

ACCU ELECTRIC MOTORS INC

USA: (888) 932-9183

CANADA: (905) 829-2505

- ✓ Over 100 years cumulative experience
- ✓ 24 hour rush turnaround / technical support service
- ✓ Established in 1993



The leading independent repairer of servo motors and drives in North America.

Visit us on the web:

www.servo-repair.com

www.servorepair.ca

www.ferrocontrol.com
www.sandvikrepair.com
www.accuelectric.com

Scroll down to view your document!

For 24/7 repair services :

USA: 1 (888) 932 - 9183

Canada: 1 (905) 829 -2505

Emergency After hours: 1 (416) 624 0386

Servicing USA and Canada

Application Design

The information in this chapter will enable you to:

- ❑ Recognize and understand important considerations that must be addressed before you implement your application
- ❑ Understand the capabilities of the system
- ❑ Customize the system to meet your requirements
- ❑ Use sample applications to help you develop your application

Defining Moves

The following section describes things you should consider when creating move profiles with the Compumotor Plus.

Application Considerations

This section describes some differences between theoretical and real-world performance in defining moves.

Position
Accuracy
Versus
Repeatability

In positioning systems, some applications require high absolute accuracy. Others require repeatability. You should clearly define and distinguish these two concepts when you address the issue of system performance.

For many systems, the term accuracy is used when repeatability is required. When the motor always moves to the same distance from the same position, the primary positioning goal is not accuracy, but repeatability. Repeatability measures how accurately you can repeat moves to the same position. For example, a bottle labeling machine must rotate one revolution each time a label is applied. Since the motor always ends up in the same position and is always moving the same direction, repeatability is the dominant accuracy factor.

Accuracy on the other hand, is the error in finding a random position. For example, suppose the job is to measure the size of an object. The size of the object is determined by moving the positioning system to a point on the object and using the move distance required to get there as the

measurement value. In this situation, basic system accuracy is important. The system accuracy must be better than the tolerance on the measurement that is desired.

Consult the technical data section of *The Compumotor Catalog* for more information on accuracy and repeatability.

Calculating Move Times

You can calculate the time it takes to complete a move by using the acceleration, velocity, and distance values that you define. However, you should not assume that this value is the actual move time.

There is calculation delay and motor settling time that make your move longer. After you issue the Go (G) command the indexer can take up to 100ms to calculate the move before the motor starts moving. You should also allow some time for the motor to settle into position. There is normally a delay of less than 30ms.

$$T_{Total} = T_{Calculation} + T_{Move} + T_{Settling}$$

You can virtually eliminate the calculation delay by using predefined moves (GDEF). This feature is available in Z3 and higher software revisions.

Positioning Modes

Incremental vs. Absolute Positioning

A preset move is a move with a distance that you specify (in motor steps). You can select preset moves by putting the Compumotor Plus into normal mode using the Mode Normal (MN) command. Preset moves allow you to position the motor in relation to the motor's previous stopped position (incremental moves) or in relation to a defined zero reference position (absolute moves). You can select incremental moves by using the Mode Position Incremental (MPI) command. You can select absolute moves using the Mode Position Absolute (MPA) command.

Incremental Moves (Preset Mode)

When you are in the Incremental mode (MPI), a preset move rotates the motor the specified distance from its starting position. For example, to move the Compumotor Plus motor 2 revolutions. You must specify a preset move with a distance of +10,000 steps, assuming a resolution of 5,000 steps per revolution. Every time the indexer executes this move, the motor moves 2 revolutions from its resting position. You can specify the direction of the move in one command. You specify the direction by using the optional sign (D+10,000 or D-10,000), or you can define it separately with the Set Direction (H) command (H+ or H-).

Example

Command	Description
> MPI	Set to Incremental Position mode
> A2	Set acceleration to 2 rps ²
> V5	Set velocity to 5 rps
> D10000	Set distance to 10,000 steps
> G	Execute the move (Go)
> G	Repeat the move (Go)
> H	Reverse direction of next move
> G	Execute the move (Go)

The motor moves 2 revolutions and stops. It then moves another 2 revolutions in the same direction and stops. The motor changes direction and moves 2 revolutions.

Absolute Moves (Preset Mode)

A preset move in the Absolute mode (MPA) moves the motor the distance that you specify (in motor steps) from the absolute zero position. You can set the absolute position to zero with the Position Zero (PZ) command, the Reset (Z) command, or by cycling the power to the drive. The absolute zero position is initially the power-up position.

The direction of an absolute preset move depends upon the motor position at the beginning of the move and the position you command it to move to. For example, if the motor is at absolute position +12,800, and you instruct the motor to move to position +5,000, the motor will move in the negative direction a distance of 7,800 steps to reach the absolute position of +5,000.

The Compumotor Plus powers up in Incremental mode. When you issue the Mode Position Absolute (MPA) command, it sets the mode to absolute. When you issue the Mode Position incremental (MPI) command the unit switches to Incremental mode. The Compumotor Plus Drive retains the absolute position, even while the unit is in the Incremental mode. You can use the Position Report (PR) command to read the absolute position.

Example

Command	Description
> MPA	Set to Absolute Position mode
> A2	Set acceleration to 2 rps
> V10	Set velocity to 10 rps
> PZ	Set the current position to as home
> D5000	Set distance to 5,000 steps
> G	Execute the move (Go)
> D10000	Set distance to 10,000 steps
> G	Move the motor to absolute position 10,000 (Go)
> D0	Set the move distance to 0
> G	Execute the move (Go)
> MPI	Set indexer to Incremental Position mode

The motor moves 1 revolution and stops. It then moves another revolution and stops. The motor then moves in the opposite direction two revolutions.

Programming Highlights

This section contains information you will need when programming the Compumotor Plus.

Interactive Programming

You can operate the Compumotor Plus RS-232C interface in two modes based on the needs of your application. The two modes are *interactive* and *non-interactive*. In the interactive mode the Compumotor Plus returns a prompt (>) when it is ready for another command. Use the Enable Interactive Mode (SSI0) command to make the Compumotor Plus interactive. You use the non-interactive mode when controlling the Compumotor Plus from a pre-programmed computer (i.e., not from terminal mode). You do this because it is easier for the computer to understand the Compumotor Plus without the prompt (>) character coming back after each command.

When you enable the interactive mode, the device address must be set to 1. The indexer responds with a > or a ? when it receives a command. It responds with a > when the command has been successfully processed and a question mark (?) when it does not receive a valid command. If you enter a valid command, but enter an invalid range (e.g., V70), the Compumotor Plus responds with a ?. These interactive responses are preceded with a carriage return and a line feed. You must enter and define entire loops and sequences before the system provides an interactive response.

Programmable Delays

You can use the Time (T) command to halt the operation of the indexer function for a preset time. If the Compumotor Plus is in Continuous mode, you may use the Time (T) command to run the motor at continuous velocity for a set time, then change to a different velocity.

In the Preset mode, the motor finishes the move before the indexer executes the time delay.

Example

Command	Description
> PS	Wait for the controller to receive a Continue (C) command before executing the next command
> G	Move motor 25,000 steps
> T5	Wait 5 seconds after the move ends
> H	Change motor direction
> G	Move motor 25,000 steps in the opposite direction
> C	Continue execution

Program Branching

This section discusses methods of changing the path of program execution through a conditional statement like IF, or through a fixed branch such as a loop or goto.

Looping

This section discusses methods of establishing loops in the program you write for your application. Loops are implemented with the Loop (L) command and the End Loop (N) command. Loops can be created individually or nested up to 16 levels deep.

The Loop (L) command repeats only buffered commands. Buffered commands are queued up and executed in order, so you can enter sequences of commands which are then executed one after the other. Place the commands to be executed between the L command and the N command. They are repeated the number of times indicated in the L command, for example, L3 causes three loops to be executed.

You can use the Immediate Pause (U) command to pause execution of the loop while it is in progress. *The U command does not work in Continuous mode.* To resume loop execution, issue the Continue (C) command.

The example below shows a sample loop. In this example, the motor makes 2 moves with a half second delay between the moves.

Example

Command	Description
> L2	Loop twice
> G	Execute the move (Go)
> T.5	Wait 0.5 seconds
> N	End the loop

Nesting Loops

The example below shows how you can nest one loop inside another loop.

Example

Command	Description
> L	Loop indefinitely
> CR	Send a carriage return
> L2	Loop twice
> G	Execute the move (Go)
> T.5	Wait 0.5 seconds
> N	End the loop
> N	End the loop

Branching with IF

You can perform conditional branching with the Compare Error Flag (**IFER**), Compare User Flag (**IFFL**), and Compare Trigger Status (**IFTR**) commands. All three of these commands are very similar to **IF_THEN** statement in *Basic programming*. If the condition evaluates true, the Compumotor Plus performs the commands that immediately follow the **IF** command. If the condition evaluates false, the Compumotor Plus skips all of the commands that follow the **IF** command until it reaches the End of **IF** Statement (**NIF**) command.

IF An Error Occurs

The **IFER** command checks to see if there is an error (such as slip fault). If there is an error, the commands immediately following the **IFER** are executed, otherwise the commands immediately following the End of **IF** statement are executed. For a detailed description of this command, refer to *Chapter 5, Software Reference*.

Example

Command	Description
> XE10	Erase Sequence #10
> XD10	Define Sequence #10
> IFER	If hardware error or limit encountered, then execute the following
> "SYSTEM_	Write to RS-232C port to inform user of system status
> "ERROR_	
> "OR_LIMIT_	
> "ENCOUNTERED_	
> ST1	Turns amplifier off
> ELSE	Otherwise
> "SYSTEM_	Writes to RS-232C Port
> "READY_	
> NIF	Ends IF statement
> XT	Ends sequence definition

IF User Flags

The Compare User Flag (**IFFL**), command compares the pattern set by the User Flag (**SFL**) command. If the patterns match the commands following the **IFFL** command are executed. If the patterns do not match the program continues after the next **NIF** command (End If).

This command is useful if you wish to make a decision based on previous program events that set or clear the user flag bits. For example, in an application with several sequences (or programs), at the end of each sequence, you can assign different bit patterns with the **SFL** command. If you select these sequences from the host computer, you may wish to make different moves depending on the sequence you ran. For a detailed description of this command, refer to *Chapter 5, Software Reference*.

Example

Command	Description
> PS	Wait for the controller to receive a Continue (C) command before executing the next command
> SFL1010	Set user flag bits 7 and 5 and clears bits 6 and 4, the remaining bits are not altered
> IFFL1010	If user flag bits 5 and 7 are set, and bits 6 and 4 are clear, perform the following commands
> A10	Set acceleration to 10 rps ²
> V5	Set velocity to 5 rps
> D25000	Set distance to 25,000 steps
> G	Execute the move (Go)
> NIF	End IF statement
> C	Continue execution

The **IFFL** pattern matches the **SFL** setting and the motor moves 25,000 steps.

IF a Trigger Input

The **IFTR** command compares its pattern with the state of the programmable inputs defined as trigger inputs. If the patterns match the following commands are executed. If the patterns do not match, the commands following the next **NIF** command are executed. This command is useful for branching and performing conditional moves using the programmable inputs. For a detailed description of this command, refer to *Chapter 5, Software Reference*.

Example

Command	Description
> IFTRXXX10X	If sequence #1 is active and sequence #2 is not active, execute the following commands:
> A10	Set acceleration to 10 rps ²
> V5	Set velocity to 5 rps
> D25000	Set distance to 25,000 steps
> G	Execute the move (Go)
> NIF	End IF statement
> IFTRXXX01X	If sequence #1 is not active (open) and sequence #2 is active (closed), issue the following commands:
> A10	Set acceleration to 10 rps ²
> V5	Set velocity to 5 rps
> D5000	Set distance to 5,000 steps in the opposite direction
> G	Execute the move (Go)
> NIF	End IF statement
> IFTR1XX	If Trigger #1 is active, do the following command.
> 1"DONE	End message DONE
> NIF	End IF statement

Subroutines

When you use the Goto Sequence (**XG**) and the Execute a Sequence (**XR**) command sequences, you can execute different sequences from within a sequence.

These commands are similar to **GOTO** and **GOSUB** commands in *Basic* programming. If you use an **XG** command, the program will call or go to the sequence that you specified in the **XG** command. After executing the specified sequence, the system will not return to the original sequence. You could cause it to return to the original sequence by issuing another **XG** command, but there is a better way, the Run Sequence (**XR**) command. You use the **XR** command when you want to return to the calling (original) sequence upon completion of the called sequence when a Terminate Sequence (**XT**) command is encountered.

You can make as many as 16 calls with the **XR** command before returning to the calling sequence. There is no limit to the number of times you can use the **XG** command since the program need not return control to the original sequence.

Example

Command	Description
> XE2	Erase sequence #2
> XD2	Define sequence #2
> A10	Set acceleration to 10 rps ²
> V5	Set velocity to 5 rps
> D2000	Set distance to 2,000 steps
> G	Execute the move (Go)
> XT	End sequence #2 definition
> XE3	Erase sequence #3
> XD3	Define sequence #3
> A10	Set acceleration to 10 rps ²
> V5	Set velocity to 5 rps

```

> D-2000      Set distance to 2,000 steps (CCW)
> G           Execute the move (Go)
> XT         End sequence #3 definition
> XE1        Erase sequence #1
> XD1        Define sequence #1
> XR2        Execute sequence #2
> XR3        Execute sequence #3
> XT         End sequence #1 definition
> XR1        Execute sequence #1

```

In the previous example, when you execute sequence #1, the program moves to sequence #2. After executing sequence #2, the program returns to sequence #1. The program then moves to execute sequence #3.

Example

Command	Description
> XE1	Erase sequence #1
> XD1	Define sequence #1
> IFTR1	If TRIG 1 is active (Closed)
> XG2	Execute sequence #2
> NIF	End IF statement
> A1	Set acceleration to 1 rps ²
> V5	Set velocity to 5 rps
> D25000	Set distance to 25,000 steps
> G	Execute move.
> XT	End sequence #1 definition
> XR1	Execute sequence #1

In the previous example, when you execute Sequence 1, the program checks the input pattern. If TRIG1 is on, the program moves to execute Sequence 2. After executing sequence 2, program does not return to Sequence 1. If TRIG1 is off, the program ignores the XG2 command and makes the 25,000-step move.

Input/Output

The following section describes operation of the programmable inputs and outputs, and the move completion signals.

Programmable Outputs (POBs)

You can turn the programmable outputs (OUT 1 - OUT 2) on and off with the O command. Outputs OUT 1 and OUT 2 are factory set as programmable outputs. However, you can configure the outputs to perform different functions with the Output Mode (OM) command. Refer to the OM command in *Chapter 5, Software Reference* for descriptions of the available functions. You can use these outputs to turn on and off other devices (i.e., lights, switches, etc.).

Example

Command	Description
> PS	Pause command execution until the controller receives a C
> A10	Set acceleration to 10 rps ²
> V5	Set velocity to 5 rps
> D25000	Set distance to 25,000 steps
> OM1	Set OUT1 and OUT2 as programmable outputs
> O10	Turns OUT1 on and OUT2 off
> G	Execute the move
> O01	Turn OUT1 off and OUT1 on
> C	Continue execution

This example above defines **OUT1** and **OUT2** as programmable outputs. **OUT1** turns on and **OUT2** is turned off, the motor moves 25,000 steps. When the motor stops and **OUT1** turns off and **OUT2** turns on.

Event Completion Signals

You can program the Compumotor Plus to notify you when a move or other event is complete. *Note: You may only signal the completion of buffered events.*

- LF Line feed
- CR Carriage return
- O Output command
- " Quote command (literal string)

Example

	Command	Description
	> A2	Set acceleration to 2 rps ²
	> v.5	Set velocity to 0.5 rps
	> D12500	Set distance to 12,500 steps
	> G	Execute the move (Go)
<i>Line feed</i> ⇨	> 1LF	Send a line feed over the RS-232C interface

The motor moves 12,500 steps. When the move is complete, the Compumotor Plus sends a line feed to the host over the RS-232C interface.

Example

	Command	Description
	> A2	Set acceleration to 2 rps ²
	> v.5	Set velocity to 0.5 rps
	> D12500	Set distance to 12,500 steps
	> G	Execute the move (Go)
<i>Carriage return</i> ⇨	> 1CR	Send a carriage return

The motor moves 12,500 steps. When the move is complete, the Compumotor Plus sends a carriage to the host over the RS-232C interface.

Example

	Command	Description
	> A2	Set acceleration to 2 rps ²
	> v.5	Set velocity to 0.5 rps
	> D12500	Set distance to 12,500 steps
	> G	Execute the move (Go)
<i>Output</i> ⇨	> O1	Turn on Output #1

The motor moves 12,500 steps. When the move is complete, Output 1 is turned on.

Example

	Command	Description
	> A2	Set acceleration to 2 rps ²
	> v.5	Set velocity to 0.5 rps
	> D12500	Set distance to 12,500 steps
	> G	Execute the move (Go)
<i>Message</i> ⇨	> 1"DONE	Set the DONE message

The motor moves 12,500 steps. When the Compumotor Plus completes the move, the unit issues the **DONE** message from the Compumotor Plus to the host over the RS-232C interface.

Remote Jogging

In some applications, you may want to adjust the motor position by toggling a switch on and off. The motor moves when the switch is on and stops when the switch is off. This is called jogging the motor. You can configure the Compumotor Plus for jogging operation by doing the following.

- Define programmable inputs as jog inputs using the Input Mode (IM) command
- Define the jogging velocity with the Jog Velocity (JV) command
- Enable jogging with the OSE1 command
- Attach a switch to the jog inputs. Use a two pole switch for jogging both directions
- Turn the switch on to jog the motor

The following example shows how you can define power-up sequence #40 to set up jogging.

Step ①

Define a power up sequence:

Command	Description
> XE40	Erase sequence #40
> XD40	Define sequence #40
JA2	Set jog acceleration to 2 rps ²
OSE1	Enables jog function
JV5	Set jog velocity to 5 rps
IM3	Define TRIG 2 and TRIG 3 as jog CW and CCW lines
XT	End sequence definition
> SV	Saves sequence definitions into EEPROM
> Z	Resets the Compumotor Plus controller
>	

Step ②

Turn on TRIG 2 input to move the motor in the CW direction at 5 rps (until you turn off TRIG 2).

Step ③

Turn on TRIG 3 input to move the motor in the CCW direction at 5 rps (until you turn off TRIG3).

Defining and Using Programs (Sequences)

Use the following commands to define, erase, and run programs. *In the Compumotor Plus language programs are referred to as sequences of commands or sequences.* Refer to Chapter 5, *Software Reference*, for detailed descriptions and syntax of the following commands.

Pertinent Commands

Command	Description
> XD	Start sequence definition
> XE	Delete sequence EEPROM
> XQ	Set/reset interrupted Run mode
> XRP	Run sequence with a pause
> XT	End sequence definition
> XU	Upload sequence
> XR	Run sequence
> SBJ1	Run a sequence defined by BCD sequence inputs
> XG	Exit current sequence and move to execute another sequence

A sequence is a series of commands. The commands are executed in order whenever the sequence is run. Only buffered commands may be used in a sequence. Immediate commands cannot be stored in a sequence, just as they cannot be stored in the command buffer.

The Compumotor Plus has 1,500 bytes of non-volatile memory to store 40 sequences. The sequences may have different lengths, so you may have one

long sequence or several short ones, as long as the total length does not exceed 1,500 characters.

To define a sequence do the following:

- Enter the Define Sequence (**XD**) command immediately followed by sequence identifier number (1 to 40) and a delimiter. For example, **XD1**.
- Enter the commands you want to execute when the sequence is run.
- Enter the Terminate Sequence (**XT**) command to end the sequence definition.
- Enter the **SAVE** command to save the sequence.

All commands that you enter after the **XD** command and before the **XT** command are executed when the sequence is run. An example is provided below.

Example

Command	Description
> XE1	Erase sequence #1
> XD1	Begin definition of sequence #1
A2	Set acceleration to 2 rps ²
V10	Set velocity to 10 rps
D5000	Set distance to 5000 steps
G	Execute the move (Go)
H	Reverse direction
G	Execute the move (Go)
XT	End definition of sequence
> XR1	Runs sequence #1

Sequence #40 is reserved for power up execution.

You can run a sequence by entering the **XR** command and sequence identifier number (in the range of 1 to 39) and a delimiter.

You can also run the sequences by specifying the sequence number in BCD on the programmable inputs. To accomplish this do the following.

- Define some programmable inputs as sequence-select inputs using the Input Mode (**IM**) command
- Enable the Continuous Sequence scan mode (**SSJ1**), you can also execute a sequence by turning on Sequence inputs to indicate which sequence you wish to run

Once you define a sequence, it cannot be redefined until you delete it. You can delete a sequence by entering the **XE** command immediately followed by a sequence identifier (1 to 40) and a delimiter. You may then redefine that sequence.

Sequences that you define are not saved into the non-volatile memory, until a Save (**SAVE** or **SV**) command is issued.

Sequence Selection Methods

After you define the sequences from the RS-232C interface, you can execute the sequences by using one of the following methods.

- Standalone Use thumbwheel switches to select and run the sequence
- Computer Interface Use the Execute Sequence (**XR**) command to run the sequences
- PLC Use the sequence select inputs to run a sequence

Standalone Operation

This section explains and provides examples of how to store programs and run them with remote switches, and run them automatically when you power up the system. First, you will need to enter the programs into the Compumotor Plus. You will need a terminal or a computer with RS-232C communication capabilities for programming the Compumotor Plus controller.

Power-Up Sequence execution

Sequence #40 is always run on power up and after a Reset (z) command. To run another sequence on power up, put an XR or XG at the end of sequence #40. If sequence #40 is empty, no sequence is run on power up. Refer to *Chapter 5, Software Reference*, for detailed descriptions and syntax of the following commands.

Example

Command	Description
> XE40	Erase sequence #40
> XD40	Begin definition of sequence #40
LD3	Disable limits if they are not connected
A2	Set acceleration to 2 rps ²
V5	Set velocity to 5 rps
D12500	Set distance to 12,500 steps
G	Execute the move (Go)
XT	End sequence definition
> Z	Reset the controller and runs sequence #40

A power-up sequence is typically used to store set-up or initialization parameters that your application requires. Some of these commands are listed below.

- SSJ1 Continuous Sequence Scan mode
- SN Scan time
- JA Jog acceleration
- JVL Jog velocity low
- JVH Jog velocity high

You can put any buffered commands into Sequence #40 to have them executed during power-up. Immediate commands cannot be put into sequences.

Remote Sequence Execution

You can execute sequences remotely using a switch selection by doing the following.

Step ①

Example

Define a power up sequence

Command	Description
> XE40	Erase sequence #40
> XD40	Define sequence #40
JA2	Set jog acceleration to 2 rps ²
OSE1	Enables jog function
JV5	Set jog velocity to 5 rps
IM3	Define TRIG 2 and TRIG 3 as jog CW and CCW lines
XT	End sequence definition
>	

Step ②

Define all sequences that your application may require. The following example defines sequence #1.

Command	Description
> XE1	Erase sequence #1
> XD1	Define sequence #1
A1	Set acceleration to 1 rps ²
V2	Set velocity to 2 rps
D1000	Set distance to 1,000 steps CW
G	Execute the move (Go)
XT	End sequence definition
>	

The following example defines sequence #2.

> XE2	Erase sequence #2
> XD2	Define sequence #2
A1	Set acceleration to 1 rps ²
V2	Set velocity to 2 rps
D-1000	Set distance to 1,000 steps CCW
G	Execute the move (Go)
XT	End sequence definition
>	

Step ③

Enter the XR1 command to move the motor 1,000 steps CW.

Step ④

Enter the XR2 command to move the motor 1,000 steps CCW.

PLC Operation

You can use a PLC to execute 39 of the 40 sequences that are stored in the Compumotor Plus controller (the 40th is reserved for power-up execution). You can configure up to eight inputs as sequence-select inputs. You can accomplish this by changing the BCD values of the PLC outputs

Scanning for Sequence Execution

Changing the BCD values of sequence input lines results in a new sequence being run that corresponds to the new value. The sum of the values for each input determines which sequence the indexer will run. For example, if you set up all six inputs as sequence-select inputs, the lowest input (SEQ 3) will have the least significant value and the highest input (TRIG 1) will have the most significant value. Refer to the table below.

Do the following to set the Compumotor Plus inputs as sequence select lines.

- Enter the Input Mode (IM) command to set up the inputs as sequence-select inputs.
- Enter the SSJ1 command to configure the Compumotor Plus controller to read sequences via BCD input.
- Optionally enter the XQ1 command to configure the Compumotor Plus controller to use the Interrupted Run mode. See below for a detailed explanation.
- Optionally, enter the Scan (SN) command to tell the Compumotor Plus how long the inputs must be stable before accepting the input as valid. See below, or *Software Reference*, for a detailed explanation.

Once you issue the SSJ1 command, the Compumotor Plus controller scans the sequence select inputs to find the sequence that you have specified for execution. If it finds a valid sequence number on the inputs (any number other than zero) it runs the sequence. The Compumotor Plus completes the execution of the desired sequence and the process begins again.

The Interrupted Run Mode (XQ1) command makes the Compumotor Plus wait until all of the sequence inputs are turned off (sequence zero) before selecting the next sequence to execute.

Sample PLC Applications and Commands

The Scan (SN) command determines how long the sequence-select input must be maintained before the controller executes the program. This is also called the debounce time.

This section provides step-by-step procedures to run sequences from your PLC. First, you need to enter the programs into the Compumotor Plus. You will need a terminal or a computer with RS-232C communication capability. You need to define the sequences before you can execute them with your PLC's BCD outputs. The Compumotor Plus controller saves these sequences with the Save (SV) command.

Step ①

Define a power-up sequence

Command	Description
> XE40	Erase sequence #40
> XD40	Define sequence #40
> SSJ1	Execute sequences via PLC input
> SN20	Set scan time to 20 ms
> XQ1	Set to Interrupted Run mode
> A10	Set acceleration to 10 rps ²
> V2	Set velocity to 2 rps
> IM3	Define all six TRIG/SEQ inputs as sequence-select lines
> LD3	Disable the limits (if they are not connected)
> XT	End the sequence definition

Every time you power up the Compumotor Plus controller, it executes these commands and enables the Compumotor Plus to read up to 39 sequences from the sequence select inputs.

Step ②

Define any sequences that your application may require.

Command	Description
> XE1	Erases sequence #1
> XD1	Defines sequence #1
> D2000	Set distance to 2,000 steps (CW)
> G	Execute the move (Go)
> XT	End sequence #1 definition
> XE2	Erases sequence #2
> XD2	Defines sequence #2
> D4000	Set distance to 4,000 steps (CW)
> G	Execute the move (Go)
> XT	End sequence #2 definition
> XE3	Erase sequence #3
> XD3	Define sequence #3
> D8000	Set distance to 8,000 steps (CW)
> G	Execute the move (Go)
> XT	End sequence #1 definition
> XE39	Erase sequence #39
> XD39	Define sequence #39
> D-14000	Set distance to -14,000 steps (CCW)
> G	Execute the move (Go)
> XT	End sequence #39 definition

Step ③

Verify that your programs were stored properly by uploading each entered sequence (XU). If you receive responses that differ from what you programmed, re-enter those sequences.

Step ④

Save the sequences entered by typing the Save (SV) command

Step ⑤ Run each program from the RS-232C interface with the Run Sequence (XR) command. Make sure that the motor moves the distance that you specify.

Step ⑥ Assuming your PLC accepts open collector outputs connect the inputs and outputs as shown in the following table. If not, you will need to add pull up resistors to the outputs.

PLC	Compumotor Plus
Output 6	Trig 6
Output 5	Trig 5
Output 4	Trig 4
Output 3	Trig 3
Output 2	Trig 2
Output 1	Trig 1
Ground	I/O Ground

Step ⑦ Refer to the user guide that accompanied your PLC unit to turn on the proper combination of outputs to execute one of the four sequences programmed and stored in the Compumotor Plus controller.

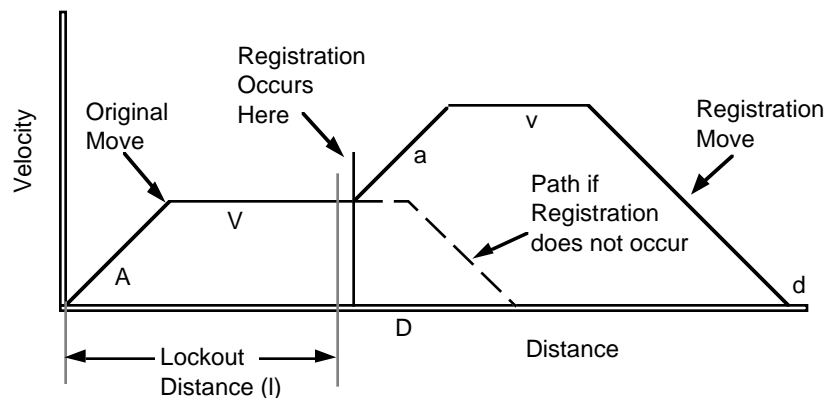
- Turning on only the PLC's Output 1 will execute sequence #1
- Turning on only the PLC's Output 2 will execute sequence #2
- Turning on only the PLC's Outputs 1 and 2 will execute sequence #3
- Turning on only the PLC's Outputs 1, 4, 5, and 6 will execute sequence #39

Step ⑧ Cycle power to the Compumotor Plus. The system will execute Sequence #40.

Step ⑨ Turn on the appropriate sequence-select input [set for the least Scan Time(SN)] to execute the proper sequences. Since the sequence feature (refer to the xQ1 command) was enabled during the power-up sequence, your PLC program must turn off all of the sequence-select inputs before you can select another sequence.

Registration and Synchronization

The registration function allows you to make a *move within a move* based on an external event. This is done by responding to an external trigger input, which causes the existing move to blend into a new move (the registration move). The registration move causes the motor to accelerate or decelerate as required to get to the specified end point using the parameters defined for the registration move.



Hint
Registration is used to synchronize materials which slip or change size when processed.

The typical use of registration moves is to synchronize web applications. For example, a sheet of pre-printed cardboard is to be cut up into paper plates. Registration moves synchronize the cutter with the blank space between the plates. This is done by using a sensor which picks up something unique about the web, such as a registration mark (hence the name registration moves). The web is then moved to a point a fixed distance from the registration mark and an operation such as a cut is made.

To specify the move, the acceleration, velocity, and distance of the move as well as the lockout distance are required. The lockout feature is designed to eliminate false triggering. The lockout distance is the distance to move before looking for the registration mark. Refer to the above figure. The lockout distance is specified from the beginning of the current move, the move that is aborted when the registration mark is detected. Lockout is required to prevent the registration sensor from triggering falsely on a pattern printed on the web.

To define the registration move do the following.

- Issue the registration command: `REGa,v,d,l` where *a* is the acceleration, *v* is the velocity, *d* is the distance, and *l* is the lockout distance of the registration move. This defines the registration move.
- Issue the `SSK1` command. This enables the registration feature.

Once the registration move has been defined and enabled, the Compumotor Plus begins looking for the registration mark during any move except jog and homing moves. This condition is referred to as being in registration mode.

If the parameters specified are impossible to execute the registration command is ignored. If the lockout distance is longer than the currently defined move the registration move is never executed. Refer to *Chapter 5, Software Reference* for information about how to check the move to see if it is impossible to execute.

If the currently defined move is a continuous move the registration feature operates normally, but *should registration never occur the motor will continue on just as it would if it were not in registration mode*. In contrast, when registration is used with preset moves, the move ends at the preset distance defined by the distance (`D`) command if a registration trigger never occurs. This is useful for indicating an error condition to the operator. It can also keep material waste to a minimum.

Input modes (`IM4` or `IM5`) define an input for use as the registration input. Additionally, two other inputs, may be defined as jog inputs. The jog inputs are used to position the load for initial setup (refer to *Software Reference*).

The Compumotor Plus records the motor's position as soon as the registration input is received. You should not use the `SN` command to debounce the registration input. The debounce process introduces a variable delay between receiving the registration signal and recording the motor position, reducing accuracy.

Example

Command	Description
> <code>SSK1</code>	Enable registration
> <code>IM4</code>	Set Input mode #4 as registration input
> <code>A10</code>	Set normal acceleration to 10 rps ²
> <code>V1</code>	Set normal velocity to 1 rps
> <code>D50000</code>	Set normal distance to 50000
> <code>MN</code>	Set to mode normal
> <code>REG50,20,5000,40000</code>	Set registration move to accel=50, vel = 20, distance from registration mark=5,000, wait until 40m000 steps from start of move to start looking for registration mark
*SETTING UP REG TABLE	
*REG TABLE COMPLETE	
> <code>G</code>	Execute the move (Go)

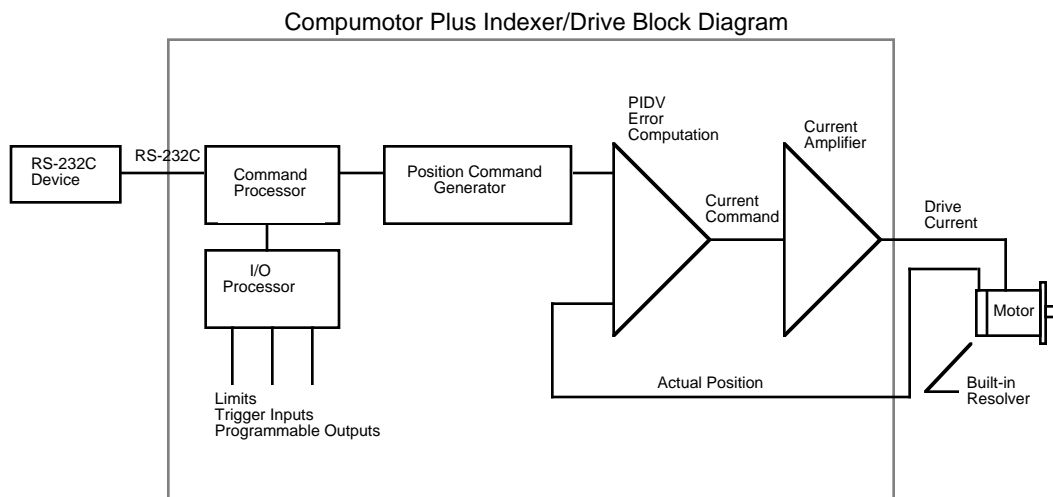
Tuning Your System

The Compumotor Plus is a servo system requiring a closed feedback loop to control the motor and thus the load. The Compumotor Plus employs a sophisticated algorithm to control the response of the motor and its load. This system is tunable: you can adjust its behavior to suit your needs. You tune the system by adjusting gains. The gains are adjusted by pushing buttons on the Compumotor Plus front panel or via commands over RS-232C. Below is a description of how the system works, how to use it and what to look out for.

Tuning Theory

The Compumotor Plus Drive can be divided into two major areas: the digital controller and the analog amplifier. All of the positioning compensation (Proportional, Integral and Derivative gains) and the velocity gains are set by the user and processed by the digital controller board. Once the values are set, they can be stored or saved in the EEPROM by issuing a Save (sv) command. The Compumotor Plus is preprogrammed at the factory with gains appropriate for the size of motor provided. Refer to the block diagram of the Compumotor Plus Drive servo system shown below.

The controller board sends two digitized waveforms from its digital-to-analog converters (DACs) to the analog amplifier board. These waveforms represent two commanded, motor-phase currents. The analog amplifier board measures the actual motor current to determine the correct voltage to apply to the motor windings. The controller commands a desired current to the amplifier board, the amplifier board then attempts to generate that desired current in the motor's windings. The position of the motor's shaft is sensed by the controller via the resolver attached to the motor. The controller uses this position information to generate the desired current command to the amplifiers.



The current command to the amplifier is based on several quantities including the position of the servo motor's shaft, the desired position (which is generated by the indexer), and previous position errors.

The controller subtracts the motor's actual position from this desired position to determine the position error. The position error is put into an equation, along with previous position errors, to generate the current command for the amplifier.

In the Compumotor Plus, the equation is a digital approximation of an analog continuous-time PID (proportional, integral, and derivative) and velocity control network. The analog continuous-time PID is traditionally used to stabilize conventional servo systems. It consists of several

potentiometers, resistors, and capacitors. In the Compumotor Plus, these parts are replaced with digital logic controlled by a microprocessor.

The digital approximation referred to above is the discrete-time equivalent to a continuous-time PID network. It is called a discrete-time PID network because it operates on sampled digital data rather than on continuous analog data. The sampling rate of the Compumotor Plus controller is the rate at which the equation is evaluated and the current command to the amplifier is updated. The sample rate of the Compumotor Plus controller is 3,333 times per second, or every 300ms. This rate is sufficiently fast to provide excellent dynamic response.

Actual motor position information is provided by a resolver built into the motor. The resolver is read three times every update period. The desired position is compared to the actual position and an error or correction value is generated.

The velocity command is calculated using the following formula.

$$V_c = P_g d_{e_n} + I_g \sum (d_{e_1 \dots e_n}) + D_g (d_{e_n} - d_{e_{n-1}})$$

where,

V_c is the commanded velocity

d_e is the position error

P_g is the proportional gain

I_g is the integral gain

D_g is the derivative gain

General Tuning Considerations

The gains set at the factory are satisfactory for most applications, but if your application requires higher performance, you can change these gains.

If you adjust the gains and the system appears unstable, use the Return to Factory settings (RFS) command or pushbutton combination to return to the factory settings (pushbuttons P and I together).

Proportional, Integral, Derivative, and Velocity tuning (P, I, D, and V) can be implemented through RS-232C commands or through the front panel pushbuttons. The factory values of the tuning algorithm are based on the following.

- The ratio of load-to-rotor inertia is 10 to 1 or less
- The load is purely inertial (i.e., there is little or no friction impeding the load)
- It is directly and solidly coupled to the motor
- A 4% velocity overshoot is tolerable

If you find that your system cannot be tuned satisfactorily using the pushbuttons on the front panel, the load to rotor inertia ratio may be higher than the default factory gain **maximums** are capable of handling. This may also be the case if there is a significant amount of friction in the system, or the coupling to the load is torsionally flexible.

Tuning usually involves choosing between a high degree of stability accuracy, or responsiveness. A stable system wants to remain at rest and is therefore not responsive. It will maintain a small, repeatable, steady state error. An accurate system which is critically damped at the end of its moves will be somewhat stable, but the cycle time for the move will be somewhat higher than the best the system can do. Finally, a very responsive system will turn in very fast cycle times, but overshoot and ringing will be significant

You can increase the speed of your machine at the price of motor/drive heating, final position accuracy, and settling time. Or you can improve system accuracy at the price of final position stability. In other words, you can trade final position accuracy for system response time. All of the gains are interactive. It may take considerable experimentation to find the exact

combination of gain values required to get the best performance in your application.

As a general rule the velocity gain should be set first, followed by proportional, integral, and finally the derivative gain. The velocity gain affects the responsiveness of the system as a whole. The proportional gain affects stiffness. The derivative gain affects settling time and dynamic response. The integral gain affects the final position accuracy. All gains interact with each other.

Gain Descriptions

The following section describes the four gains found in the Compumotor Plus: proportional, integral, velocity and derivative. The first letters of each gain type makes up the term PIDV which describes the tuning method used in the Compumotor Plus.

Velocity Gain	The velocity gain affects the overall responsiveness of the system. If the system is sluggish, increase the velocity gain. If the system overshoots unacceptably, or the motor rings at the end of each move, reduce the velocity gain. You can increase the derivative gain to compensate for post-move ringing as well.
Proportional Gain	Proportional gain affects system stiffness and accuracy. As the proportional gain increases, the influence of the feedback signal increases. If the gain is too high, the system oscillates. This happens because very small resolver changes are amplified into very large error signals. Eventually the motor begins to lag behind the feedback signal, causing the feedback and the command signals to be in phase. This is what causes oscillation.
Integral Gain	Integral gain allows the system to compensate for steady-state position errors (due, for example, to friction). Integral gain also reduces velocity ripple. It does this by slowing down the electronic response time so that it more closely resembles the response of the mechanical components of the system.
Derivative Gain	Derivative gain adds damping to the system. This damping helps reduce the oscillations at the end of moves, and ringing when the velocity is suddenly changed (at the end or beginning of an acceleration ramp). When this happens, increasing the derivative gain will reduce the oscillation. The derivative gain adds phase lead to compensate for the system's natural phase lag.

Tuning Your System Manually

The Compumotor Plus uses a conventional discrete-time PIDV servo algorithm. The algorithm is a recursive function using four user adjustable gains to control system responsiveness and accuracy. The four gains model static and dynamic characteristics of the motion control system and its load. The gains are adjusted to stabilize the system.

The four gains in a Compumotor Plus are normalized to a range of 0 to 100. This makes tuning the system much simpler due to the small numbers employed. To ensure that you have sufficient range in the gain settings a gain maximum is provided for each gain which allows you to change the meaning of the gain number from 0 to 100. Refer to the **CPM**, **CIM**, **CVM**, and **CDM** commands in *Chapter 5, Software Reference* for more information.

Tuning a Compumotor Plus Drive servo system usually requires adjusting only one controller gain (normally velocity). The other gains are predefined and, in most cases, require no further adjustment.

If you find that you are tuning a given parameter in the extreme range (i.e., 0-5% or 95-99%), you might consider changing the maximum gain value via the **CIM**, **CPM**, **CVM**, and **CDM** commands.

Once you have the system installed and the motor connected to its intended load, you can determine whether any fine tuning is required by observing the

response of the system to commands from your indexer and by observing how stiff the system is when at rest.

With the motor at rest, try to deflect the shaft. You should not be able to easily turn the shaft away from its rest position. If it feels very soft, the system gains probably need to be increased. A very soft system does not respond very quickly to move commands.

If the shaft feels stiff, check that the system is not vibrating. If it is, the gain may be too high. This causes the drive to provide excess current, and can shorten the life of mechanical components. In extreme cases the vibration grows, producing violent motions that cause the drive to fault or possibly break the equipment it is attached to. This is called instability. For this reason, you should tune the Compumotor Plus Drive with some caution.

Pushbutton Tuning

There are two methods available to adjust an Compumotor Plus Drive's servo compensation network.

- The six pushbuttons on the Compumotor Plus Drive's front panel
- The RS-232C communications port

The buttons are labeled UP, DOWN, P, I, V, and D and are defined as follows. Refer to *Chapter 3, Installation* for more information on pushbuttons.

- P selects the PROPORTIONAL gain
- I selects the INTEGRAL gain
- V selects the VELOCITY gain
- D selects the DERIVATIVE gain

Select one of the P, I, V, or D buttons and hold it down. When you do, the two-digit display displays the present value for the gain you have selected (from 0-99%). You should note this value in case you wish to return to it. While holding the button for the gain you selected, push the UP button to increase the gain. Push the DOWN button to decrease the gain. Continue to hold the UP or DOWN button and the proportional gain changes automatically.

 **Hint**
Don't forget to
SAVE!

The value displayed when you release the UP or DOWN button becomes the new gain setting. This value is used by the Compumotor Plus immediately. However, you must issue a **SAVE** or **SV** command to make the change permanent. Because the gains must be manually saved you may return to your previous settings by resetting the drive, by cycling power, or issuing the **Z** command. This recalls the last setting in non-volatile memory.

If you wish to retrieve the factory default values, all you need to do is push both the P and I gain buttons together. This causes the factory values for each gain to be reloaded.

After you have completed the pushbutton tuning procedure, it will be necessary to save the values you have selected into non-volatile EEPROM. This is done by pressing all four gain buttons (P, I, D, & V).

Tuning Via RS- 232

The tuning process of the front-panel switches can be duplicated via the RS-232C communication port.

You use the following commands to set the gains.

- CPG sets the proportional gain
- CIG sets the integral gain
- CDG sets the derivative gain
- CVG sets the velocity gain

To set the gain maximums use the following commands.

- CPM sets the proportional gain maximum
- CIM sets the integral gain maximum
- CDM sets the derivative gain maximum
- CVM sets the velocity gain maximum

Tuning Your System Automatically

With the Z5 revision level software (see the **RV** command), your Compumotor Plus Drive/Indexer is equipped with the ability to tune itself. Self-tuning refers to the procedure by which the Compumotor Plus operating system automatically determines the proper servo gains for your application.

You use the **TUNE** command to implement self-tuning. Once self-tuning is complete you have two sets of gains you can use: PIDV and Pole-placed. You use the **GAINX** command to select between the two sets of gains.


The gains computed with the **TUNE** command are high performance gains. This means that they are fairly high for typical loads. Consequently, this implementation of self-tuning is intended for loads below ten times the rotor inertia. When large loads are used with this system, the gains computed are typically too large to be of use. You may be able to use option four to produce stable gains for large loads. See below for a description of the **TUNE** options.

If you need to drive loads in the 10:1 rotor inertia range and above, you should use PIDV tuning. If you find that the resulting gains produce excessive noise in the motor, you may want to filter out this noise with the **FILT** command. The **FILT** command sets the digital torque filter's time constant. Since larger loads typically result in lower system bandwidth, you may be able to increase the digital torque filter's time constant from the default value of 5 ms to a value as high as 10 ms.

The algorithm used in the Compumotor Plus is referred to as a self-tuning regulator, and it consists of two main stages. Stage one is system identification. When the **TUNE** command is invoked, the Compumotor Plus executes a brief predefined move sequence. This sequence consists of a series of short step moves. Step moves are moves with infinitely high acceleration and deceleration. The move sequence involves both the motor and your application load. Since some applications have restrictions on how they can move, the **TUNE** command has three move options described below.

- Move one (default) consists of back-and-forth, 512 step moves (at 12,800 steps/rev)
- Move two consists of CW 512 step moves
- Move three consists of CCW 512 step moves

During the move sequence, the Compumotor Plus records the value of current sent to the motor, and the position of the motor for 1500 time intervals. This data is then used to estimate the performance characteristics of the motor and load. In particular, the system determines the total load of the motor and the application.

 **Hint** *Secure the motor to the load*

For system identification to be as accurate as possible, it is important that you secure the motor to its mount and firmly couple the shaft to the load. If the motor body moves around or the coupler slips during the preset move, the resulting data will be flawed and the system will not be able to compute the proper gains.

The Four Options

After the data has been acquired, the operating system shuts the amplifier off and analyzes the data. The Compumotor Plus issues the following message.

```
*TUNING_BEGUN  
>
```

Because of the large amount of data being processed, this analysis takes approximately two minutes to complete. At the end of this time, the system uses the estimated total system load to compute the appropriate gains. It then issues the following message,

```
*TUNING_COMPLETE  
>
```

In order to compute the proper gains for an application, there has to be some desired final closed-loop performance specified. Different sets of gains result in different final performances. To make this choice easier for you, the **TUNE** command provides four optional final motor responses as describe below.

- Option one specifies responses with very low overshoot but limited in position stiffness.
- Option two specifies responses with some overshoot on fast accelerations but greater in position stiffness.
- Option three specifies responses with high in-position stiffness and greater overshoot on fast accelerations than options one and two.
- Option four specifies responses using a lower set of gains for very large loads.

Tuning a servo system always involves trade-offs in selecting desired performance characteristics. These four options offer you flexibility in tuning your Compumotor Plus.

After the self-tuning command has calculated the new gains, the Compumotor Plus re-energizes the amplifier. The system then waits approximately five seconds with the new gains in place to confirm that the system is stable.

Do not touch the rotor at this time because this will be interpreted as instability in the system. When the calculated gains are found to be unstable, the system will send the message **TUNING_COMPLETE** to the terminal and return the system to the PIDV control scheme.

The **TUNE** command is designed to catch all instances of unstable gains and default to PIDV when they occur. Nonetheless, it is a good idea to make sure that the maximum following error is set to a reasonable value (e.g. 1 rev) with the **CPE** command before self-tuning. This causes the Compumotor Plus to fault if the motor moves more than one revolution away from the commanded position.

The gains computed with the **TUNE** command form a control structure known as a *Pole-Placed Controller*. These gains are not directly compatible with the PIDV control structure. This means that either you use the self-tuning gains or you use the PIDV gains. The **GAINX** command allows you to switch between these different control schemes at any time. **GAINX:1** implements the self-tuning gains previously computed; **GAINX:0** implements the PIDV gains. This command is a buffered command so it can be used in sequences. This command is also independently saved so you can determine which control scheme will be in effect when the system is powered up.

*Do not issue the **GAINX:1** command when self-tuning has not been performed and do not issue this command if the results of the self-tuning were unstable gains. Issuing **GAINX:1** under these conditions can result in motor instability.*

Self Tuning Procedure

- Step ① Mount the motor and connect it to the load. Motor should be firmly mounted and there should be no slippage in the coupling.
- Step ② The **TUNE** command takes two parameters: tuning-move and tuning-option. In the command **TUNE:m,o** the **m** refers to the tuning-move and the **o** refers to the tuning-option.
- Determine which tuning move to use. If the load cannot move in a particular direction or if there is known backlash in the system, use **TUNE:2,o** or **TUNE:3,o**. Otherwise use **TUNE:1,o**.
 - Determine what closed-loop response you want for your application. If in-position stiffness is important, use **TUNE:m,2** or **TUNE:m,3**. If low overshoot is important, use **TUNE:m,1**. If you are using a larger load.
- Step ③ Issue the **TUNE** command. If you want to use tuning-move 1 with tuning-option 2 type the following:
- ```
> TUNE:1,2
```
- If the motor does not move, reset the system and check your limit status. The motor must move for self-tuning to work properly.
- Step ④ Wait for the system to compute new gains. When the amplifier is re-energized, do not touch the rotor until the system has responded with the following.
- ```
> *TUNING_COMPLETE
```
- Step ⑤ If no error message has been sent to the terminal, the new self-tuned gains are now in place. If you wish to save these gains in non-volatile memory, issue the **SV** command. Once saved, this control structure is automatically put in use when the system is reset or powered up. To return to PIDV control, issue the **GAINX:0** command. To have PIDV control implemented on power-up, issue **SV** after **GAINX:0**.

Tuning Problems

The following section describes common problems you may run into while tuning your system

Ringing and Overshoot

If the system exhibits excessive overshoot (when making the transition from acceleration to constant velocity or from deceleration to a stop), it may be caused by a commanded acceleration that is much higher than the system can actually achieve. If this occurs, reduce the move acceleration with the **A** command until the overshoot is at an acceptable level. Reducing the integral gain also limits overshoot in most cases.

Position Errors Due to Load Inertia

Load is the inertia (mass times the radius of rotation squared) seen by the shaft of the motor (measured in oz/in/in). The amount of inertia affects the torque required. The torque required is a function of acceleration and inertia. If you find that your load inertia is larger than 10 times the motor inertia, you may want to trade some speed performance for accuracy at the final position. In this case, reduce the velocity gain and increase the integral gain.

Slow Response Due to Load Inertia

In a system where the load inertia is larger than the factory setting supports, it may be possible to improve the system response by allowing the overshoot to increase. In this case, increase the velocity gain, increase the proportional gain, and decrease the integral gain.

Position Errors Due to Friction

If load friction represents a large percentage of the motor's torque, the end-of-move position may have an unacceptable error. In this case, you can trade off system response for final position accuracy. To do this, increase the integral gain, increase the derivative gain, and if necessary, decrease proportional or velocity gains.

Shaft and
Coupling
Vibration

If the load is coupled to the shaft through a non-rigid coupler, it is possible for the shaft and coupler to oscillate at a frequency greater than the system's natural frequency. In this case, it may be possible to trade system response for system stability. To do this increase the integral and derivative gains, while decreasing velocity or proportional gains.

Inadequate
Response Time
(Frequency
Response)

Some systems may involve very little inertia, but need very high acceleration. In this case, you may want to narrow the range of values that the system can handle to optimize the move profile for a light load. To do this, increase the proportional and velocity gains while decreasing the integral gain.