed is increased by a correction factor E.

Instruction	**Interpretation**
#WTRJET	Label
DP0	Define starting X position
GAZ	Set Z as master
GR1	X follows Z at 1:1
VMXY	Vector mode XY plane
#LOOP	Label subroutine
E = _TPX/2400	Drift correction factor
SPEED = _TVZ*(1+E)	Master speed
VS SPEED*8.616	Command vector speed
VP -2400,0	Motion AB
CR 1000,170,-90	Motion BC
VP -3400,7000	Motion CD
CR 1000,180,-180	Motion DE
VP -1400,1000	Motion EF
CR 1000,0,-90	Motion F to end
VE	End of motion
BGS	Start move
AV 2400	Wait for point B
SB1	Turn waterjet on
AMS	Wait for end of cycle
CB1	Turn waterjet off
JP #LOOP	Repeat the cycle
EN	End of program

ELECTROMATE

Application Example: Web Cutting with Fixed Cut Length

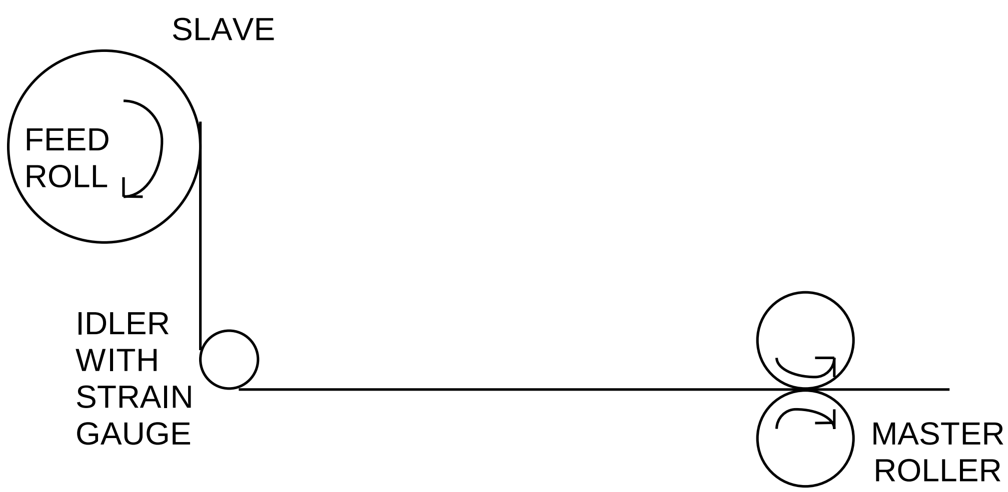
- To cut at a given length, add PR command on top of gearing.
  Knife circumference=20”
  Desired cut length=16”
  Knife encoder resolution=4000 counts
    Knife must be advanced
    4000/5=800 counts
- Position of knife to clear contact: 2500 counts

Instruction	**Interpretation**
#A	Label
DP0,0	Define X and Y position 0
GA,X	Define X axis ask master*
GR,.25	Set Slave Gear Ratio 1
V=2500	Position of Knife to advance
#LOOP	
MF,V	Wait for knife to reach pos.
PR,800	Advance knife position
BGY	Begin advancing movement
AMY	Wait for move to complete
V=V+4000	Add 1 Rev to next position
JP#LOOP	Repeat
EN	

*For DMC-1000, DMC-1500 controllers the gear command should be: GAX

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Application Example: Tension Control by Electronic Gearing



Objective: Master may run independently. Drive the slave motor to keep the tension constant.

Method 1 - Traditional PID

Add a tachometer to slave motor for damping. Close the position loop with an analog signal.

Note: If master starts and stops, this method limits the acceleration rate.

Method 2 - Electronic Gearing

Add an encoder to slave motor. Control in electronic gearing. Adjust gear ratio according to tension sensor output.

Note: As tension increases, we need to increase the gear ratio. The gear ratio has a slowly changing base, B, and a rapidly changing temporary value R. Total gear ratio is G.

Note: This method is most effective when gear ratio changes slowly.

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Instruction	Interpretation
#TENSION	Label
GAX	Define master
GR, 0.3	Initial gear ratio
JG,0	Set slave in jog mode
BGY	Start mode
XQ#GEAR,1	Execute gear estimation routine
#LOOP	Label
E=@AN[1]-4	Compute tension error
JG,E*20	Trim the speed
JP#LOOP	Repeat the process
EN	End

Instruction	Interpretation
#GEAR	Gear program
G=0.3	Initial gear value
XP=_TPX	Initial X position
YP=_TPY	Initial Y position
#LG	Label
X=_TPX	Read new X position
JP#LG,X=XP	Verify that X motion is non-zero
Y=_TPY	Read Y position
DX=X-XP	Find X increment
DY=Y-YP	Find Y increment
XP=X	Update previous X position
YP=Y	Update previous Y position
R=DY/DX	Compute ratio
G=(G*7+R)/8	Filter ratio and estimate gear
GRG	Adjust gear ratio
JP#LG	Repeat
EN	End

Application Example: Gearing Acceleration

The following program can be used to slowly engage electronic gearing. This is useful when the master axis is already moving, allowing for a smooth transition on the slave axis.

This program users the X axis as a master and the Y axis as a slave for illustration. Line numbers 6, 15, 23, 27, 31 must be modified if gearing is between different axes.

Instruction

```
#SMGEAR
TTL=10000
CT=TTL
GO=10
R=1
RATE=10
GA ,X
#LP3
JP#G,((CT=TTL)&(GO=1))|((CT=0)&(GO=-1))
JP#H, ((CT>0)&(GO=1))|((CT,TTL)&(GO=-1))
JP#STOP, GO=0
JP#LP3,GO=99
JP#TOOHI,@ABS[GO}>1
JP#LP3
#STOP
GRY=0
CT=TTL
JP#LP3
#TOOHI
GO=99
JP#LP3
#G;JP#LP3,(RATE<1)|(RATE>10000);AT0
#H;CT=CT+((GO*-1)*RATE)
GRY=(1-(CT/TTL))*R;AT-10
JP#LP3,(CT>=0)&(CT<=10000)
JP#ZERO,CT<0
CT=10000
GRY=0
JP#LP3
#ZERO
CT=0
GRY=R
JP#LP3
EN
```

The user is required to set the following variables:

1. GO is used to engage and disengage the gearing.
   GO=1 engages gearing
   GO=-1 disengages gearing
   GO=0 disengages gearing immediately

2. R is the gear ratio between the master axis and the slave axis. The default value for R is 1.

3. RATE is used to set the amount of time to be used to engage gearing. The time is given by the equation: T=(10000/RATE)*10msec. This variable must be a value between 1 and 10000 and the default value 10 (T=10sec). If the master is moving at a constant rate, the acceleration of the slave axis can be calculated as:

$(SPEED_{master}*R*RATE)/(10000*.01)$ per sec^2

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ELECTROMATE

Application Example:
Synchronizing 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

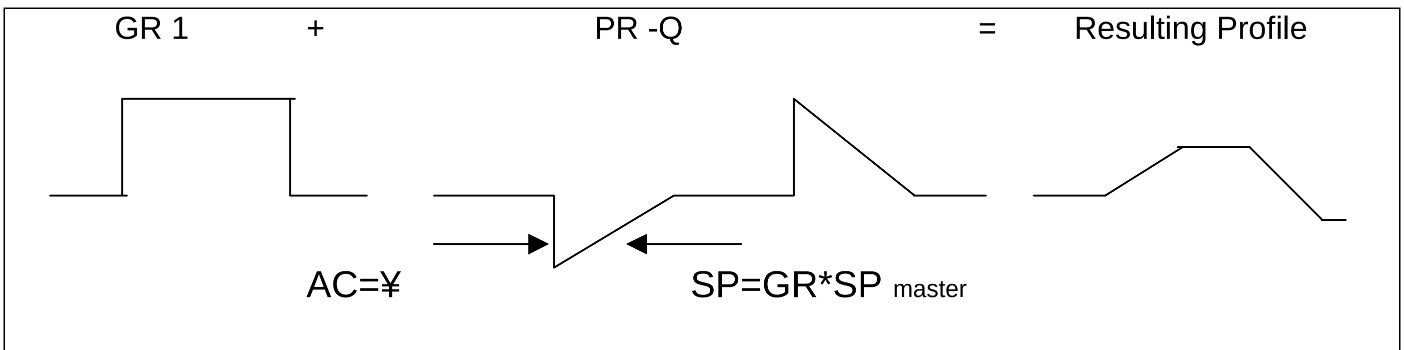
DMC-1200	DMC-1600	DMC-1700
DMC-1800	DMC-2000	

Application Example: Gearing Acceleration (Superimposed profile method)

Normal operation of the controller has a slave axis instantaneously gearing to the master upon execution of the GRn command. This can be undesirable in certain situations, such as the case when the master is already moving at the time the GRn command is issued. In this situation, it may be beneficial to engage the slave according to some acceleration rate, therefore ramping up to the speed of the master.

The following application program superimposes, or blends, two motion profiles to provide this acceleration. The first profile is a PR move on the slave axis with a known AC and DC. This profile is combined with the GRn profile to give the final acceleration to the gear ratio.

The theory behind this is as follows. Setting the AC of a negative PR move to the maximum number will cause instantaneous acceleration. The profile will then follow the DC rate to complete the move. When this profile is added to the instantaneous acceleration of the positive GRn command, the resultant profile has the slave 'ramping' to the speed of the master according the DC rate set.



Velocity profiles used for gearing acceleration

In this diagram, Q depends on the desired acceleration according to the equation, $Q=SP^2/2DC$

This opposite move counteracts the instantaneous acceleration of the GR command, resulting in the smooth acceleration. In order to ramp down, or decelerate from the gearing, a PR in the positive direction is given in conjunction with the GR0 command.

The limitation to this method is a slight jump in the motion as the gearing is engaged. This is due to the controller attempting to command the servo at the maximum acceleration. The magnitude of this jump is influenced by the speed of the master.

```
#GEARAMP
START=0;STOP=0
AC67107840
GAY
IN"SPEED OF THE MASTER (cts/sec)?",JOG
IN"FINAL GEAR RATIO?",RATIO
IN"TIME TO ACCELERATE (ms)?",RATE
RATE=RATE/1000
SPEED=(JOG*RATIO)
DECEL=SPEED/RATE
DC DECEL;SP SPEED
RAMP=(SPEED*SPEED)/(2*_DCX)
JG,JOG;BGY
MG"TYPE START=1 TO ENGAGE"
MG"OR TYPE STOP=1 TO ABORT"
#LOOP;JP#LOOP,(START=0)&(STOP=0)
JP#STOP,STOP=1
PR-RAMP;BGX;GR RATIO
#VELOOP;JP#VELOOP,(SPEED=>_TVX)&(STOP=0)
JP#STOP,STOP=1
MG"TYPE START=0 TO DISENGAGE"
MG"OR TYPE STOP=1 TO ABORT"
#LOOP2;JP#LOOP2,(START=1)&(STOP=0)
JP#STOP,STOP=1
GR0;PRRAMP;BGX
#VELOOP2;JP#VELOOP2,(_TVX>0)&(STOP=0)
JP#STOP,STOP=1
WT1000
STY
JP#GEARAMP
#STOP
AB1
JP#GEARAMP
EN
```

Application Example: 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:

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

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

DMC-1200	DMC-1600	DMC-1700
DMC-1800	DMC-2000	

Application Example: Variable Length Ecam Motion

Set up the ECAM function so that X is the master and Y is the follower. The ECAM profile will be set up to move the Y axis at the beginning of the ECAM cycle. Then the Y axis will wait for the X axis to 'roll over' and the cycle will start again. To cut different lengths, all we do is stop the ECAM cycle, change the length of the X axis rollover, then engage the ECAM mode again.

Here is an example of that program:

Assumptions:

1) X length is 600 counts or greater
2) 4000 encoder counts per Y revolution

```
#INIT                      Execute this code only
                           once at beginning
EAX                        Set X as ECAM master
EM LENGTH, 4000            Sets the roll over point of
                           X and Y
EP 100,0                   Set interval for table for
                           100 master counts

ET[0]=,0                   Enter ECAM table values.
ET[1]=,1000                Y will start moving
ET[2]=,2000
ET[3]=,3000
ET[4]=,4000                Y stops moving a full
                           revolution later
LOOP=5
#B
ET[LOOP]=,4000             Fill in table with the end
                           position
LOOP=LOOP+1
JP#B,LOOP<255
EN
#START                     To start ECAM motion
EB1                        Enable ECAM
```

EG,0	Engage ECAM
EN	

Insert code to change the variable 'LENGTH'

#CHANGE	To change the cut length
MF 3999	After position *end of Y axis motion*
EB0	Disable ECAM
EM	LENGTH,4000 Set new X rollover value
EB1	
EG,0	
EN	

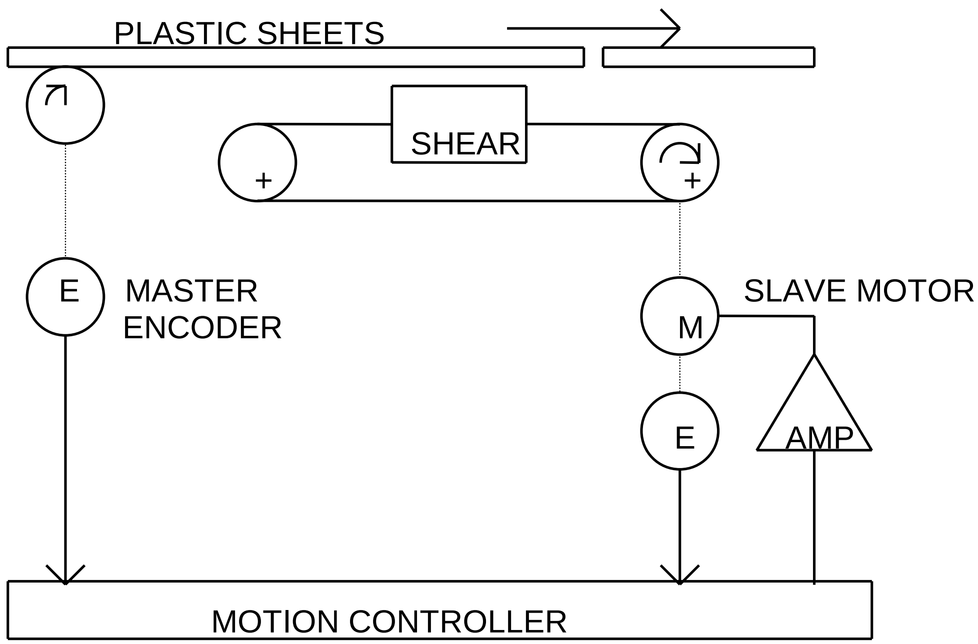
Note: X axis can be moving during this entire operation.

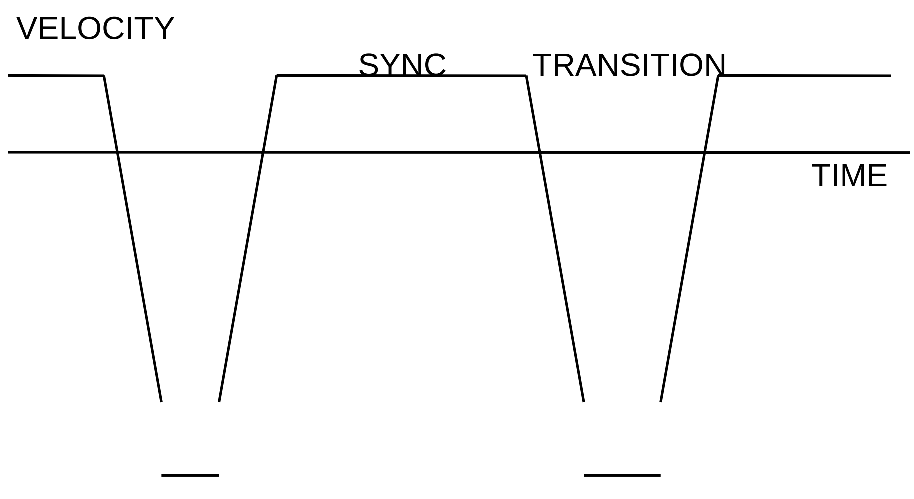
Application Example: Cutting Material on Moving Belt

- A plastic sheet is placed on a moving belt.
- The belt is driven open loop.
- The objective is to perform a circular cut on the sheet.
- Method:
  - Attach an encoder to the belt. Input to AUX X.
  - Belt encoder is master.
  - Y axis is geared to belt.
  - XY axes perform circular move.

#MOVE	Label program
VMXY	Specify vector motion path
GAY=DX	Define master encoder gear axis
GRY=1	Define y axis gearing
GMY=1	Set gantry mode
CR 5000,0,360	Perform circle
VS 7000	Define vector speed
VA 100000	Define vector acceleration
VD 100000	Define vector deceleration
VE	Vector end
BGS	Begin sequence
EN	End program

Application Example: Flying Shears





ELECTROMATE

Flying Shears Example:

Need to cut material in 12-inch intervals (60,000 counts per cycle). Flying shears must move forward 8 inches and return back rapidly.

Motion starts at X = 0
Turn cutter on at X = 5000
Turn cutter off at X = 35000
Start motion back at X = 40000

Instruction	**Interpretation**
#SHEARS	Label program
DP0	Define position zero
GAY	Select Y as master
GR1	Shears is slave
#WAIT1	Wait subroutine
JP #WAIT1,_TPX<5000	Shears between 0 - 5000
SB1	Turn cutter on
#WAIT2	Wait subroutine
JP #WAIT2,_TPX<35000	Shears between 5000 and 35000
CB1	Turn cutter off
#WAIT3	Wait subroutine
JP #WAIT3,_TPX<40000	Wait for end point
PR -60000	Return move
AC 1000000	Define acceleration
DC 1000000	Define deceleration
SP_TVY*4	Speed proportional to master
BGX	Begin X motion
AMX	After move of X motion
JP #WAIT1	Repeat process

Motion Programming: Tangent Motion

When two motors perform coordinated motion in a plane, it is often desirable to control a third motor in a manner whereupon its angle remains tangent to the direction of motion. This feature is useful in applications such as cutting cardboard with a knife. As the knife moves in the plane, it is necessary to keep the blade angle tangent to the motion trajectory. The user must specify several parameters to generate the tangent motion. First, the user must select the motors for the different roles. For the rest of this discussion, assume that X and Y generate the motion in the plane and Z is the tangent motor.

The second parameter is the resolution of the Z motor (or the number of units of resolution that will turn the Z motor 1°). For example, if the encoder resolution is 3600 counts per revolution, this parameter is 10.

Finally, the position of the Z motor at which its angle is 0° in the XY plane must be given. These parameters define the requirements for the tangent motion completely.

To generate tangent motion by a controller, first select the axes with the instruction VM. For example, the instruction:

```
VM XYZ
```

selects the first two axes, X and Y, to form the motion in the plane, and the third axis, Z, to form the tangent motion. The remaining two parameters are defined with the instruction TN m,n. For example, the instruction

```
TN 20,700
```

sets the resolution to 20 counts per degree. This also indicates that when the Z motor is at the absolute position of 700 counts, its angle in the XY plane is 0°.

Motion Programming: Proportional Motion

Another useful feature of motion controllers is their ability to generate additional controlled motion in proportion to the vector speed. This feature is useful with the generation of helical motion whereby two axes form circular motion and a third axis moves at a proportional velocity in the vertical direction. Proportional velocity is also useful in applications such as dispensing glue. Suppose that the motion in the XY plane is performed at variable speed; in order to produce uniform amounts of glue per unit of length, it is necessary to drive the glue pump at a rate that is proportional to the vector speed in the XY plane.

Electronic gearing can be used to implement proportional motion is done by Electronic Gearing (See previous examples). Once the motion in the plane is defined, that motion is defined as the master motion and a third motor is required to follow it at a specific gear ratio. The procedure is illustrated by the following example.

Consider an XYZ system where the resolution is 100 counts/mm for all axes. The objective is to generate a helical motion with 10 full turns of 5 mm radius in the XY plane and a height of 20 mm in the Z direction. The vector speed in the XY plane is 20 mm/s, and both the vector acceleration and deceleration equal 1000 mm/s^2. The motion parameters can be expressed in units of resolution by the parameters

radius = 5 mm = 500 counts
vector speed = 20 mm/s = 2000 counts/s
accel/decel = 1000 mm/s^2 = 10^5 counts/s^2

To determine the gear ratio, note that the path in the XY plane consists of 10 circles with a radius of 5 mm resulting in a total length of 314 mm. On the other hand, the motion in the Z direction is 20 mm. The ratio between the two motions is

ratio = 20/314 = 0.0637

which suggests a gear ratio of 0.0637

The required motion is generated by the following program.

Instruction	**Interpretation**
#HELICAL	Label
VMXY	Define XY plane
GAS	Select master motion
GR,,0.0637	Set Z as follower
CR 500,0,3600	Define 10 circles

VE	End of path
VS 2000	Vector speed
VA 100000	Vector acceleration
VD 100000	Vector deceleration
BGS	Start motion
EN	End of program

These data points are sent to the motion controller that interpolates between them before performing the motion. The following discussion presents another application of the contouring mode.

ELECTROMATE

Application Example: Constant Force

• Drive a motor against a load. Measure the force with an analog transducer. The objective is to maintain a constant force.

• **Method 1**: Modify the controller to accept analog feedback (standard option).

Suppose we need a force of 4 Volts

With 12 bit ADC	10 Volts = 2048 counts
	4 Volts = 819 counts

```
PA      819
BGX   Begin Motion
```

• **Method 2**: With standard controller, close the loop with incremental rotary encoder. Jog the motor continuously to hold the force constant. Make the jog speed proportional to the force error.

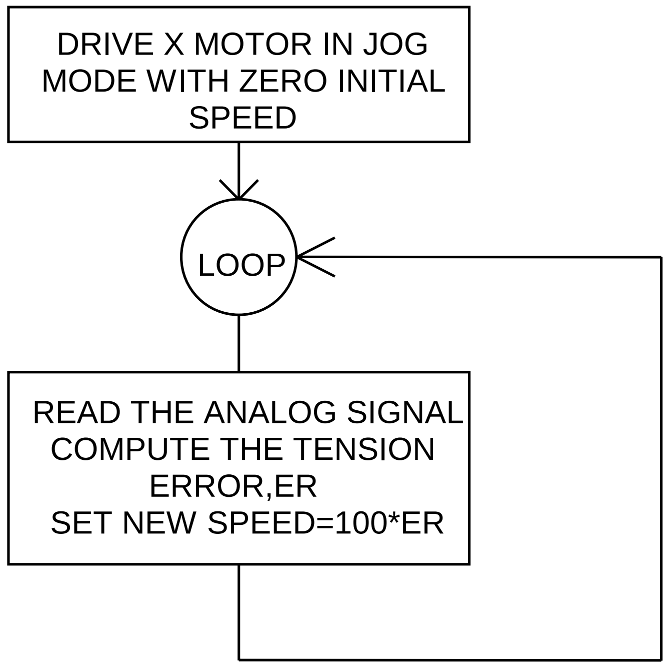
Instruction	Interpretation
#FORCE	Label program
JG 0	Set controller to jog mode
BGX	Begin motion
#LOOP	Label subroutine
E = 4 - @AN[1]	Compute proportional force error
V = E*1000	Define jog speed
JGV	Jog motor
JP #LOOP	Repeat process
EN	End program

• With multitasking, this program can be run in the background.

• **Constant Force Example 2:**

Drive the X motor against a slowly moving target and maintain a constant force between the two elements. The contact force is measured by a load cell whose output is applied to analog input #1. Assume that the desired force level corresponds to a signal level of 5V and that forward motion increases the tension between the two elements.

The simplest control procedure is to run the motor in the jog mode where the motor runs continuously at a specified speed. The controller reads the analog signal to determine the force level and sets the speed of the motor in proportions with the error in the force level. The flowchart and program is shown below.



Instruction	**Interpretation**
#FORCE	Label
AC 100000	Acceleration rate
DC 100000	Deceleration rate
JG 0	Set jog mode
BGX	Start motion
#LOOP	Label
ER=5-@AN[1]	Measure error in force level
VEL=ER*100	Set velocity
JG VEL	Update velocity
JP#LOOP	Repeat the process
EN	End

ELECTROMATE

Application Example: Joystick Control

Read the voltage of an X-Y joystick and drive the motors at proportional velocities.

10 Volts = 3000 rpm = 200,000 count/s

Speed/Analog Input = 200000/10=20000

Instruction	**Interpretation**
#JOYS	Label
JG 0,0	Set in Jog Mode
BG XY	Start motion
#LOOP	Define subroutine
VX=@AN[1]*20000	Read joystick and compute speed X
VY=@AN[2]*20000	Read joystick and compute speed Y
JG VX, VY	Change speeds
JP #LOOP	Jump to subroutine
EN	End program

ELECTROMATE

Application Example: Joystick with Nonlinear Function

Instruction	**Interpretation**
#NLNR	Label program
JG 0	Set in Jog Mode
BGX	Begin motion
#L	Define subroutine
A=@AN[1]	Read joystick
V=A*A*A+A*1000	Create nonlinear speed
JGV	Update speed
JP#L	Jump to subroutine
EN	End program

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ELECTROMATE

Application Example: Single Axis Joystick Routine with Deadband

Task : Read the voltage of a single axis joystick and drive the motors at proportional velocity. Also, do not affect motor speed if analog input is less than .5 volts (tolerance for offset when joystick is in center position).

INSTRUCTION	**INTERPRETATION**
#JOYSTK	Label 'Joystick'
JG0	Set Jog mode with Jog Speed 0
BGX	Begin motion on X axis
#LOOP	Label 'Loop'
V1=@AN[1]	Let V1 = value of analog input 1
JP#GO,.5<@ABS[V1]	Update speed if input is > .5 volts
V1=0	Otherwise, set jog speed to 0
#GO	Label
JGX=V1*1000	Change jog speed
JP#LOOP	Repeat
EN	End program

Application Example: Backlash Compensation

- An XY table is driven by two servo motors via leadscrews. The coupling has backlash. The system has two linear encoders and two rotary encoders.

- Effective where the objective is to move to a final point and stop precisely. Step 1 is to drive the motor with rotary encoders. After the move is complete, read the linear encoder and perform a correction.

- **Example:** Assume the resolution is 1 micron for both rotary and linear encoders. The objective is to move to the absolute position X=3000 Y=4000

Instruction	Interpretation
#DUAL	Label program
PA 3000,4000	Command motion
BGXY	Wait for completion
AMXY	After X & Y motion
WT20	Wait for settling
DX=3000 -_DEX	Compute X correction
DY=4000 -_DEY	Compute Y correction
PR DX, DY	Perform correction
BGXY	Begin motion
EN	End program

ELECTROMATE

Application Example: Teach Mode

Objective: Record the position of a motor over 4 seconds in 16 ms intervals, and play back.

Method: Step 1 - Record positions and store in array
Step 2 - Run in Contour Mode from array

Instruction	**Interpretation**
#RECORD	Label program
RA POS[251]	Select the arrays
RD_TPX	Select data
RC 4,251	Record 251 times at $16=2^4$ ms intervals
EN	End program
#RUN	To run the motor
CMX	Contour Mode
DT4	Time interval
C=0	Initialize counter
#LRUN	Label subroutine
D=C+1	Define variable
DX=POS[D]-POS[C]	Compute increment
CD DX	Contour data
WC	Wait for completion
C=C+1	Increment counter
JP #LRUN,C<250	Repeat 250 times
DT0	Stop Contour Mode
CD0	Contour data
EN	End program

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Program Flow: Multitasking

• Eight independent programs called "threads" can run simultaneously

• All threads have equal priority

• One thread can execute or halt another thread

• All trippoints in every thread available

• Main thread only uses INPUT command, IN. Input interrupts   main thread only

• Useful for background PLC functions; truly independent motions

• Example:

Commands:

XQ #Label,n
Execute Program Thread, where n=0
through 3 is thread number.
0 is main thread.

Hxn
Halt Execution of Thread

Application Example: Multitasking

Complex applications often require several independent tasks to be executed at the same time. The controller allows simultaneous execution of up to four independent applications programs. This is an ideal feature for executing independent operations of PLC tasks in the background.

Each separate task is defined as task 0 through 3. Any task can be executed or halted from another task. For example, XQ#POS1,0 begins Task 0. HX0 halts Task 0. All controller commands, including event triggers and conditionals, can be used in each task. However, input interrupts and input prompts are available only in Task 0.

Multitasking--Example 1: *Background PLC. Functions Example:*
This example shows how four independent programs can be executed simultaneously from the controller memory. The main task #MAIN starts all the other tasks--#PLC1, #PLC2, #MOVE--after Input 1 is high. It halts all tasks when Input 1 goes low. #PLC1 sets Output 2 only when Input 2 and Input 3 are high. #PLC2 generates a waveform on Output 1 which is high for 10 msec and low for 40 msec. #MOVE moves the X axis 100 counts, repetitively. The program is illustrated on the following page:

Instruction	Interpretation
#MAIN	Main Program -- Task 0
AI1	After Input 1 high
XQ#PLC1,1	Execute Task 1
XQ#PLC2,2	Execute Task 2
XQ#MOVE,3	Execute Task 3
AI-1	After Input 1 low
HX	Halt all tasks
EN	End program
#PLC1	#PLC1 -- Task 1
OB2,@IN[2]&@IN[3]	Set Output 2 if Input 2 and 3 high
JP#PLC1	Loop
#PLC2	#PLC2 -- Task 2
AT0	Set reference time
#LOOP	Loop label
SB1;AT10	Set Output 1; wait 10 msec
CB1;AT-50	Clear Output 1; at 50 msec; reset timer
JP#LOOP	Loop
#MOVE	#MOVE -- Task 3
PR100;BGX;AMX	Move 100 counts
WT20;JP#MOVE	Wait 20 msec and repeat

Multitasking -- Example 2:

Instruction	Interpretation
#X	X-Thread
PR 1000;BGX;AMX	Move 1000
PR -1000;BGX;AMX	Move -1000
JP #X	Repeat Motion
#Y	Y-Thread
PR,500;BGY;AMY	Move 500
PR,-500;BGY;AMY	Move -500
JP #Y	Repeat Motion
#TIME	I/O Thread
AT50;SB1;AT10;CB1	Every 50 msec Set Bit 1
JP #TIME	Repeat
#MAIN	Main Thread
JP #EXIT,@IN[1]=1	If input 1 high, exit
JP #MAIN	Loop if input 1 low
#EXIT; HX	Halt all threads
EN	End Program

• To execute:

```
XQ #MAIN,0
XQ #X,1
XQ #Y,2
XQ #TIME,3
```

Application Example:
Using Multitasking To Produce a Waveform on Output 1 Independent of a Move

Instruction	Interpretation
#TASK1	Task1 label
AT0	Initialize reference time
CB1	Clear Output 1
#LOOP	Loop1 label
AT 10	Wait 10 msec from reference time
SB 1	Set Output 1
AT -40	Wait 40 msec from reference time, then initialize reference
CB 1	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 (i.e. Thread 0). #TASK1 is executed within TASK2.

Application Example: Arrays

Arrays are structured memories where data can be stored and retrieved in a certain order. They are useful for storing sequences of position points or output signals.

An array is characterized by a name and a size. Each element in the array is identified by its index. For example, the array XPOS may have a size of 100 units. As a consequence, each point is identified as XPOS [N] where N varies between 0 and 99.

To store data in the array, we use an instruction such as:

```
XPOS [5] = _TPX
```

which reads the current position of the X axis and stores the value in the array. The value of the position may be later retrieved with instructions of the form:

```
X = XPOS [5]
```

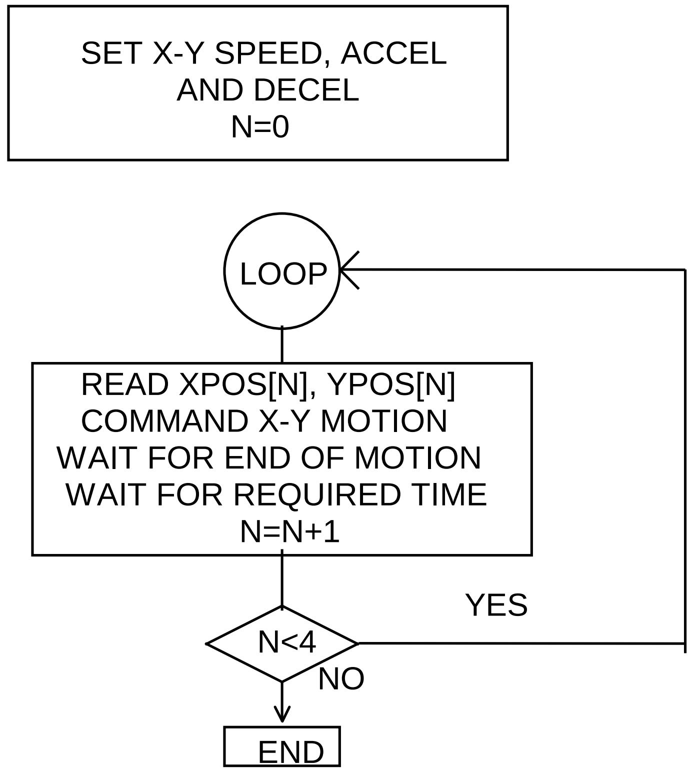
which transfers the position value to the variable X. The use of arrays is illustrated by the following example.

The controller is required to move the X & Y axes of a positioning table to four positions characterized by their coordinates. The values of the required points are stored in the arrays XPOS and YPOS. Once the controller reaches the specified point, it must wait a certain amount of time before resuming the motion. The waiting time in milliseconds is stored in the array WAIT.
The program is described in two parts. The first part, #STORE, defines the arrays. The second part, RUN, performs the moves according to the requirements.

Instruction	**Interpretation**
#STORE	Label
DM XPOS [4], YPOS [4], WAIT [4]	Define arrays
XPOS [0] = 100	Set values
XPOS [1] = 320	
XPOS [2] = 450	
XPOS [3] = 500	
YPOS [0] = -100	
YPOS [1] = 20	
YPOS [2] = 153	
YPOS [3] = 200	
WAIT [0] = 50	

WAIT [1] = 100	
WAIT [2] = 60	
WAIT [3] = 120	
EN	End program

Instruction	**Interpretation**
#RUN	Label
AC 200000,200000	Accelerations
DC 200000,200000	Decelerations
SP 50000,50000	Speeds
N = 0	Initial count
#LOOP	Label subroutine
PA XPOS [N], YPOS [N]	Specify final position
BGXY	Start motion
AMXY	Wait for completion
WT WAIT [N]	Wait specified time interval
N = N+1	Increment index
JP#LOOP, N<4	Repeat 4 times
EN	End program



Flowchart for Array Program

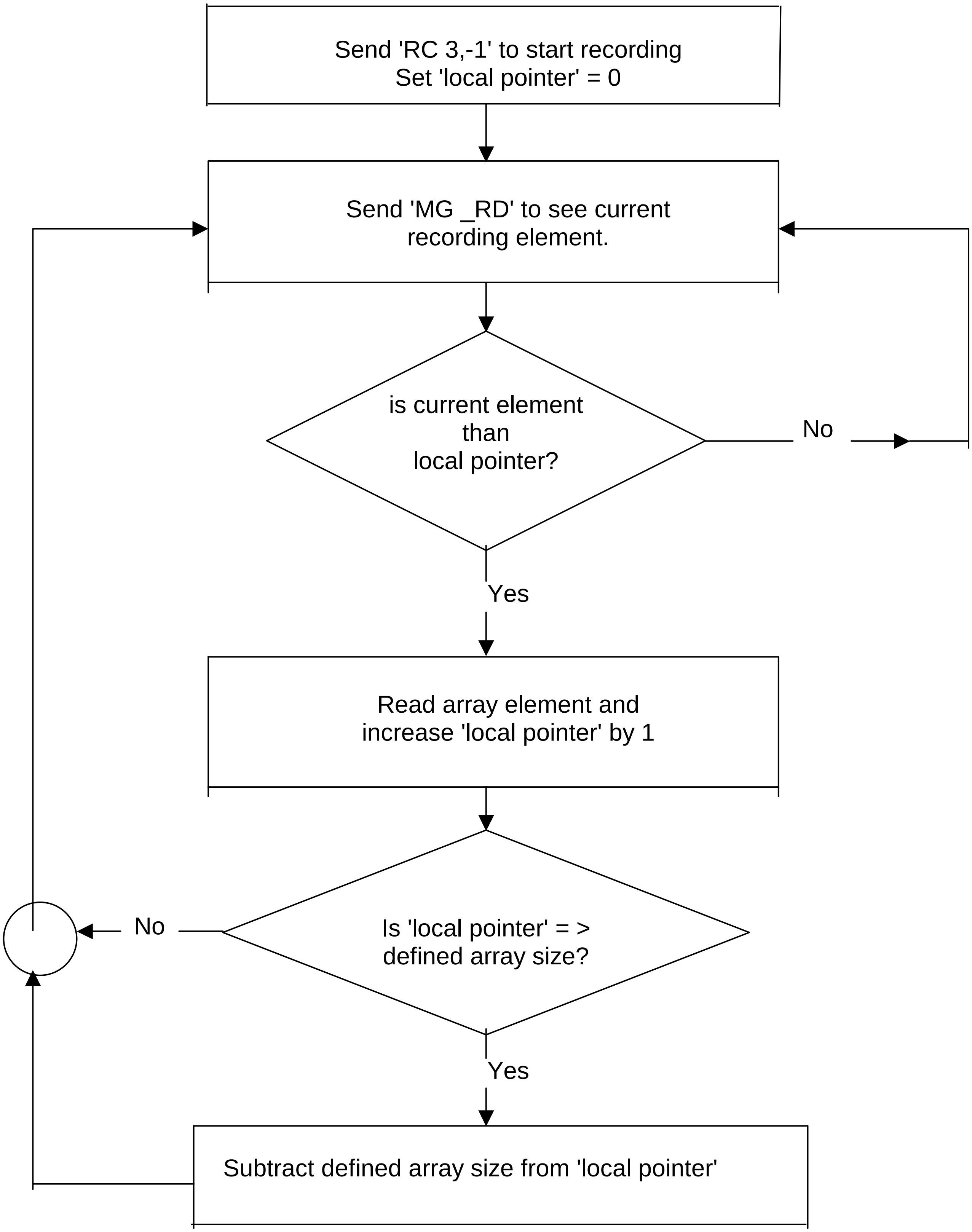
One of the main uses of the array is for recording motion as described in the following pages.

DMC-1200	**DMC-1600**	**DMC-1700**
DMC-1800	**DMC-2000**	

Motion Programming: Infinite Array Recording

The following note explains the ability to collect an infinite amount of data. Using the Record Array mode, data is captured and stored in the array space within the controller. The number of data points available is limited to 8000 for the OPTIMA Series controller; with the infinite record feature, the array space is limited only by the size of the hard drive on a host computer. To accomplish this, data is captured and written to the first array element, then the second, and third, and so on. When the end of the array is reached the controller will start at the first element of the array and overwrite the data contained there. This 'loop' will continue until a command is given to stop the Record Array function. The host computer must read the array elements before they are overwritten. This is done using the _RD command. _RD returns the current array element that was just written to by the Record Array function. By reading the array elements as fast as they are being recorded the system can record as much data as can fit in the hard drive. Here is a flow chart for the operation of the host computer program to read the captured data: (See Diagram on next page)

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM



Send 'RC 3,-1' to start recording
Set 'local pointer' = 0
Send 'MG _RD' to see current
recording element.
is current element
than
local pointer?
No
Yes
Read array element and
increase 'local pointer' by 1
Is 'local pointer' = >
defined array size?
No
Yes
Subtract defined array size from 'local pointer'

Here, the host computer reads one element at a time and is always looking at the state of the controller. If the array size is large and the capture rate slow, the host computer could look at the controller less often and upload more than one array element. Please note the command 'RC 3,-1' starts the recording from the host.

An example of the commands needed to set up infinite recording is below. These commands set up the Record Array function to capture the X axis position and store it in an array of 1000 elements:

DM POSX[1000] define array
RA POSX[] specify which arrays to record
RD _TPX specify data to be recorded ' TPX'

The host computer program starts the recording with the 'RC' command. To stop recording send the command: 'RC 0'

ELECTROMATE

Application Example: Array Data Storage

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
CONT=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
CONT=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

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

ELECTROMATE

Application Example:
A Method To Increase Array Space For Use With Record Array

When using the record array function, it may be desirable to increase the total number of recorded data points in the controller's memory. The following method takes advantage of the fact that, often times, the recorded data elements are smaller than the array elements. In this case, the recorded data can be packed.

To illustrate, consider a system with one motor which has a complete travel of+/-30000 counts. If the motor position is to be recorded (_TPX), each data point will only require 16 bits of information (16 bits can represent the numbers up to +/-32767). Since each array element on the Galil controllers is 48 bits – each array element can hold 3 data elements.

An Example:

To further illustrate, the following program commands the X axis to move in a sinusoidal motion over a

#RECORD	Record Function
DP 0	Define Position
DA *[]	De-allocate all arrays and variables
DM CB UFF[1002], FINBUFF[6000]	CBUFF record array, FINBUFF- final array
COUNT=0 CBUFF	Routine to clear
#CLRC	
CBUFF[COUNT]=0	
COUNT=COUNT+1	
JP#CLRC, COUNT<1002	
COUNT=0 FINBUFF	Routine to clear
#CLRA	
FINBUFF[COUNT]=0	
COUNT=COUNT+1	
JP#CLRA,COUNT<6000	Set Record Update Time to 2msec
RTIM=2	Set up sine motion on X axis (demo)
VMXN	
VS 1000	
CR 8000,0,36000	Set up record function on CBUFF

RA CBUFF[]	Record position Information
RD_TPX	Reset counter to zero
COUNT=0	Reset array index to zero
INDEX=0	Begin circular array recording (1002 pts)
RC RTIM,-1002	Begin Motion Sequence
BGS	Define temporary variable, TEMP
TEMP=0	
HIGH=0	Define temporary variable, HIGH
PTR=0	Define temporary variable, PTR
#LOOP	Main Loop
PTR=_RD	Set PTR to last recorded array element
WT 10	Wait 10 msec
JP #ROLL, PTR<HIGH	Jump to ROLL if record array ‘rolled over’
HIGH=PTR	If not, set HIGH to last element recorded
JP#LOOP,(PTR<(30+INDEX))	Loop until 30 points have been recorded
#UPDATE	Otherwise, update the final array
TEMP=CBUFF[INDEX]	
JS#SIGN, TEMP<0	
TEMP=(TEMP&32767)	
FINBUFF[COUNT]=65536*TEMP	Pack first element into top of FINBUFF
INDEX=INDEX+1	Update index
TEMP=CBUFF[INDEX]	
JS#SIGN,TEMP<0	Set highest bit to 1 if negative
TEMP=(TEMP&32767)	
FINBUFF[COUNT]=FINBUFF[COUNT]+TEMP	Pack 2nd element into top of FINBUFF
INDEX=INDEX+1	Update index
TEMP=CBUFF[INDEX]	

```
JS#SIGN,TEMP<0
TEMP=(TEMP&32767)
FINBUFF[COUNT]=FINBUFF[COUNT]+(TEMP/65536)
                                              Pack 3rd element into top of FINBUFF
INDEX=INDEX+1
COUNT=COUNT+1
JP#REPLAY,COUNT>=6000
#CHECK
JP#UPDATE,(INDEX<(PTR-3))
JP#LOOP
#ROLL
TEMP=CBUFF[INDEX]
JS#SIGN,TEMP<0
TEMP=(TEMP&32767)
FINBUFF[COUNT]=65536*TEMP
INDEX=INDEX+1
TEMP=CBUFF[INDEX]
JS#SIGN,TEMP<0
TEMP=(TEMP&32767)
FINBUFF[COUNT]=FINBUFF[COUNT]+TEMP

INDEX=INDEX+1
TEMP=CBUFF[INDEX]
JS#SIGN,TEMP<0
TEMP=(TEMP&32767)
FINBUFF[COUNT]=FINBUFF[COUNT]+(TEMP/65536)
INDEX=INDEX+1
COUNT=COUNT+1
JPREPLAY,COUNT>=6000
JPROLL,INDEX<1002
INDEX=0
HIGH=0
JP#UPDATE
#REPLAY                                       Playback routine
STX
AMX
```

Application Example: Recording

Controllers can record motion data at a fixed rate and store the data in an array. This data can be later used to duplicate the motion or to analyze it.

The recording process includes several steps: First the array where the data is stored must be specified, and later the type of data, such as position, position error, etc., must be defined before the actual recording is accomplished at a specified rate. The process is illustrated by the following example.

Write a program that records the position error of the Y axis every 16 msec, a total of 100 times, and stores the results in the array YERROR.

The actual recording is done with the instruction RC that has two parameters: the first parameter, n, defines the recording time interval as 2^n msec. In the given example, n = 4 results in 16 msec intervals. the second parameter sets the number of the recorded points.

Instruction	**Interpretation**
#RECORD	Label
RA YERROR [100]	Storage array
RD _TEY	Data type
RC 4,100	Actual recording
EN	End program

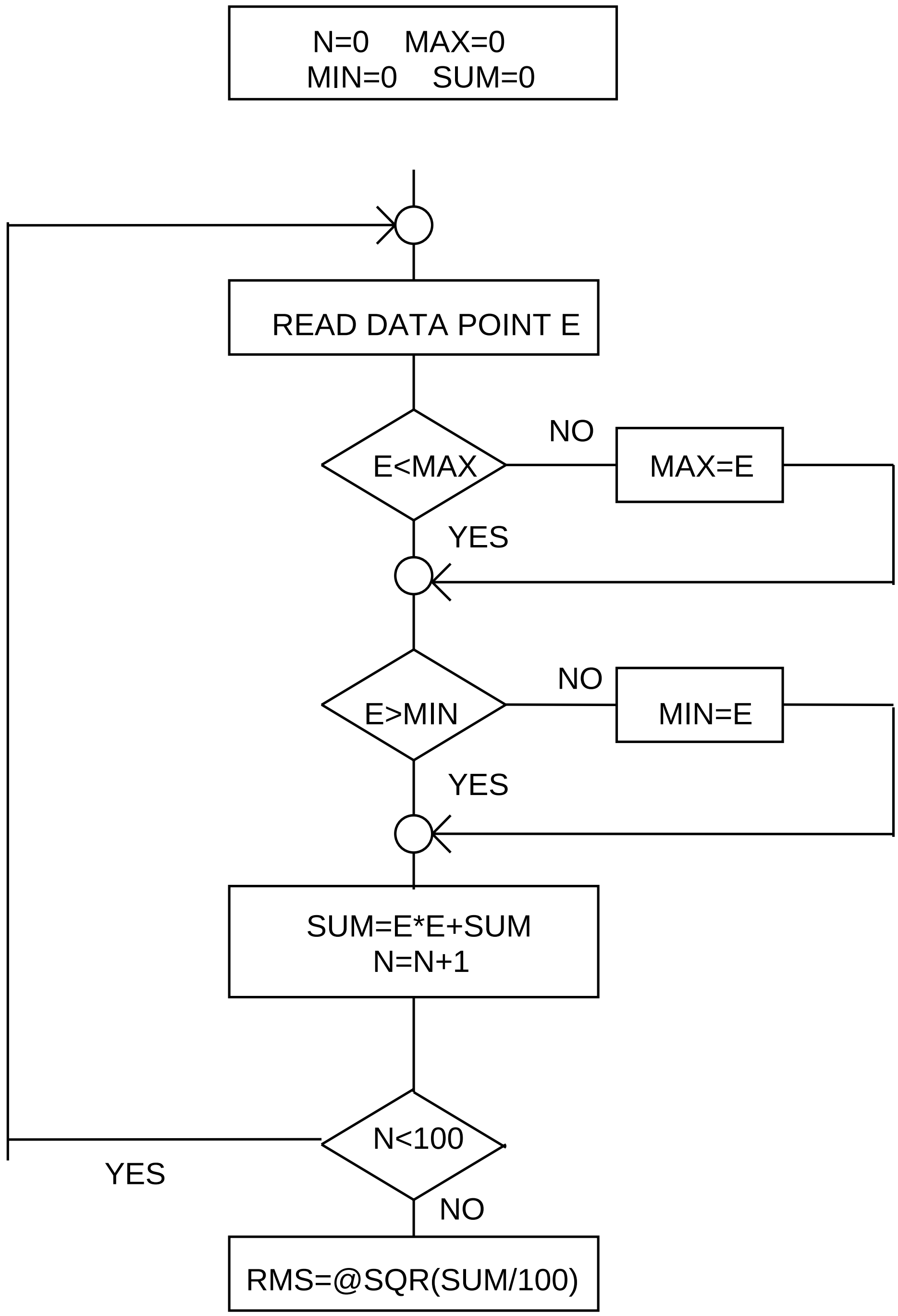
The following example illustrates how recorded data can be used for analysis.

Use the position error data collected in the previous example to perform a statistical analysis on the position error of Y. Determine the maximum and the minimum values as well as the mean square value. The following program performs the required tasks and stores the results under the variables MAX, MIN, and MEAN.

Instruction	**Interpretation**
#STAT	Label
N = 0	Initial values
MAX = 0	
MIN = 0	
SUM = 0	
#LOOP	Label
E = YERROR [N]	Read position error
JP#MAX, E<MAX	Compare with MAX
MAX = E	Redefine MAX

ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

```
#MAX                    Label
JP#MIN, E>MIN           Compare with MIN
MIN = E                 Redefine MIN
#MIN                    Label
SUM = E*E+SUM           Compute sum of squares
N = N+1                 Increment index
JP#LOOP, N<100          Repeat 100 times
MEAN = SUM/100          Mean square
ROOT = @ SQR[MEAN]      Root mean square (RMS)
EN                      End program
```



Flowchart for #STAT Program

Application Example: Record and Play Back

The contour mode is an effective tool for performing record and play-back types of motion. In these applications, the motion is originally generated by manual means. While the motion takes place, the positions of the motors are recorded. Later, the recorded positions are used as the reference positions for repeating the moves. The process in analogous to that of recording music and playing it back.

The first part of this process is recording the motion. It requires a controller with the ability to record and store the positions of the motors involved with the motion at the specified times. An advanced level controller, for example, performs the recording automatically at fixed time intervals and stores the data in it's array.

The recorded data can be used as the source for the repeated moves. The contour mode receives position commands from the stored data and performs it as required. To illustrate the process, consider the following example.

Record And Playback Example:

Consider a robot arm with three degrees of freedom. Each joint can be controlled with an independent motor. The objective is to move the arm manually along a certain trajectory with the motion lasting 12 seconds. The motion is to be repeated later automatically.
The first step is to select the time interval as a basis for the recording time. Because the motion is performed manually, it is unlikely to include abrupt changes in velocity. Therefore, we can record the motion at relatively long time intervals of 2^5 = 32 ms without any loss of information. The number of sampling intervals, which is the ratio between the total motion time and sampling interval, equals

$$N = 12/0.032 = 375$$

The recording is performed with the following program. It starts by defining the arrays and assigning them for the recorded data. Later, the type of recorded data is noted. In the given example, the positions of the three motors are requested. If we assume that the recording is to start upon a pulse on Input 1, the instruction AI1 delays the start of the recording to the required moment.

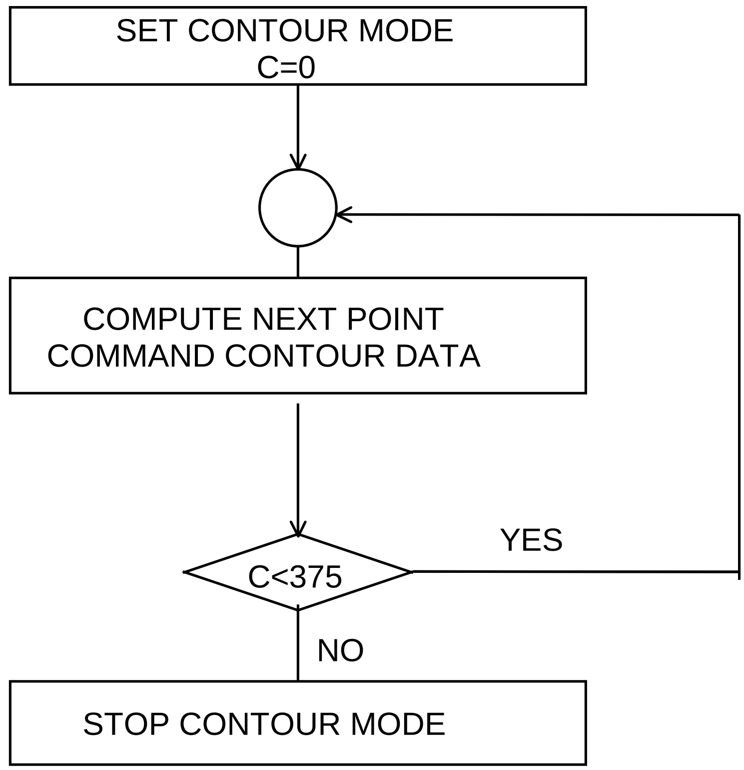
ELECTROMATE INDUSTRIAL SALES LTD.
TOLL FREE PHONE (877) SERVO98 • TOLL FREE FAX (877) SERVO99
SALES@ELECTROMATE.COM • WWW.ELECTROMATE.COM

Instruction	Interpretation
#RECORD	Label
DM XPOS[376], YPOS[376], ZPOS[376]	Define arrays
RA XPOS[376], YPOS[376], ZPOS[376]	Assign arrays for recording
RD _TPX, _TPY, _TPZ	Define what to record
AI1	Wait for start pulse
RC 5,376	Start automatic recording
EN	End of program

The motion is generated by the contour mode. Note, however, that the recorded data is expressed in absolute positions whereas the contour mode commands are expressed in position increments. This implies that the position increments must be computed from the absolute positions before the start of the motion. The process is illustrated by the program flowchart on the next page and the program below.

Instruction	Interpretation
#PLAY	Label
CMXYZ	Set contour mode
DT 5	Time interval 2^5 = 32 ms
C = 0	Set index
#LOOP	Define subroutine
D = C+1	Next index
DX = XPOS[D]-XPOS[C]	Compute increments
DY = YPOS[D]-YPOS[C]	
DZ = ZPOS[D]-ZPOS[C]	
CD DX,DY,DZ	Command contour segment
WC	End of segment
C = C+1	Increment index
JP#LOOP, C<375	Repeat if necessary
DT 0; CD0	End contour
EN	End of program

A desirable feature in the playback process is the ability to control the speed. The program shown above repeats the motion at the speed of recording. Speed variations can be performed in different ways; the advanced level controller allows changing the speeds by factors of 2 with the DT instruction. Note the third instruction of the program #PLAY. If that instruction is changed to DT4, for example, it shortens the contour time interval to 16 ms, resulting in a velocity that is twice as high as the recording velocity.



Robot Arm Example

For finer changes in speed, the user can vary the sample time. The instruction DT5 sets the contour time interval to 32 sampling periods, with a default value of 1 ms each. Changing the sampling interval to 875 μs, for example, shortens the contour time interval to 28 ms, resulting in a 14% increase in speed.

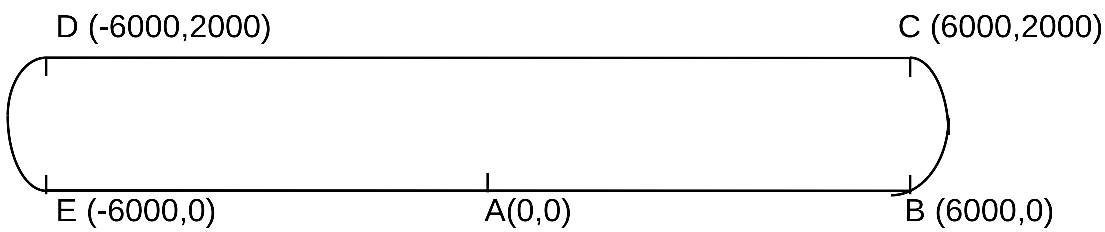
Application Example: Feedrate Override

In some cases it is desirable to give the machine operator the ability to adjust the machine speed. This ability is called feedrate override. This feature may be useful, for example, when starting a machine for the first time; the operator may wish to increase the speed gradually.

The method of speed control is often done by a potentiometer with an output voltage between 0V and 10V. When the potentiometer output is at the full 10V, the machine is supposed to run at full speed. In all other cases, the speed should be proportional to the potentiometer voltage.

To accomplish this function, motion controllers read the potentiometer voltage and adjust the feedrate in proportions.

Consider the motion path described by the following example starting at Point A and moving towards B. The motion is in the XY plane, the radius of the corners is 1000 counts, the vector speed is 20,000 counts/s and the vector acceleration and deceleration rates are both 100,000 counts/s^2.



Motion Path for Feedrate Override

The instructions and their interpretations are shown below.

Instruction	**Interpretation**
#MOVE	Label
VM XY	Specify XY plane
VP 6000,0	Move to Point B
CR 1000,270,180	Move to Point C
VP -6000,2000	Move to Point D
CR 1000,90,180	Move to Point E
VP 0,0	Return to Point A
VE	End of path
VA 100000	Vector acceleration
VD 100000	Vector deceleration
BGS	Start motion
EN	End of program

Repeating the motion of the previous example with feedrate override, the operator generates a voltage in the range between 0V and 10V. This voltage is applied to Analog Input #1 to set a proportional feedrate.

The motion is generated by the two programs, #MOVE and #SPEED. Both programs operate simultaneously in multitasking. The program #SPEED reads the analog inputs and adjusts the vector speed continuously. In the second program, #MOVE, the vector speed is not specified. The two programs are listed below.

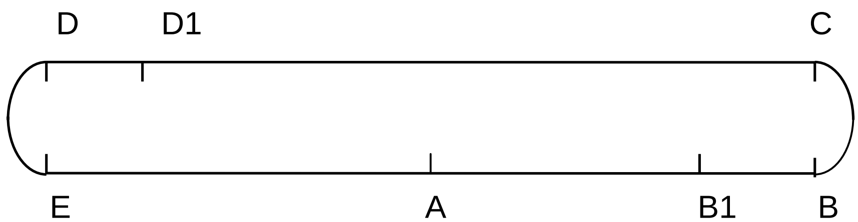
Instruction	**Interpretation**
#SPEED	Label
A = @AN[1]	Read analog voltage
V = A*2000	Set feedrate
VS V	Update feedrate
JP #SPEED	Repeat the process
EN	

Application Example: Variable Feedrate

Some motion control processes, such as an engraving, do not require a constant feedrate. In those cases, the objective is to complete the motion in the shortest time possible while keeping the position error within tolerance. The challenge is, therefore, to determine the highest velocity allowed under those conditions.

The position errors in typical motion control systems are proportional to the acceleration rates. This implies that, in order to limit the errors, we must limit the acceleration and deceleration rate.

Two types of acceleration exist: linear and centrifugal. Linear acceleration occurs when the XY motion control system changes its feedrate while moving along a straight line. Such an acceleration can be controlled directly and, therefore, can be easily limited. Centrifugal acceleration, on the other hand, is the radial acceleration that occurs when an XY system performs a move along a circular arc. Such an acceleration is a function of both the feedrate and the radius of the arc and, therefore, can be controlled indirectly by limiting the feedrate along the arc. The process of limiting the feedrate according to the radius of the arc and the allowed acceleration is shown below.



Motion Path for Variable Feedrate Example

Note that the motion starts at Point A and moves toward Point B. The initial velocity is high because the first segment is a straight line. Upon reaching Point B1, the motion controller must start a gradual deceleration so that the velocity at Point B is low. After completing a circular arc, Point C, the speed is increased and later it is lowered again at Point D1. Finally, at Point E, the speed is raised to high level before the system comes to a stop at A.

In order to program such velocity changes, we need to determine the location of the transition points along the path.

For example, let the high and low speeds be 20,000 and 10,000 counts/s, respectively, and let the vector acceleration and deceleration rates be 100,000 counts/s^2. To determine the deceleration distance, note that the deceleration time is

Time=Speed change/Deceleration=10000/100000=0.1second

Because the average speed on this interval equals 15,000 counts/sec, the deceleration interval is 1500 counts. Accordingly, the coordinates of the points B1 and D1 are (4500,0) and (-4500,2000) respectively.

To change the feedrate along the path, we break the segments AB and CD into two parts each and attach a vector speed to each segment by adding the symbol <n at the end of the segment instruction. This sets the attached speed to n counts/s. The resulting program is shown on the following page.

Instruction	**Interpretation**
#MOVE	Label
VMXY	Specify XY plane
VP 4500,0<20000	Move to Point B1 at speed 20000
VP 6000,0<10000	Move to Point B at speed 10000
CR 1000,270,180	Move to Point C - no speed change
VP -4500, 2000<20000	Move to Point D1 at speed 20000
VP -6000,2000<10000	Move to Point D at speed 10000
CR 1000,90,180	Move to Point E
VP 0,0<20000	Move to Point A at speed 20000
VE	End of path
VA 100000	Vector acceleration
VD 100000	Vector deceleration
BGS	Start motion
EN	End of program

DMC-1200	**DMC-1600**	**DMC-1700**
DMC-1800	**DMC-2000**	

Application Example: Routine for Monitoring Encoder Failure

An encoder failure or a broken encoder wire can cause the motor to run away. There are two main reasons for this:

1. A motor command has been given. In this case, the commanded position changes but the actual position remains constant since the encoder has failed. With a large position error, the controller will supply the motor amplifier with a large command signal. Use of the error limit, the Off-On-Error function, and the #POSERR automatic subroutine can be used to limit this problem.

2. The integrator gain term, KI, is non-zero. In this case, a very small position error will cause the controller to produce a motor command that increases to maximum until encoder counts are detected. Since the encoder has failed, the motor commanded will always increase to maximum and cause motor runaway. Use of the Integrator Limit, IL, can limit the maximum amount of signal that will be contributed by the Integrator Gain. (Note that use of IL can also effect the position accuracy during motion). Use of a monitor routine can also detect when an encoder has failed. The following example routine can be executed on an unused thread to monitor the status of the X axis encoder.

This routine disables the X axis motor when the controller is producing a command signal above the level required to move the motor and position movement is not detected. The user must determine the minimum signal required to move the motor.

```
#BROKEN                                      Encoder failure monitor routine
OE 1                                         Enable Off-On-Error for X axis
FRICX=.2                                     FRIC = min torque to move motor
#LOOP                                        Loop until torque above FRIC and
AT0                                          motion is not detected. Reset timer
JP#LOOP,(@ABS[_TTX]<FRICX)|(@ABS[_TVX]>0)    during loop
AT20                                         Wait 20 msec
JP#KILL,(@ABS[_TTX]>FRICX)&(@ABS[_TVX]=0)    If motion still not
```

	detected, kill motor
JP#LOOP	Otherwise, continue looping
#KILL	Kill routine
AB1	Abort current motion on X
MG "THE X AXIS MOTOR ENCODER HAS FAILED"	Display message
EN	

ELECTROMATE

Application Example: Routines for Monitoring Step Motor Operation

Galil controllers operate step motors as part of an open-loop system. However, encoder inputs and application memory are provided which allow position maintenance and stall detection for step motors. An encoder is placed on the motor or load and is read using the DE command. The following application program checks for three conditions; the first condition is known as “In Position”. This represents the motor and load reaching the commanded position or within the error band at the end of motion. The second condition is known as “ Position Maintenance”. Here the system is checked to make sure that if the motor is not asked to move, the load is not moving. The third condition is “Stall Detect”. This check makes sure that the system has not stopped in the process of moving to the final destination. Please note that these examples show only X-axis operation, however, they can be expanded to other axes setup for step motor.

Monitor Routine

#MONITOR	Monitor Program
DDBNDX=10	Define Deadband variable
PSRX=0	Define Position Error variable
NPSFLG=0	Define In Position flag
STLFLGX=0	Define Stall Detect Flag
#STPMNT	Routine for background execution
JS #INPSMNT, _BGX=0	If not moving, test In Position + Maintenance
JS #STLDTC,_BGX=1	If moving, test Stall Detect
JP#STPMNT	Loop

ELECTROMATE

Position Maintenance

#INPSMNT	Routine for In Position and Maintenance
POSERR=_DEX - _TPX	Calculate error between command and encoder
JP#ATPOS, @ABS [POSERR]<DDBNDX	In Position if error less than Deadband for X
NPSFLG=0	Clear In Position Flag when error
IP POSERR	Increment the position by the amount of error
EN	End of subroutine

In Position

#ATPOS	When In Position, Jump here
NPSFLG=1	Set the In Position Flag
EN	End of subroutine

Stall Detect

#STLDTC	Routine for Stall Detect
PSRX=_DEX-_TPX	Calculate error from pulses out vs. encoder
JP#STLX, @ABS[PSRX]>DDBNDX	If error larger than Deadband, jump to Stall
STLFLGX=0	Ed of subroutine
EN	Jump here when error Stall occurs
#STLX	Set the Stall Detect Flag for X axis
STLFLGX=1	End of subroutine
EN	

Appendix -- Optima Series DMC-1200, 13x8, 1600, 1700, 1800, 2000, 2100

SERVO MOTOR COMMANDS

AF	Analog feedback
DV	Dual loop operation
FA	Acceleration feedforward
FV	Velocity feedforward
IL	Integrator limit
KD	Derivative constant
KI	Integrator constant
KP	Proportional constant
NB	Notch Bandwidth
NF	Notch Frequency
OF	Offset
PL	Pole
SH	Servo here
TL	Torque limit
TM	Sample time
ZR	Zero

STEPPER MOTOR COMMANDS

KS	Stepper motor smoothing
MT	Motor type
RP	Report commanded position
TD	Step counts output
TP	Tell position of encoder
DE	Define encoder position
DP	Define reference position

BRUSHLESS MOTOR COMMANDS

BA	Brushless axis
BB	Brushless phase
BC	Brushless calibration
BD	Brushless degrees
BI	Brushless inputs
BM	Brushless modulo
BO	Brushless offset
BS	Brushless setup
BZ	Brushless zero

I/O COMMANDS

AL	Arm Latch
CB	Clear bit
CI	Communication interrupt
CO	Configure I/O points
EI	Enable interrupts
II	Input interrupt
OB	Define output bit
OC	Output compare function
OP	Output port
SB	Set bit
UI	User Interrupts

SYSTEM CONFIGURATION

AO	Analog output (DMC-2100)
BN	Burn parameters
BP	Burn program
BV	Burn variables and arrays
CC	Configure auxiliary port
CE	Configure encoder type
CN	Configure switches
CO	Configure I/O points
CW	Data adjustment bit
DE	Define dual encoder position
DP	Define position
DR	DMA/FIFO update rate
DV	Dual velocity (dual loop)
EI	Enable interrupts
EO	Echo off
IA	Set IP address (DMC-2100)
IH	Internet handle (DMC-2100)
IT	Independent smoothing
LZ	Leading zeros format
MO	Motor off
MT	Motor type
PF	Position format
QD	Download array
QU	Upload array
RS	Reset
VF	Variable format

MATH/SPECIAL FUNCTIONS

@SIN[X]	Sine of x
@COS[X]	Cosine of x
@COM[X]	1's compliment of x
@ASIN[X]	Arc sine of x
@ACOS[X]	Arc cosine of x
@ATAN[X]	Arc tangent of x
@ABS[X]	Absolute value of x
@FRAC[X]	Fraction portion of x
@INT[X]	Integer portion of x
@RND[X]	Round of x
@SQR[X]	Square root of x
@IN[X]	State of digital input x
@OUT[X]	State of digital output x
@AN[X]	Value of analog input x

INTERROGATION COMMANDS

LA	List arrays
LL	List labels
LS	List program
LV	List variables
MG	Message command
QR	Data record
QZ	Return DMA information
RP	Report command position
PL	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 program
TS	Tell switches
TT	Tell torque
TV	Tell velocity

PROGRAMMING COMMANDS

DA	Dealocate variables/arrays
DL	Download program
DM	Dimension arrays
ED	Edit program
ELSE	Conditional statement
ENDIF	End of cond. Statement
EN	End program
HX	Halt execution
IF	If statement
IN	Input variable
JP	Jump
JS	Jump to subroutine
NO	No-operation-for remarks
RA	Record array
RC	Record interval
RD	Record data
REM	Remark program
UI	User interrupt
UL	Upload program
ZS	Zero stack

ERROR CONTROL COMMANDS

BL	Backward software limit
ER	Error limit
FL	Forward software limit
OE	Off-on-error function
TL	Torque limit
YW	Timeout for in-position

INDEPENDENT MOTION COMMANDS

AB	Abort motion
AC	Acceleration
BG	Begin motion
DC	Deceleration
FE	Find edge
FI	Find index
HM	Home
IP	Increment position
IT	Smoothing time constant
JG	Jog mode
PA	Position absolute
PR	Position relative
SP	Speed
ST	Stop

TRIPPOINT COMMANDS

AD	After distance
AI	After input
AM	After motion profiler
AP	After absolute position
AR	After relative distance
AS	At speed
AT	After time
AV	After vector distance
MC	Motion complete
MF	After motion-forward
MR	After motion-reverse
WC	Wait for contour data
WT	Wait for time

CONTOUR MODE COMMANDS

CD	Contour data
CM	Contour mode
DT	Contour time interval
WC	Wait for contour data

ECAM/GEARING

EA	Ecam master
EB	Enable ECAM
EC	Ecam table index
EG	Ecam go
EM	ECAM cycle
EP	ECAM interval
EQ	Disengage ECAM
ET	Ecam table entry
GA	Master axis for gearing
GM	Gantry mode
GR	Gear ration for gearing

VECTOR/LINEAR INTERPOLATION

CA	Define vector plane
CR	Circular interpolation move
CS	Clear motion sequence
ES	Ellipse scaling
LE	Linear interpolation end
LI	Linear interpolation segment
LM	Linear interpolation mode
ST	Stop motion
TN	Tangent
VA	Vector acceleration
VD	Vector deceleration
VE	Vector sequence end
VM	Coordinated motion mode
VP	Vector position
VR	Vector speed ration
VS	Vector speed
VT	Smoothing time constant-vector

PC/104, COMPACT PCI, ISA BUS, VME & USB/ ETHERNET/ RS232/ RS422/ RS485 CONTROLLERS

Appendix -- Econo Series DMC-1410, 1411, 1412, 1414

MOTION

AB	Abort motion
AC	Acceleration
BG	Begin motion
CD	Contour data
CM	Contour mode
DC	Deceleration
DT	Contour time interval
EB	Enable cam mode
EG	Start cam motion
EM	Modulus for cam
EP	Master counts per table entry
EQ	Stop cam motion
ET	Cam table entry
FE	Find edge
FI	Find index
GR	Gear ratio
HM	Home
IP	Increment position
IT	Smoothing time constant-independent
JG	Job mode
KS	Stepper smoothing
PA	Position absolute
PR	Position relative
SP	Speed
ST	Stop

PROGRAM FLOW

AD	Wait for specified distance
AI	Wait for specified input
AM	Wait for motion complete
AP	Wait for absolute position
AR	Wait for relative distance
AS	Wait for "At Speed"
AT	Wait for elapsed time
EN	End program
HX	Halt task
IN	Input variable
II	Input interrupt
JP	Jump to program location
JS	Jump to subroutine
MG	Message
MC	Wait for "In Position"
MF	Forward motion past position
MR	Reverse motion past position
NO	No operation
RE	Return from error subroutine
RI	Return from interrupt
WC	Wait for contour data
WT	Wait for elapsed time
XQ	Execute program
ZS	Zero subroutine stack
TW	Timeout for "In Position"

CONFIGURATION

AL	Arm latch
BN	Save parameters in EEPROM
BP	Burn program (1412, 1414)
BV	Burn variables and array (1412, 1414)
CB	Clear output bit
CC	Configure 2nd RS232 port (1412, 1414)
CE	Configure encoder type
CN	Configure switches
DA	Deallocate arrays
DE	Define dual encoder position
DL	Download program
DM	Dimension arrays
DP	Define position
ED	Edit mode
EI	Enable ISA interrupts (1410, 1411)
EO	Echo off
LS	List program
MO	Motor off
MT	Motor type
OB	Define output bit
OP	Output port
PF	Position format
QD	Download array
QU	Upload array
RA	Record array
RC	Record
RD	Record data
RS	Reset
SA	Set address (1412, 1414)
SB	Set output bit
UI	User interrupt (1410, 1411)
UL	Upload program
VF	Variable format

CONTROL FILTER SETTINGS

DV	Damping for dual loop
FA	Acceleration feedforward
FV	Velocity feedforward
GN	Gain
IL	Integrator limit
KD	Derivative constant
KI	Integrator constant
KP	Proportional constant
OF	Offset
SH	Servo here
TL	Torque limit
TM	Sample time
ZR	Zero

STATUS

RP	Report command position
RL	Report latched position
SC	Stop code
TB	Tell status
TC	Tell error code
TD	Tell dual encoder position
TE	Tell position error
TI	Tell input
TP	Tell position
TR	Trace program
TS	Tell switches
TT	Tell torque
TV	Tell velocity

ERROR AND LIMITS

BL	Reverse software limit
ER	Position error limit
FL	Forward software limit
OE	Off on error

ARITHMETIC FUNCTIONS

@SIN	Sine
@COS	Cosine
@ABS	Absolute value
@FRAC	Fraction portion
@INT	Integer portion
@RND	Round
@SQR	Square root
@IN	Return digital input
+	Add
-	Subtract
*	Multiply
/	Divide
&	And
|	Or
( )	Parentheses

BRUSHLESS MOTOR COMMANDS

BA	Brushless axis
BB	Brushless phase
BC	Brushless calibration
BD	Brushless degrees
BI	Brushless inputs
BM	Brushless modulo
BO	Brushless offset
BS	Brushless setup

Appendix -- Ethernet Econo Series DMC-1415, 1416, 1425

SERVO MOTOR COMMANDS
AF Analog feedback
DV Dual loop operation (1415/1416)
FA Acceleration feedforward
FV Velocity feedforward
EL Integrator limit
KD Derivative constant
KI Integrator constant
KP Proportional constant
NB Notch bandwidth
NF Notch frequency
OF Offset
SH Servo here
TL Torque limit
TM Sample time
ZR Zero

STEPPER MOTOR COMMANDS
KS Stepper motor smoothing
MT Motor type
RP Report commanded position
TD Step counts output
TP Tell position of encoder
DE Define encoder position
DP Define reference position

BRUSHLESS MOTOR COMMANDS
BA Brushless axis
BB Brushless phase
BC Brushless calibration
BD Brushless degrees
BI Brushless inputs
BM Brushless modulo
BO Brushless offset
BS Brushless setup

I/O COMMANDS
AL Arm latch
CB Clear bit
CI Communication interrupt
CO Configure I/O points
EI Enable interrupts
II Input interrupt
OB Define output bit
OC Output compare function
OP Output port
SB Set bit

SYSTEM CONFIGURATION
BN Burn parameters
BP Burn program
BV Burn variables and arrays
CE Configure encoder type
CF Default port
CN Configure switches
CO Configure I/O points
CW Data adjustment bit
DE Define dual encoder position
DP Define position
DV Dual velocity (dual loop)
EO Echo off
IA Set IP address
IH Internet handle
IT Independent smoothing
LZ Leading zeros format
MB ModBus
MO Motor off
MT Motor type
PF Position format
QD Download array
QU Upload array
VF Variable format

MATH/SPEACIAL FUNCTIONS
@SIN[x] Sine of x
@COS[x] Cosine of x
@COM[x] 1's compliment of x
@ASIN[x] Arc sine of x
@ACOS[x] Arc cosine of x
@ATAN[x] Arc tangent of x
@ABS{x] Absolute value of x
@FRAC[x] Fraction portion of x
@INT[x] Integer portion of x
@RND[x] Round of x
@SQR[x] Square root of x
@IN[x] State of digital input x
@OUT[x] State of digital output x
@AN[x] Value of analog input x

INTERROGATION COMMANDS
LA List arrays
LL List labels
LS List program
LV List variables
MG Message command
QR Data record
QZ Return data record
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 program
TS Tell switches
TT Tell torque
TV Tell velocity

PROGRAMMING COMMANDS
DA Deallocate variables/arrays
DL Download program
DM Dimension arrays
ELSE Conditional statement
ENDIF End of cond. statement
EN End program
IF If statement
IN Input variable
JP Jump
JS Jump to subroutine
NO No-operation-for remarks
REM Remark program
UI User interrupt
UL Upload program
ZS Zero stack

ERROR CONTROL COMMANDS
BL Backward software limit
ER Error limit
FL Forward software limit
OE Off-on-error function
TL Torque limit
TW Timeout for in-position

TRIPPOINT COMMANDS
AD After distance
AI After input
AM After motion profiler
AP After absolute position
AR After relative distance
AS At speed
AT After time
AV After vector distance
MC Motion complete
MF After motion-forward
MR After motion-reverse
WC Wait for contour data
WT Wait for time

INDEPENDENT MOTION COMMANDS
AB Abort motion
AC Acceleration
BG Begin motion
DC Deceleration
FE Find edge
FI Find index
HM Home
IP Increment position
IT Smoothing time constant
JG Jog mode
PA Position absolute
PR Position relative
SP Speed
ST Stop

CONTOUR MODE COMMANDS
CD Contour data
CM Contour mode
DT Contour time interval
WC Wait for contour data

ECAM/GEARING
EA Ecam master
EB Enable ECAM
EC Ecam table index
EG Ecam go
EM ECAM cycle
EP Ecam interval
EQ Disengage ECAM
ET Ecam table entry
GA Master axis for gearing
GM Gantry mode
GR Gear ratio for gearing

VECTOR/LINEAR INTERPOLATION
(DMC-1425 Only)
CA Define vector plane
CR Circular interpolation move
CS Clear motion sequence
ES Ellipse scaling
LE Linear interpolation end
LI Linear interpolation segment
LM Linear interpolation mode
ST Stop motion
TN Tangent
VA Vector acceleration
VD Vector deceleration
VE Vector sequence end
VM Coordinated motion mode
VP Vector position
VR Vector speed ratio
VS Vector speed
VT Smoothing time constant-vector

Appendix -- Distributed Ethernet Series DMC-3425

SERVO MOTOR COMMANDS
AF Analog feedback
FA Acceleration feedforward
FV Velocity feedforward
IL Integrator limit
KD Derivative constant
KI Integrator constant
KP Proportional constant
NB Notch bandwidth
NF Notch frequency
OF Offset
SH Servo here
TL Torque limit
TM Sample time
ZR Zero

STEPPER MOTOR COMMANDS
KS Stepper motor smoothing
MT Motor type
RP Report commanded position
TD Step counts output
TP Tell position of encoder
DE Define encoder position
DP Define reference position

BRUSHLESS MOTOR COMMANDS
BA Brushless axis
BB Brushless phase
BC Brushless calibration
BD Brushless degrees
BI Brushless inputs
BM Brushless modulo
BO Brushless offset
BS Brushless setup

I/O COMMANDS
AL Arm latch
CB Clear bit
CI Communication interrupt
CO Configure I/O points
EI Enable interrupts
II Input interrupt
OB Define output bit
OC Output compare function
OP Output port
SB Set bit
UI User interrupts

SYSTEM CONFIGURATION
BN Burn parameters
BP Burn program
BV Burn variables and arrays
CC Configure auxiliary port
CE Configure encoder type
CN Configure switches
CO Configure I/O points
CW Data adjustment bit
DE Define dual encoder position
DP Define position
DR DMA/FIFO update rate
DV Dual velocity (dual loop)
EI Enable interrupts
EO Echo off
IA Set IP address
IH Internet handle
IT Independent smoothing
LZ Leading zeros format
MB ModBus
MO Motor off
MT Motor type
PF Position format
QD Download array
QU Upload array
VF Variable format

DISTRIBUTED CONTROL COMMANDS
CH Connect handle
LR Launch slave record
NA Specify # of axes
SA Send slave command
QW Slave record update rate

MATH/SPECIAL FUNCTIONS
@SIN[x] Sine of x
@COS[x] Cosine of x
@COM[x] 1's compliment of x
@ASIN[x] Arc sine of x
@ACOS[x] Arc cosine of x
@ATAN[x] Arc tangent of x
@ABS[x] Absolute value of x
@FRAC[x] Fraction portion of x
@INT[x] Integer portion of x
@RND[x] Round of x
@SQR[x] Square root of x
@IN[x] State of digital input x
@OUT[x] State of digital output x
@AN[x] Value of analog input x

INTERROGATION COMMANDS
LA List arrays
LL List labels
LS List program
LV List variables
MG Message command
QR Data record
QZ Return DMA information
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 program
TS Tell switches
TT Tell torque
TV Tell Velocity

PROGRAMMING COMMANDS
DA Deallocate variables/arrays
DL Download program
DM Dimension arrays
ELSE Conditional statement
ENDIF End of cond. statement
EN End program
IF If statement
IN Input variable
JP Jump
JS Jump to subroutine
NO No-operation-for remarks
REM Remark program
UI User interrupt
UL Upload program
ZS Zero stack

ERROR CONTROL COMMANDS
BL Backward software limit
ER Error limit
FL Forward software limit
OE Off-on-error function
TL Torque limit
TW Timeout for in-position

TRIPPOINT COMMANDS
AD After distance
AI After input
AM After motion profiler
AP After absolute position
AR After relative distance
AS At speed
AT After time
AV After vector distance
MC Motion complete
MF After motion-forward
MR After motion-reverse
WC Wait for contour data
WT Wait for time

INDEPENDENT MOTION COMMANDS
AB Abort motion
AC Acceleration
BG Begin motion
DC Deceleration
FE Find edge
FI Find index
HM Home
IP Increment position
IT Smoothing time constant
JG Jog mode
PA Position absolute
PR Position relative
SP Speed
ST Stop

CONTOUR MODE COMMANDS
CD Contour data
CM Contour mode
DT Contour time interval
WC Wait for contour data

ECAM/GEARING
EA Ecam master
EB Enable ECAM
EC Ecam table index
EG Ecam go
EM ECAM cycle
EP ECAM interval
EQ Disengage ECAM
ET Ecam table entry
GA Master axis for gearing
GM Gantry mode
GR Gear ratio for gearing

VECTOR/LINEAR INTERPOLATION
CR Circular interpolation move
CS Clear motion sequence
ES Ellipse scaling
LE Linear interpolation end
LI Linear interpolation segment
LM Linear interpolation mode
ST Stop motion
TN Tangent
VA Vector acceleration
VD Vector deceleration
VE Vector sequence end
VM Coordinated motion mode
VP Vector position
VR Vector speed ratio
VS Vector speed
VT Smoothing time constant-vector

Appendix -- Legacy Series DMC-1000, 1300, 1500

MOTION
AB Abort motion
AC Acceleration
BG Begin motion
CD Contour data
CM Contour mode
CR Circle
CS Clear motion sequence
DC Deceleration
DT Contour time interval
ES Ellipse scaling
FE Find edge
FI Find index
GA Master axis for gearing
GR Gear ratio
HM Home
IP Increment position
JG Jog mode
LE Linear interpolation end
LI Linear interpolation distance
LM Linear interpolation mode
PA Position absolute
PR Position relative
SP Speed
ST Stop
TN Tangent
VA Vector acceleration
VD Vector deceleration
VE Vector sequence end
VM Coordinated motion mode
VP Vector position
VS Vector speed

PROGRAM FLOW
AD After distance
AI After input
AM After motion complete
AP After absolute position
AR After relative distance
AS At speed
AT After time
AV After vector distance
EN End program
HX Halt task
IN Input variable
II Input interrupt
JP Jump to program location
JS Jump to subroutine
MG Message
NO No operation
RE Return from error subroutine
RI Return from interrupt
WC Wait for contour data
WT Wait
XQ Execute program
ZS Zero subroutine stack

GENERAL CONFIGURATION
AL Arm latch
BN Burn
CB Clear bit
CE Configure encoder type
CN Configure switches and stepper
DA Deallocate arrays
DE Define dual encoder position
DL Download
DM Dimension arrays
DP Define position
ED Edit mode
EI Enable interrupts
EO Echo off
LS List
MO Motor off
MT Motor type
OB Define output bit
OP Output port
PF Position format
RA Record array
RC Record
RD Record data
RS Reset
SB Set bit
UI User interrupt
UL Upload
VF Variable format

CONTROL FILTER SETTINGS
DV Damping for dual loop
FA Acceleration feedforward
FV Velocity feedforward
GN Gain
IL Integrator limit
IT Smoothing time constant - independent
KD Derivative constant
KI Integrator constant
KP Proportional constant
OF Offset
SH Servo here
TL Torque limit
TM Sample time
VT Smoothing time constant - vector
ZR Zero

STATU
RP Report command position
RL Report latch
SC Stop cod
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

ERROR AND LIMITS
BL Reverse software limit
ER Error limit
FL Forward software limit
OE Off on error

EDITOR
ED Edit mode
<return> Save line
<cntrl>P Previous line
<cntrl>I Insert line
<cntrl>D Delete line
<cntrl>Q Quit editor

ARITHMETIC FUNCTIONS
@SIN Sine
@COS Cosine
@ABS Absolute value
@FRAC Fraction portion
@INT Integer portion
@RND Round
@SQR Square root
@IN Return digital input
@AN Return analog input
+ Add
- Subtract
* Multiply
/ Divide
& And
| Or
() Parentheses

PC AT/XT BUS CONTROLLER

ELECTROMATE

Appendix -- Command Summary DMC-700 Series

MOTION

AB	Abort motion instantly
AC	Acceleration rate
BG	Begin motion
CD	Contour data
CM	Contour mode
CR	Circular segment
CS	Clear motion sequence
DE	Dual encoder position
DP	Define position
DT	Time increment for contour
FE	Find edge
FI	Find index
GA	Master axis for gearing
GR	Gear ratio
HM	Home
IP	Increment position
JG	Jog mode
LE	Specify linear end
LI	Linear interpolation distance
LM	Linear interpolation mode
MF	Frequency reference
MP	Master position
MS	Master/Slave mode
PA	Position absolute
PR	Position relative
PV	Proportional ratio
RM	Acceleration ramp
SP	Slew speed
ST	Stop motion
TA	S-curve profile
TF	Tell master frequency
TN	Tangent axis
TV	S-curve-vector move
VA	Vector acceleration
VM	Coordinated mode
VP	Vector position
VR	Accel ramp-vector move
VS	Vector speed
ZM	Zero master

CONTROL SETTINGS

DR	Sets DAC resolution
FA	Acceleration feedforward
GN	Gain
KI	Integrator
MO	Motor off
OF	Offset
PS	Position scale factor
RS	Reset controller
SH	Servo here
SS	Speed scale factor
SV	Servo
TL	Torque limit
TM	Sample time
VV	Vector scale factor
ZR	Filter zero

PROGRAM FLOW

AD	Distance trippoint
AI	Input trippoint
AM	Motion complete trippoint
AP	Absolute position trippoint
AR	Relative distance trippoint
AS	At speed trippoint
AV	Vector distance trippoint
JP	Conditional jump
JS	Conditional jump subroutine
WC	Wait for contour data
WT	Programmable delay

PROGRAMMING - GENERAL

AL	Arm latch
BN	Burn program into memory
DA	Deallocate array space
DL	Download program into memory
DM	Define array dimension
EN	End program
LS	List program
NO	No OP
RA	Automatic array capture
RC	Time interval for data capture
RD	Specify data for capture
RI	Return from interrupt subroutine
RL	Report latch
TR	Trace
UL	Upload program
XQ	Execute program
ZS	Zero subroutine stack
#n	Define program

COMMUNICATION & I/O

CB	Clear output bit
CC	Configure COM Port 2
CF	Message delay
CI	Communication Interrupt
EO	Echo off
II	Input interrupt
IL	Invert limit switch
IN	Input prompt
IO	Write to IO bus
MG	Message
OP	Write output port
PF	Position display format
SB	Set output bit
VF	Variable format

ERROR HANDLING & STATUS

ER	Define error limit
OE	Automatic error shut-off
RE	Return from error subroutine
SC	Stop status
TB	Tell status bits
TC	Error code
TE	Tell error
TI	Tell inputs
TP	Tell position
TS	Tell switches
TT	Tell torque

EDITOR

ED	Edit mode
<Return>	Save line
<cntrl>P	Previous line
<cntrl>I	Insert line
<cntrl>D	Delete line
<cntrl>Q	Quit editor

ARITHMETIC FUNCTIONS

@SIN	Sine
@COS	Cosine
@ABS	Absolute value
@FRAC	Fraction portion
@INT	Integer portion
@RND	Rounds
@SQR	Square root
@IO	I/O Bus data
@IN	Returns digital input
@AN	Returns analog input

STAND-ALONE MOTION CONTROLLER

Appendix -- Command Summary DMC-600 Series

MOTION
AB Abort motion instantly
AC Acceleration rate
BG Begin motion
CM Contour mode
CR Arc mode
CS Clear motion sequence
DP Define position
FE Find edge
HM Home
IP Increment position
JG Jog mode
MF Frequency reference
MP Master position
MS Master/Slave mode
PA Position absolute
PR Position relative
SP Slew speed
ST Stop motion
TF Tell master frequency
VA Vector acceleration
VP Vector position
VS Vector speed
ZM Zero master

CONTROL SETTINGS
DB Deadband
FA Acceleration feedforward
GN Gain
KI Integrator
MO Motor off
OF Offset
PL Pole
RS Reset controller
SH Servo here
SV Servo
TL Torque limit
TM Sample time
ZR Filter zero

PROGRAM FLOW
AD Distance trippoint
AI Input trippoint
AM Motion complete trippoint
AP Absolute position trippoint
AS At speed trippoint
JP Conditional jump
JS Conditional jump subroutine
WT Programmable timer

PROGRAMMING - GENERAL
DL Download program into memory
EN End program
LS List program
NO No OP
RI Return from interrupt subroutine
TR Trace
UL Upload program
XQ Execute program
ZS Zero subroutine stack
#n Define program

COMMUNICATION & I/O
CB Clear output bit
DC Decimal mode
HX Hex mode
II Input interrupt
IN Input prompt
MG Message
OP Write output port
SB Set output bit

ERROR HANDLING & STATUS
ER Define error limit
OE Automatic error shut-off
RE Return from error subroutine
SC Stop status
TC Error code
TE Tell error
TI Tell inputs
TP Tell position
TS Tell switches
TT Tell torque

EDITOR
ED Edit mode
<return> Save line
<cntrl>P Previous line
<cntrl>I Insert line
<cntrl>D Delete line
<cntrl>Q Quit editor

PC AT/XT BUS CONTROLLER

Appendix -- Command Summary DMC-400 Series

CONTROL SETTINGS

DB	±127	Deadband compensation#
ER	0-1023	Following error tolerance#
GN	1-255	Filter gain#
KI	0-127	Filter integration#
OF	±127	Offset#
PL	0-255	Filter pole (damping)#
TL	0-127	Torque limit#
TM	500-65000	Control loop sample time#
ZR	0-255	Filter zero (damping)#

MOTION

FE		Find edge - homing
IM		Incremental positioning-continuous path
MO		Turn servos off
PA	$\pm 8x10^6$	Absolute positioning-profiled
PR	$\pm 8x10^6$	Relative positioning-profiled
RP	0-255	Repetitive cycling-same direction
RR	0-255	Repetitive cycling-alternate direction
SH		Turn servos on at current position
SN	$\pm 8x10^6$	Index from run#
SV		Turn servos on
VM	4-250000	Jogging mode-profiled#

MOTION PARAMETERS OE

AB		Abort motion-instantaneous#
AC	$0\text{-}1.3x10^x$	Acceleration of velocity profile
BG		Begin motion
ES	0 or 1	Specifies end motion on switch if 1
IP	$\pm 8x10^6$	New position while motor in motion#
SP	0-250000	Slew speed of velocity profile#
SS	0 or 1	Specifies start motion on switch if 1
		Stop motion-decelerate#
WT	0-32000	Wait time between cycles#

STATUS

TC	Tell motion status#
TD	Tell position continuously#
TE	Tell position error#
TI	Tell inputs & controller status#
Bit	
7	Executing sequence*
6	Executing move*
5	FWD limit switch*
4	REV limit switch*
3	Remote/local*
2	Stop/start*
1	Direction input
0	Excessive position error
TP	Tell position#
TS	Tell latched position#
TT	Tell motor torque#
TV	Tell velocity#

OTHER

DC	Input parameters in decimal#	
DH	$\pm 8x10^6$	Define home position
DS	0 or 1	Define direction of motion by switch if 1
HX		Input parameters in Hex#
LT	0 or 1	Latch position on input if 1#
0 or 1		Turn motor off-on-error if 1#
RD	0 or 1	Report ASCII "H" when motion complete if 1
RS		Resets controller
SM	0 or 1	Specifies sign/magnitude output if 1

DEFAULT PARAMETERS
The system wakes up in a position-control servo mode unless MOF is jumpered. The default values are: GN 8, ZR 232, PL 0, KI 0, ST SP 32768 and AC 65536.

PC AT/XT BUS CONTROLLER