

Feature Implementation

The information in this chapter will enable you to understand and implement the 6270's features into your application:

- Support Software (Motion Architect, DOS Disk)
- Safety Features
- Scaling
- End-of-Travel Limits
- Homing
- Positioning Modes
- Dithering Hydraulic Servo Valves
- User Interface Options:
 - Programmable I/O
 - Thumbwheel Interface
 - I/O Device Interface (including PLCs)
 - Joystick Interface
 - 14-Bit Analog Input Interface (**6270-ANI Option only**)
 - RP240 Remote Operator Panel Interface
 - Host Compumotor Control
- Variables
- Teach Mode
- S-Curve Profiling
- X-Y Linear Interpolation
- RS-232C Daisy-chaining

See Also, Chapter 4

Tuning-related features are described in chapter 4:

- Selecting a servo sampling rate (*SSFR*)
- Selecting a max. position error (*SMPER*)
- Tuning Procedures
- Gains
- Gain Sets
- Setpoint Window Gain Sets
- Target Zone Settling Mode

Before You Proceed With This Chapter



WARNING



Most of the features described in this chapter are used for operating your system's electrical and mechanical components. Therefore, before proceeding with this chapter, you should test your system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel. Be sure to complete all the installation and test procedures provided in Chapter 2 and Chapter 3, and complete the tuning procedures in Chapter 4 or the **Servo Tuner User Guide**.

EMERGENCY SHUTDOWN: At all times, you should be prepared to shut down the valve or drive (e.g., if the system becomes unstable or experiences a runaway). You can use the **ENBL** input (disconnect it from ground) to disable the 6270's analog output signal. An alternative is to issue the `@DRIVEØ` command to the 6270 over the communication interface, but this requires connecting a shutdown output to the drive. If the drive does not have a shutdown input, use a manual emergency stop switch to disable the valve's/drive's power supply.

6000 Series Software Reference Guide

Since this chapter often refers to the 6000 Series Command Language employed by the 6270's operating system, keep the **6000 Series Software Reference Guide** nearby as a reference for programming guidelines and detailed command descriptions.

Support Software

The 6270 is shipped with two support software tools, Motion Architect™ and the *6000 DOS Support Disk*.

6000 DOS Support Disk

The *6000 DOS Support Disk* (p/n 95-012266-01) contains a program that provides terminal emulation and program editing capabilities specifically designed for use with any 6000 Series stand-alone product. Also included on the disk are sample 6000 command language programs. For more detailed user information, refer to the **6000 DOS Support Disk Quick Reference**.

Motion Architect®

Motion Architect® is an intuitive Microsoft® Windows™ based programming tool. A brief description of Motion Architect's basic features is provided below. For more detailed user information, refer to the **Motion Architect User Guide**.

- **System Configurator and Code Generator:** Automatically generate controller code of basic system set-up parameters (I/O definitions, encoder operations, etc.).
- **Tuning and Data Gathering Tool (Servo Tuner option for Motion Architect):** Tune the 6270 and the attached servo drives (if any) and receive instant data feedback on customizable displays. The Servo Tuner™ option is an add-on module and does not automatically come with the basic Motion Architect software package. To order your copy of Motion Architect Servo Tuner, contact your local Automation Technology Center.
- **Program Editor:** Create blocks or lines of 6270 controller code, or copy portions of code from previous files. You can save program editor files for later use in BASIC, C, etc., or in the terminal emulator or test panel.
- **Terminal Emulator:** Communicating directly with the 6270, the terminal emulator allows you to type in and execute controller code and transfer code files to and from the 6270.
- **Test Panel and Program Tester:** You can create your own test panel to run your programs and check the activity of I/O, motion, system status, etc. This can be invaluable during start-ups and when fine tuning machine performance.
- **On-line Context-sensitive Help and Command Reference:** These on-line resources provide help information about Motion Architect, as well as interactive access to the contents of the **6000 Series Software Reference Guide**.
- Options (available from your local Automation Technology Center or distributor):
 - **Servo Tuner™** (Tuning and Data Gathering Tool): Tune the 6270 and receive instant data feedback on customizable displays. The Servo Tuner option is an add-on module and does not automatically come with the basic Motion Architect software package.
 - **CompuCAM™:** CAD-to-Motion software allows you to translate DXF, HP-GL, and G-Code files into 6000 Series Language motion programs.

6270 Safety Features

To help ensure a safe operating environment, you should take advantage of the 6270's safety features listed in the table below.

Feature	Description	See Also
Enable Input	<p>The enable input (ENBL), found on pin #14 on the AUX connector, is provided as an emergency stop input to the 6270.</p> <p>When you open the ENBL input, with respect to GND, the analog output voltage between CMD+ and CMD- is clamped to almost zero, and the shutdown outputs are activated on both axes. <i>Clamping occurs independent of the microprocessor and the DSP.</i> (The clamping circuit is also connected to the watchdog timer; if the 6270's microprocessor fails, the analog output voltage will be clamped.)</p>	Chapter 3: <i>System Connections</i>
Shutdown Outputs (for drives only)	<p>The 6270 uses the shutdown outputs to disable the servo drive if it detects a problem. Two types of relay outputs are found on both DRIVE connectors—SHTNC for drives that require a closed contact to disable the drive, and SHTNO for drives that require an open contact to disable the drive.</p> <p>The shutdown relay outputs are essential for smooth power-up and power-down of the system. The shutdown relay is active (disabling the drive) when no power is applied to the 6270. When the 6270 is powered up, the shutdown relay remains active until you issue the DRIVE11 command.</p>	Chapter 3: <i>Motor Driver Connections</i>
Drive Fault Inputs (for drives only)	<p>The drive fault (DFT) inputs, found on pin #5 of both DRIVE connectors, allows the drives to tell the 6270 if they encounter a fault condition. When a drive fault occurs, the 6270 stops motion (at the rate set with the LHAD command) and terminates program execution. No drive shutdown will result unless it is initiated with an ERRORP error program.</p>	Chapter 3: <i>Motor Driver Connections</i>
End-of-travel Limit Inputs	<p>End-of-travel limits prevent the load from crashing through mechanical stops, an incident that can damage equipment and injure personnel.</p> <p>You can use hardware or software limits, as your application requires. Hardware limits use the CW and CCW terminals on the LIMITS connector. Software limits are set with the LSCCW and LSCW commands.</p>	Chapter 5: <i>End-of-Travel Limits</i>
User Fault Input	<p>Using the INFCi-F command, you can assign any of the programmable inputs the <i>user fault</i> function. You can then wire the input to activate when an external event, considered a <i>fault</i> by the user, occurs.</p>	Chapter 5: <i>Input Functions</i>
Maximum Allowable Position Error	<p>A <i>position error</i> (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the feedback device (TFB). The maximum allowable position error is set with the SMPER command.</p> <p>When the maximum allowable position error is exceeded (usually due to instability or loss of position feedback), the 6270 shuts down the drive and sets error status bit #12 (reported by the TER command).</p> <p>If SMPER is set to zero (SMPER0), the position error will not be monitored.</p>	Chapter 4: <i>Servo Tuning</i>

 *Programmed Error-Handling Responses*

When any of the safety features listed above are exercised (e.g., **ENBL** input is opened, **DFT** input is activated, etc.), the 6270 considers it an error condition. With the exception of the shutdown output activation, you can enable the **ERROR** command to check for the error condition, and when it occurs to branch to an assigned **ERRORP** program. Refer to the *Error Handling* section in the **6000 Series Software Reference Guide** for further information.

Scaling

The scaling commands allow you to scale acceleration, deceleration, velocity, and position to values that are appropriate for the application. The SCALE, SCLA, SCLV, SCLD, PSCLA, and PSCLV commands are used to implement the scaling features. The default condition of the 6270 is with scaling enabled (SCALE1).

Scaling parameters are specific to the current feedback source. (See programming example on page 24.)

The scaling factors for each axis are specific to the current feedback source (selected with the last SFB command). If your application requires switching between feedback sources on the same axis, then for each feedback source, you must issue the appropriate SFB command and enter the scaling parameters specific to operating with that feedback source.

NOTE

To maximize the efficiency of the 6270's microprocessor, the scaling multiplications are performed when the program is defined or downloaded. Therefore, you must enable scaling (SCALE) and define the scaling factors (SCLD, SCLA, SCLV, PSCLA, PSCLV) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; you should also put the scaling factors into the startup (STARTP) program.

When Scaling is Disabled

- All programmed accel and decel values are entered in encoder revs/sec², LDT inches/sec², or ANI volts/sec²; encoder and LDT values are internally multiplied by the encoder resolution (ERES) or the LDT resolution (LDTRES) value to obtain acceleration and deceleration values in steps/sec² for the motion trajectory calculations.
- All programmed velocity values are entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) or LDT resolution (LDTRES) value to obtain velocity values in steps/sec for the motion trajectory calculations.
- The distance values (D and PSET) are entered in encoder, LDT, or ANI counts. These values are internally represented as steps.

Acceleration & Deceleration Scaling (SCLA/PSCLA)

If scaling is enabled (SCALE1), all accel/decel values entered are internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec² to encoder, LDT, or ANI steps/sec². Acceleration scaling affects the following commands: A, AA, AD, ADA, HOMAD, HOMADA, JOGA, JOGAA, JOGAD, JOGADA, JOYA, JOYAA, JOYAD, JOYADA, LHAD, LHADA, LSAD, and LSADA.

Path Scaling: If you are using the 6270's linear interpolation feature, the PA, PAA, PAD and PADA commands are affected by the path acceleration scaling factor (PSCLA).

As the acceleration scaling factor (SCLA/PSCLA) changes, the accel/decel command's range and its decimal places also change (see table below). An acceleration value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLA10, the A9.9999 command would be truncated to A9.9.

SCLA/PSCLA Value	Decimal Places	Max. Accel/Decel	Min. Accel/Decel (resolution)
1 - 9	0	LDT Feedback:	LDT Feedback:
10 - 99	1	$\frac{999.9999 \times \text{LDTRES}}{\text{SCLA}}$	$\frac{0.001 \times \text{LDTRES}}{\text{SCLA}}$
100 - 999	2	Encoder Feedback:	Encoder Feedback:
1000 - 9999	3	$\frac{999.9999 \times \text{ERES}}{\text{SCLA}}$	$\frac{0.001 \times \text{ERES}}{\text{SCLA}}$
10000 - 99999	4	ANI Feedback:	ANI Feedback:
100000 - 999999	5	$\frac{818999.9181}{\text{SCLA}}$	$\frac{0.819}{\text{SCLA}}$

Velocity Scaling (SCLV/PSCLV)

If scaling is enabled (SCALE1), all velocity values entered are internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec. Velocity scaling affects the following commands: V, HOMV, HOMVF, JOGVH, JOGVL, JOYVH, and JOYVL.

Path Scaling: If you are using the 6270's linear interpolation feature, the PV command is affected by the path velocity scaling factor (PSCLV). As the velocity scaling factor (SCLV/PSCLV) changes, the velocity command's range and its decimal places also change (see table below). A velocity value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLV10, the V9.9999 command would be truncated to V9.9.

SCLV/PSCLV Value (steps/unit)	Velocity Resolution (units/sec)	Decimal Places	Max. Velocity Calculation
1 - 9	1	0	LDT Feedback:
10 - 99	0.1	1	$\frac{1000 \times \text{LDTRES}}{\text{SCLV}}$
100 - 999	0.01	2	Encoder Feedback:
1000 - 9999	0.001	3	$\frac{1000 \times \text{ERES}}{\text{SCLV}}$
10000 - 99999	0.0001	4	ANI Feedback:
100000 - 999999	0.00001	5	$\frac{1000 \times 819}{\text{SCLV}}$

Distance Scaling (SCLD)

If scaling is enabled (SCALE1), D and PSET command values are internally multiplied by the distance scaling factor (SCLD). Since the SCLD units are in terms of steps/unit, all distances will thus be internally represented in encoder, LDT, or ANI steps. For instance, if the distance scaling factor is 10000 (SCLD10000) and you enter a distance of 75 (D75), the actual distance moved will be 750,000 (10000 x 75) encoder or LDT steps.

LDT Users—Programming In Inches Or Millimeters

The default SCLD value is 432 (SCLD432, 432), which allows LDT users to program in inches. To program in millimeters, use a SCLD value of 17 (SCLD17, 17). These factors must be multiplied by the number of recirculations if the LDT uses more than one recirculation.

As the SCLD scaling factor changes, the distance command's range and its decimal places also change (see table below). A distance value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLD400, the D105.2776 command would be truncated to D105.277.

SCLD (steps/unit)	Distance Resolution (units)	Distance Range (units)	Decimal Places
1 - 9	1.0	0 - ±99999999.9	1
10 - 99	0.10	0.0 - ±9999999.99	2
100 - 999	0.010	0.00 - ±999999.999	3
1000 - 9999	0.0010	0.000 - ±99999.9999	4
10000 - 99999	0.00010	0.0000 - ±9999.99999	5
100000 - 999999	0.00001	0.00000 - ±999.999999	5

NOTE

FRACTIONAL STEP TRUNCATION

NOTE

If you are operating in the incremental mode (MA0), when the distance scaling factor (SCLD) and the distance value are multiplied, a fraction of one step may possibly be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate over a period of time, when performing incremental moves continuously in the same direction. To eliminate this truncation problem, set SCLD to 1, or a multiple of 10.

Scaling Example

Axis #1 controls a 4,000 step/rev servo motor/drive system (using a 1000-line encoder). Attached is a 5-pitch leadscrew that he wants to position in inches (4,000 steps/rev x 5 revs/inch = 20,000 steps/inch).

Axis #2 controls a servo valve and hydraulic cylinder using position feedback from an LDT with a gradient of 9.0227 $\mu\text{s/inch}$, and 2 recirculations. The user would like to program in inches.

Command	Description
> SFB1,3	Feedback devices: encoder for axis 1, LDT for axis 2
> ERES4000	Set encoder resolution to 4,000 steps/rev (post quadrature)
> LDTGRD,9.0227	Set LDT gradient to 9.0227 $\mu\text{s/inch}$
> LDTRES,864	Set LDT resolution to 864 to accommodate 2 recirculations
> SCALE1	Enable scaling
> SCLD20000,864	Distance scale factors (20,000 = 5 pitch * 4000 steps/rev)
> SCLV20000,864	Velocity scale factors
> SCLA20000,864	Acceleration and deceleration scale factors
> D5,4	Set move distance to 5 inches for axis 1 and 4 inches for axis 2
> A5,8	Set acceleration to 5 in/sec ² on axis 1 and 8 in/sec ² on axis 2
> V11,4	Set velocity to 11 in/sec on axis 1 and 4 in/sec on axis 2
> GO11	Initiate move on both axes

End-of-Travel Limits

The 6270 can respond to both hardware and software end-of-travel limits.

NOTE

If you are not using hardware end-of-travel limits in your application, you must disable them. There are two methods available:

- Use the LH command (e.g., to disable hardware and software limits on axes 1 and 2, issue the LH0,0 command).
- Find the LIMITS connector on the front panel and jumper the end-of-travel limit terminals (CW and CW) to the GND terminal.

Refer to Chapter 3, Installation, for instructions to wire hardware end-of-travel limit switches.

End-of-travel limits prevent the motor's load from traveling past defined limits. Once a hardware or software limit is reached, the 6270 will decelerate that axis at a rate specified with the LHAD or LSAD command. Typically, software and hardware limits are positioned in such a way that when the software limit is reached the motor will start to decelerate towards the hardware limit. This will allow for a much smoother stop at the hardware limit. Software limits can be used regardless of incremental or absolute positioning. Refer to the LH, LS, LHAD, LHADA, LSAD, and LSADA commands in the **6000 Series Software Reference Guide** for more information.

The example below uses the SCLD scaling command to define software limits in millimeters. Software limits are defined by the LSCW and LSCCW commands, and enabled with the LS command. The software limits are referenced from a position of absolute zero. Both software limits may be defined with positive values (axis #2 in example below) or negative values. *Care must be taken when performing incremental moves because the software limits are always defined in absolute terms.* They must be large enough to accommodate the moves, or a new zero point must be defined (using the PSET command) before each move.

NOTES

Soft Limits: To ensure proper motion when using soft end-of-travel limits, be sure to set the LSCW value to a greater absolute value than the LSCCW value. (CW refers to extension, CCW to retraction)

Linear Positioning Systems: In this user guide, it is assumed that you have connected the valve so that clockwise (CW) refers to extension and counter-clockwise (CCW) refers to retraction. This convention is accurate if you connect the valve as instructed in Chapter 3.

Example Set Up	Command	Description
>	SFB3,3	Use LDTs as position feedback devices for both axes
>	LDTRES432,432	Set LDT resolution to 432 (no recirculations)
>	SCALE1	Enable scaling
>	SCLD17,17	Distance scale factor (allows programming in millimeters)
>	SCLV17,17	Velocity scale factor
>	SCLA17,17	Acceleration and deceleration scale factor
>	LH3,3	Enable limits on axes 1 and 2
>	LHAD10,10	Hard limit deceleration
>	LSAD5,10	Soft limit deceleration
>	LSCW10,20	Establish CW (extension direction) soft limit
>	LSCCW0,2	Establish CCW (retract direction) soft limit
>	LS3,3	Enable soft limits on axes 1 and 2

Homing (Using the Home Inputs)

Refer to Chapter 3, Installation, for instructions to wire hardware home limit switches.

The HOM command initiates a sequence of moves that position an axis using the Home and/or the Z channel inputs. The result of any homing operation is a repeatable initial starting location. The home inputs to be used, the edge of those inputs, and the final approach direction may all be defined by the user. If the encoder's Z channel input is to be used, the HOMZ command must be enabled. The input polarity (normally-open or normally-closed) of the home input or switch is defined with the HOMLVL command.

The velocity for a move to the home position is specified with the HOMV command. The acceleration and deceleration rates are specified with the HOMA and HOMAD commands, respectively. (HOMAA and HOMADA are also used if you are using S-curve Profiling—see *S-Curve Profiling* section below for details.) If backup to home (HOMBAC) is enabled, the velocity of the final approach toward the home position is specified with the HOMVF command.

Enabling backup to home (HOMBAC) allows you to use two other homing features, HOMEDG and HOMDF. The HOMEDG command allows you to specify the side of the home switch on which to stop. The HOMDF command allows you to specify the final approach direction. If HOMBAC is not enabled, HOMEDG and HOMDF will have no effect on the homing algorithm (see Figures A and B).

Figures A and B show the homing operation when HOMBAC is not enabled. If a limit is encountered during the homing operation, the motion will be reversed and the home switch will be sought in the opposite direction. If a second limit is encountered, the homing operation will be terminated, stopping motion at the second limit.

After a homing operation is successfully completed, the absolute position register is reset to zero (applies also to the voltage register for ANI feedback).

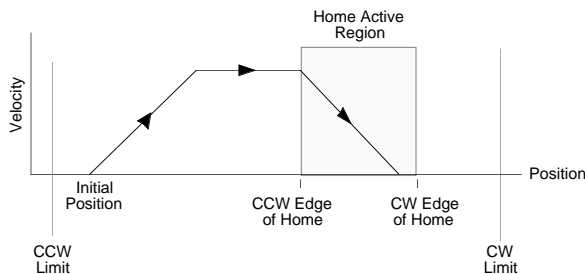


Figure A. Homing in a CW Direction (HOM0) with backup to home disabled (HOMBAC0)

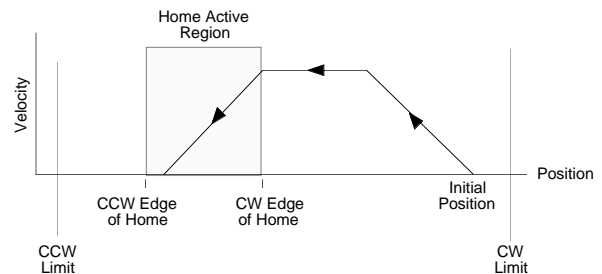


Figure B. Homing in a CCW Direction (HOM1) with backup to home disabled (HOMBAC0)

CW Homing with Backup to Home Enabled

The seven steps below describe a sample homing operation when HOMBAC is enabled (see Figure C). The final approach direction (HOMDF) is CW and the home edge (HOMEDG) is the CW edge.

NOTE

To better illustrate the direction changes in the backup-to-home operation, the illustrations in the remainder of this section show the backup-to-home movements with varied velocities. In reality, the backup-to-home movements are performed at the same velocity (HOMVF value).

- Step 1** A CW home move is started with the HOMØ command at the HOMA and HOMA_A accelerations. Default accel is 10 inches/sec².
- Step 2** The HOMV velocity is reached (move continues at that velocity until home input goes active).
- Step 3** The CCW edge of the home input is detected, this means the home input is active. At this time the move is decelerated at the HOMAD_A and/or HOMAD command values. It does not matter if the home input becomes inactive during this deceleration.
- Step 4** After stopping, the direction is reversed and a second move with a peak velocity specified by the HOMVF value is started.
- Step 5** This move continues until the CCW edge of the home input is reached.
- Step 6** Upon reaching the CCW edge, the move is decelerated at the HOMAD and HOMA_A command values, the direction is reversed, and another move is started in the CW direction at the HOMVF velocity.
- Step 7** As soon as the home input CW edge is reached, this last move is immediately terminated. The load is at home and the absolute position register is reset to zero.

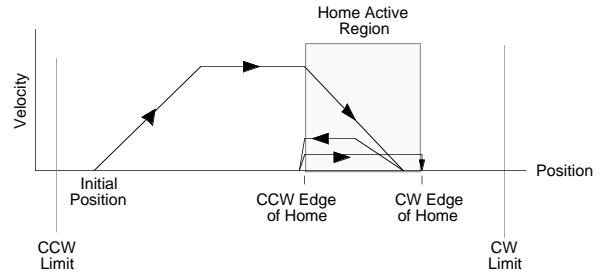


Figure C. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDGØ, HOMDFØ

Figures D through F show the homing operation for different values of HOMDF and HOMEDG, when HOMBAC is enabled.

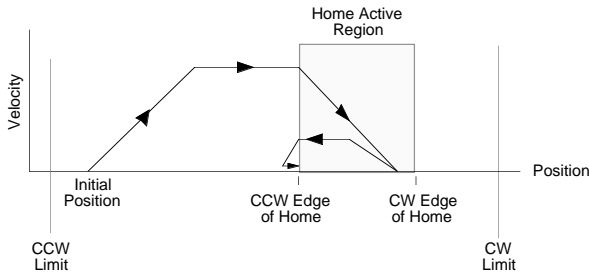


Figure D. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDG1, HOMDFØ

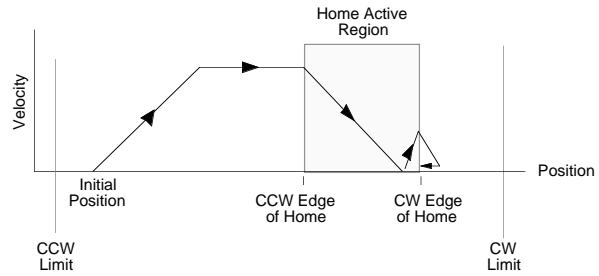


Figure E. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDGØ, HOMDF1

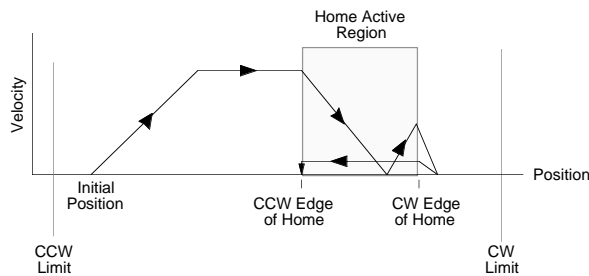


Figure F. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDG1, HOMDF1

CCW Homing with Backup to Home Enabled

Figures G through J show the homing operation for different values of HOMDF and HOMEDG, when HOMBAC is enabled.

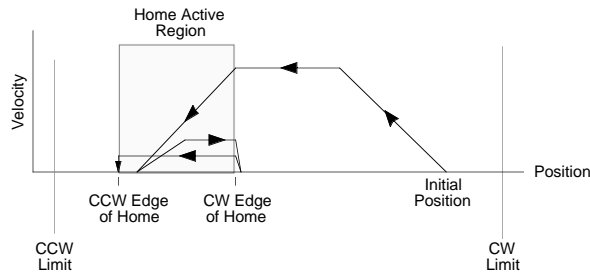


Figure G. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDF1

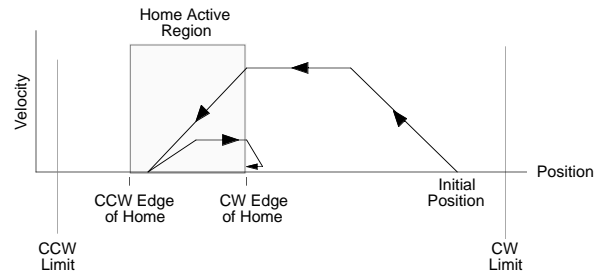


Figure H. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG0, HOMDF1

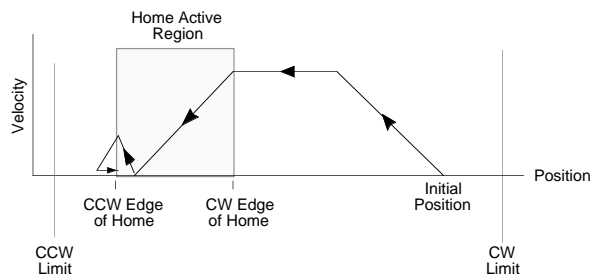


Figure I. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDF0

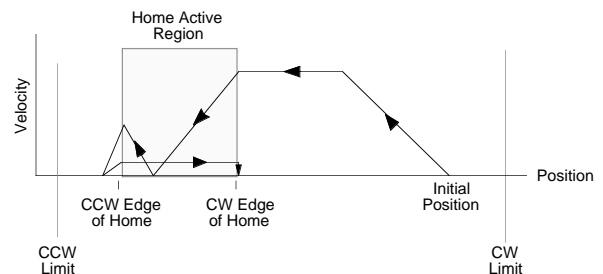


Figure J. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG0, HOMDF0

Move HOME Using The Z-Channel

Figures K through O show the homing operation when homing to an encoder index pulse, or Z channel, is enabled (HOMZ1). The Z-channel will only be recognized after the home input is activated. It is desirable to position the Z channel within the home active region; this reduces the time required to search for the Z channel.

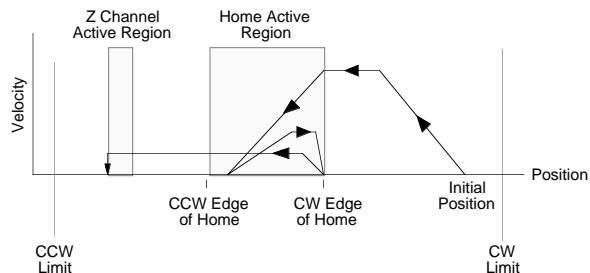


Figure K. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDF1

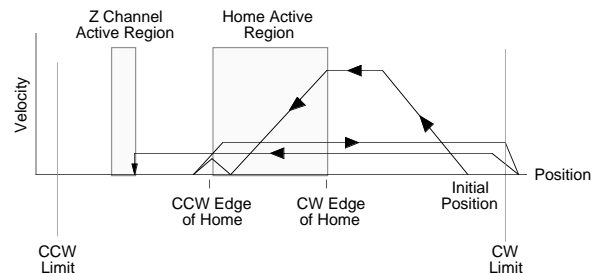


Figure L. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG0, HOMDF0

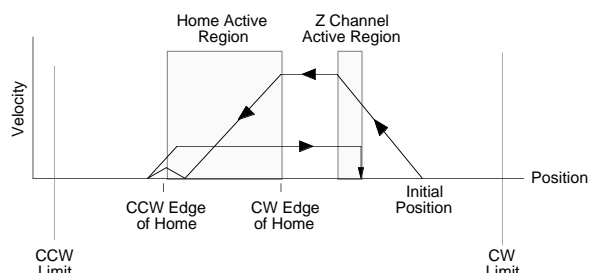


Figure M. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG0, HOMDF0

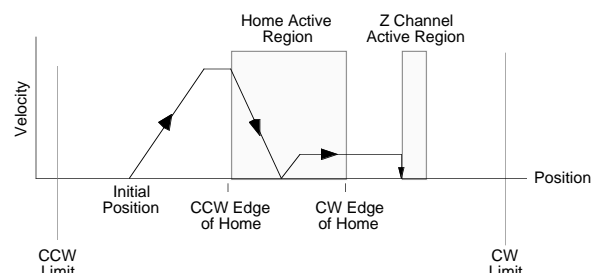


Figure N. Homing in a CW Direction (HOM0) with HOMBAC0, HOMEDG0, HOMDF0

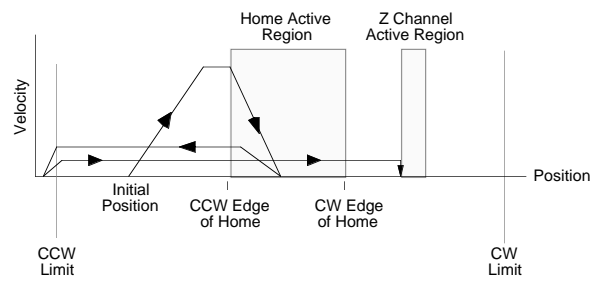


Figure O. Homing in a CW Direction ($HOM\emptyset$) with $HOMBAC\emptyset$, $HOMEDG\emptyset$, $HOMDF1$

Positioning Modes

The 6270 can be programmed to position in either the preset (incremental or absolute) mode or the continuous mode. You should select the mode that will be most convenient for your application. For example, a repetitive cut-to-length application requires incremental positioning. X-Y positioning, on the other hand, is better served in the absolute mode. Continuous mode is useful for applications that require constant movement of the load based on internal conditions or inputs, not distance.

Refer also to the Scaling section above.

Positioning modes require acceleration, deceleration, velocity, and distance commands (*continuous mode does not require distance*). The following table identifies these commands and their units of measure, and which scaling command affects them.

Parameter	Commands	Units (Unscaled)	Unit Scaling Command *
Acceleration	A, AA	Encoder Feedback: revs/sec ² (rps ²) LDT Feedback: inches/sec ² (ips ²) ANI Feedback: volts/sec ² (vps ²)	SCLA or PSCLA
Deceleration	AD, ADA	rps ² or ips ² or vps ²	SCLA or PSCLA
Velocity	V	rps or ips or vps	SCLV or PSCLV
Distance	D	steps (counts from encoder or LDT or ANI input)	SCLD

* Scaling must first be enabled with the SCALE1 command. PSCLA and PSCLV are for linear interpolated moves.

Preset Mode

A *preset move* is a point-to-point move of a specified distance. You can select preset moves by putting the 6270 into preset mode (canceling continuous mode) using the MCØ command. Preset moves allow you to position the motor/load in relation to a defined zero reference position (*absolute mode*—enabled with the MA1 command) or in relation to the previous stopped position (*incremental mode*—enabled with the MAØ command).

Absolute Mode Moves

The absolute mode is the 6270's default power-up mode. A preset move in the Absolute Mode (MA1) moves the motor/load the distance that you specify from the *absolute zero position*. You can set the absolute position to any value with the PSET command. When the Go Home (HOM) command is issued, the absolute position register is automatically set to zero after reaching the home position.

The direction of an absolute preset move depends upon the motor's/load's position at the beginning of the move and the position you command it to move to. For example, if the motor/load is at absolute position +12,500, and you instruct it to move to position +5,000, it will move in the negative (CCW, retract) direction a distance of 7,500 steps to reach the absolute position of +5,000.

The 6270 retains the absolute position, even while the unit is in the incremental mode. You can use the Feedback Device Position Report (TFB) command to read the absolute position.

Example	Command	Description
	> SFB3	Select LDT position feedback for axis 1
	> LDTRES432	Set axis 1 LDT resolution to 432 steps (counts) per inch
	> SCALEØ	Disable scaling
	> MA1	Set the 6270 to the absolute positioning mode
	> PSETØ	Set axis 1 current absolute position to zero
	> A5	Set axis 1 acceleration to 5 ips ²
	> V3	Set axis 1 velocity to 3 ips
	> D4	Set axis 1 move to absolute position 4
	> GO1	Initiate axis 1 move to absolute position 4
	> D8	Set axis 1 move to absolute position 8
	> GO1	Initiate axis 1 move (Since the motor/load was already at position 4, it moves 4 additional steps in the positive, or extension, direction.)
	> DØ	Set axis 1 move to absolute position zero
	> GO1	Initiate axis 1 move (Since the motor is at absolute position 8,000 the motor moves 8,000 steps in the negative, or retraction, direction.)

Incremental Mode Moves

When using the incremental mode (MAØ), a preset move moves the motor/load the specified distance from its starting position. For example, if you are using a hydraulic positioning system to move the load 1.5 inches, you would specify a preset move with a distance of +648 steps (1.5 revs @ 432 steps/inch). Every time the 6270 executes this move, the hydraulic cylinder moves 1.5 inches from its resting position. You can specify the direction of the move by using the optional sign (D+648 or D-648). Whenever you do not specify the direction (e.g., D648), the unit defaults to the positive (CW or extension) direction.

<i>Example</i>	Command	Description
>	SCALEØ	Disable scaling
>	SFB3	Select LDT position feedback for axis 1
>	LDTRES432	Set axis 1 LDT resolution to 432 steps (counts) per inch
>	MAØ	Set axis 1 to Incremental Position Mode
>	A2	Set axis 1 acceleration to 2 ips ²
>	V5	Set axis 1 velocity to 5 ips
>	D432	Set axis 1 distance to 432 steps (1 inch) in the extension direction
>	GO1	Initiate motion on axis 1 (load moves 1 inch in extension direction)
>	GO1	Repeat the move (load ends up at 2 inches from original position)
>	D-864	Set axis 1 distance to 864 steps (2 inches) in the retract direction
>	GO1	Initiate motion on axis 1 (load moves 2 inches in the retract direction and ends at its original starting position)

Continuous Mode

The Continuous Mode (MC) is useful in the following situations:

- Applications that require constant movement of the load
- Synchronize the motor to external events such as trigger input signals
- Changing the motion profile after a specified distance or after a specified time period (T command) has elapsed

Buffered vs. Immediate Commands

You can manipulate the motor movement with either buffered or immediate commands. After you issue the GO command, buffered commands are not executed unless the continuous command execution mode (COMEXC command) is enabled. Once COMEXC is enabled, buffered commands are executed in the order in which they were programmed. More information on the COMEXC mode is provided in the *Programming Guide* section of the **6000 Series Software Reference Guide**.

The command can be specified as *immediate* by placing an exclamation mark (!) in front of the command. When a command is specified as immediate, it is placed at the front of the command queue and is executed immediately.

<i>Example</i>	Command	Description
>	COMEXC1	Enable continuous command processing mode
>	MC1	Sets axis 1 mode to continuous
>	A1Ø	Sets axis 1 acceleration to 10
>	V1	Sets axis 1 velocity to 1
>	GO1	Initiates axis 1 move (Go)
>	WAIT(LVEL=1)	Wait to reach continuous velocity
>	T5	Time delay of 5 seconds
>	S1	Initiate stop of axis 1 move
>	WAIT(MOV=bØ)	Wait for motion to completely stop on axis 1
>	COMEXCØ	Disable continuous command processing mode

The motor accelerates to 1 rps/ips and continues at 1 rps/ips for 5 seconds, at which point it decelerates to a stop. While in continuous mode, motion can also be stopped if:

- You issue an immediate Stop (!S) or Kill (!K) command
- The load trips an end-of-travel limit switch
- The load trips an input configured as a kill or stop input with the INFNCi-C or INFNCi-D commands, respectively.

On-The-Fly Changes

You can change velocity and acceleration *on the fly* (while motion is in progress and in continuous mode) by issuing an immediate velocity (!V) and/or acceleration (!A) command followed by an immediate go (!GO). If the continuous command processing mode (COMEXC) is enabled, you can also make *on-the-fly* velocity and acceleration changes by using buffered commands (V and A), followed by a GO command.

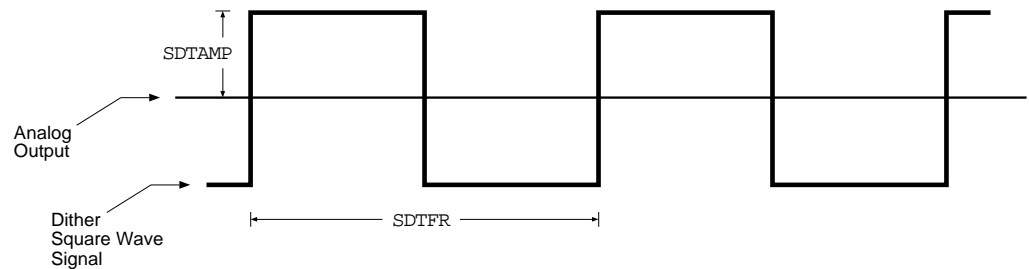
NOTE

While the axis is moving, you cannot change the parameters of some commands (such as D and HOM). This rule applies during the COMEXC mode and even if you prefix the command with an immediate command identifier (!). For more information, refer to *Changing Command Parameters During Motion* in the *Programming Guide* section of the **6000 Series Software Reference Guide**.

Example	Command	Description
	> DEF prog1	Begin definition of program prog1
	- COMEXC1	Enable continuous command processing mode
	- MC1	Set axis 1 mode to continuous
	- A10	Set axis 1 acceleration to 10
	- V1	Set axis 1 velocity to 1
	- GO1	Initiate axis 1 move (Go)
	- WAIT(1VEL=1)	Wait for motor to reach continuous velocity
	- T3	Time delay of 3 seconds
	- A50	Set axis 1 acceleration to 50
	- V10	Set axis 1 velocity to 10
	- GO1	Initiate acceleration and velocity changes on axis 1
	- T5	Time delay of 5 seconds
	- S1	Initiate stop of axis 1 move
	- WAIT(MOV=b0)	Wait for motion to completely stop on axis 1
	- COMEXC0	Disable continuous command processing mode
	- END	End definition of program prog1

Dithering Hydraulic Valves

Dither is a square-wave signal added to the analog output and is used to keep the hydraulic valve moving slightly for the purpose of reducing stiction (see illustration below). Two commands are used to select the amplitude and frequency of the dither signal—SDTAMP and SDTFR.



$$\text{SSFR (servo sampling frequency)} / \text{SDTFR} = \text{Dither Frequency (cycles/sec)}$$

The SDTAMP command selects the amplitude of the dither signal in peak volts (see illustration). The SDTFR command selects the frequency ratio of the dither.

Refer to Step 2 in the Controller Tuning Procedure in Chapter 4 for a discussion on the servo sampling rate.

The actual dither frequency is determined by the ratio of the servo sampling frequency (SSFR & INDAX settings) and the SDTFR value. For example, if the SSFR value is 4 and the INDAX value is 2 (default settings), the servo sampling rate is 2500 samples per second. Then, at SSFR4, an SDTFR value of 46 (default setting) would yield a 54.3 Hz dither frequency (2500/46 = 54.3). With an SDTFR command setting of 46, a positive voltage (SDTAMP) is added during 23 servo updates and a negative voltage is added during the next 23 servo updates.

User Interface Options

The following are the three basic user interface options for controlling the 6270:

Stand-alone operation:

After defining and storing 6270 programs with a RS-232C terminal, you can operate the 6270 as a stand-alone controller. A program stored in the 6270 may interactively prompt the user for input as part of the program (via programmable I/O, thumbwheels, an RP240, or an RS-232C terminal).

PLC interface:

The 6270's programmable I/O may be connected to most PLCs. After defining and storing 6270 programs with a RS-232C terminal, the PLC typically executes programs, loads data, and manipulates inputs to the 6270. The PLC instructs the 6270 to perform the motion segment of a total machine process.

Host computer operation:

A computer may be used to control a motion or machine process. A PC can monitor processes and orchestrate motion by sending motion commands to the 6270 or by executing motion programs already stored in the 6270. This control might come from a BASIC or C program.

Of course, you can use any one, or combination, of these options in your application. Some application examples are provided below. The sections below discuss programmable I/O (including thumbwheel and PLC interfacing), the RP240 interface, host computer interfacing, stored programs, and other aspects of the 6270 that allow the user to apply the 6270 in the variety of interface options as noted.

User Interface Option	Application Example
Stand-alone: Programmable I/O and Thumbwheel/TM8 interface	Cut-to-length: Load the stock into the machine, enter the length of the cut on the thumbwheels, and activate a programmable input switch to initiate the predefined cutting process (axis #1). When the stock is cut, a sensor activates a programmable input to stop the cutting process and the 6270 then initiates a predefined program that indexes the stock forward (axis #2) into position for the next cut.
RP240 Remote Operator Panel interface	Gainset selection: Program the RP240 to select different servo gainsets. At start-up a set of gains for cold fluid can be selected. As the fluid heats up a new set of gains may be selected.
Joystick interface	Coarse positioning: Before machining a part it must be coarsely positioned. Simulator: Use analog inputs to provide motion in flight simulator.
14-bit analog interface (6270-ANI Option only)	Injection Molding: Use for feedback from a pressure sensor and use 6270 to maintain constant, programmable force.
PLC Interface	X-Y point-to-point: A PLC controls several tools to stack and bore several steel plates at once. The 6270 is programmed to move an X-Y table in a pre-programmed sequence. The 6270 moves the load when the inputs are properly configured, signals the PLC when the load is in position, and waits for the signal to continue to the next position.
Host Computer (PC) Interface	A BASIC program example is provided later in the section labeled <i>Host Computer Operation</i> .

Programmable Inputs and Outputs

Refer to Chapter 3 for I/O connection instructions.

I/O circuit drawings and specifications are provided in Chapter 5.

There are 26 programmable inputs (includes 2 trigger inputs on the AUX connector) and 26 programmable outputs (includes 2 auxiliary outputs on the AUX connector).

All the 6270's inputs and outputs are optically isolated. 24 inputs and 24 outputs are on OPTO-22 compatible connectors for those applications that require interfacing to 120VAC I/O or to I/O with higher current requirements than the 6270 can support.

Programmable inputs and outputs are provided to allow the 6270 to detect and respond to the state of switches, thumbwheels, electronic sensors, and outputs of other equipment such as drives and PLCs. Based on the state of the inputs and outputs, read with the [IN] and [OUT] commands, the 6270 can make program flow decisions and assign values to binary variables for subsequent mathematical operations. These operations and the associated program flow, branching, and variable commands are listed below.

Operation based on I/O State	Associated Commands	See Also*
I/O state assigned to a binary variable	[IN], [OUT], VARB	Chapter 5, <i>Variables</i> section
I/O state used as a basis for comparison in conditional branching & looping statements	[IN], [OUT], IF, ELSE, NIF, REPEAT, UNTIL, WAIT, WHILE, NWHILE	<i>Program Flow Control</i> section in the 6000 Series Software Reference Guide
I/O state used as a basis for a program interrupt (GOSUB) conditional statement	ONIN	<i>Program Interrupts</i> section in the 6000 Series Software Reference Guide

* Refer also to the command descriptions in the **6000 Series Software Reference Guide**

As discussed below, you can program and check the status of each input and output with the INFNC and OUTFNC commands, respectively. To receive a binary report of the state (on or off) of the I/O, use the TIN command (inputs) or the TOUT command (outputs).

Using the INLVL and OUTLVL commands, you can define the logic levels of the 24 general-purpose inputs and outputs (including OUT-A and OUT-B) as positive or negative.

Output Functions

You can turn the 6270's 26 programmable outputs on and off with the Output (OUT or OUTALL) commands, or you can use the Output Function (OUTFNC) command to configure them to activate based on seven different situations.

The output functions are assigned with the OUTFNCi-<a>c command. The "i" represents the number of the output (the 24 general purpose outputs are outputs 1 through 24, and OUT-A and OUT-B are outputs 25 and 26). The "<a>" represents the number of the axis and is optional for the B, D, and G functions (see list below); when no axis specifier is given, the output will be activated when the condition occurs on either axis. The "c" represents the letter designator of the function (A, B, C, D, F, G or H). For example, the OUTFNC5-2D command configures output #5 to activate when axis #2 encounters a hard or soft limit.

NOTE

To activate the function of an output with the OUTFNC command, you must enable the output functions with the OUTFEN1 command.

- | | |
|---|---|
| A: Programmable Output (default function) | E: <Not Used> |
| B: Moving/Not Moving (or In Position) | F: Fault Output (indicates drive or user fault) |
| C: Program in Progress | G: Max. Allowable Position Error Exceeded |
| D: Soft or Hard Limit Encountered | H: Output on Position |

Output Status

As shown below, you can use the `OUTFNC` command to determine the current function and state (on or off) of one or all the outputs. The `TOUT` command also reports the outputs' state, but in a binary format in which the left-most bit represents output #1 and the right-most bit represents output #26.

Command	Description
> OUTFNC	Query status of all outputs; response indicating default conditions is: *OUTFNC1-A NO FUNCTION OUTPUT - STATUS OFF *OUTFNC2-A NO FUNCTION OUTPUT - STATUS OFF (response continues until all 26 outputs are reported)
> OUTFNC1	Query status of output #1; response indicating default conditions is: *OUTFNC1-A NO FUNCTION OUTPUT - STATUS OFF
> OUTFNC1-C	Change output #1 to function as a <i>Program in Progress</i> output
> OUTFNC1	Query status of output #1; response should be now be: *OUTFNC1-C PROGRAM IN PROGRESS - STATUS OFF
> TOUT	Query binary status report of all outputs; response indicating default conditions is: *TOUT0000_0000_0000_0000_0000_0000_00

Programmable Output

(OUTFNCi-A)

The default function for the outputs is *Programmable*. As such, the output is used as a standard output, turning it on or off with the `OUT` or `OUTALL` commands to affect processes external to the 6270. To view the state of the outputs, use the `TOUT` command. To use the state of the outputs as a basis for conditional branching or looping statements (`IF`, `REPEAT`, `WHILE`, etc.), use the `[OUT]` command (refer to the *Conditional Looping and Branching* section in the **6000 Series Software Reference Guide** for details).

Moving/ Not Moving (In Position)

(OUTFNCi-<a>B)

When assigned the *Moving/Not Moving* function, the output will activate when the axis is commanded to move. As soon as the move is completed, the output will change to the opposite state.

If the target zone mode is enabled (`STRGTE1`), the output will not change state until the move completion criteria set with the `STRGTD` and `STRGTV` commands has been met. (Refer to the *Target Zone* section in Chapter 4 for more details on the target zone mode.) In this manner, the *Moving/Not Moving* output functions as an *In Position* output.

Example

The following example defines output 1 and output 2 as *Programmable* outputs and output 3 as a *Moving/Not Moving* output. Before the motor moves 4 units, output 1 turns on and output 2 turns off. These outputs remain in this state until the move is completed, then output 1 turns off and output 2 turns on. While the motor/load is moving, output 3 remains on.

Command	Description
> PS	Pauses command execution until the 6270 receives a Continue (!C) command
SCALE1	Enable scaling
MCØ	Sets axis 1 to Normal mode
MAØ	Select incremental positioning mode
A1Ø	Sets axis 1 acceleration to 10
V5	Sets axis 1 velocity to 5
D4	Sets axis 1 distance to 4 units
OUTFEN1	Enable output functions
OUTFNC1-A	Sets output 1 as a programmable output
OUTFNC2-A	Sets output 2 as a programmable output
OUTFNC3-1B	Sets output 3 as a axis 1 Moving/Not Moving output
OUT1Ø	Turns output 1 on and output 2 off
GO1	Initiates axis 1 move
OUTØ1	Turns output 1 off and output 2 on
!C	Initiates command execution to resume

Program in Progress

(OUTFNCi-C)

When assigned the *Program in Progress* function, the output will activate when a program is being executed. After the program is finished, the output's state is reversed.

Limit Encountered
(OUTFNCi-<a>D)

When assigned the *Limit Encountered* function, the output will activate when a hard or soft limit has been encountered.

If a hard or soft limit is encountered, you will not be able to move the motor/load in that same direction until you clear the limit by changing direction (D) and issuing a GO command. (An alternative is to disable the limits with the LHØ command, but this is recommended only if the motor, cylinder, etc. is not coupled to the load.)

Fault Output
(OUTFNCi-F)

When assigned the *Fault Output* function, the output will activate when either the user fault input or the drive fault input (drive users only) becomes active. The user fault input is a general-purpose input defined as a user fault input with the INFNCi-F command. The drive fault input is found on the DRIVE connector, pin #5; make sure the drive fault active level (DRFLVL) is appropriate for the drive you are using.

Maximum Position Error Exceeded
(OUTFNCi-<a>G)

When assigned the *Max. Position Error Exceeded* function, the output will activate when the maximum allowable position error, as defined with the SMPER command, is exceeded.

The position error (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the feedback device (TFB). When the maximum position error is exceeded (usually due to lagging load, instability, or loss of position feedback), the 6270 shuts down the drive/valve and sets error status bit #12 (reported by the TER command if bit #12 of the ERROR command is enabled).

NOTE

If the SMPER command is set to zero (SMPERØ), the position error will not be monitored; thus, the *Maximum Position Error Exceeded* function will not be usable.

Output on Position
(OUTFNCi-H)

The *Output on Position* function for axis 1 (OUTFNC25-H) can be assigned only to output #25 (OUT-A), and the *Output on Position* function for axis 2 (OUTFNC26-H) can be assigned only to output #26 (OUT-B). The position being monitored is the currently selected feedback device. The *Output on Position* parameters are configured with the OUTPA and OUTPB commands:

- 1st data field (b): 1 enables the *output on position* function; \emptyset disables the function. If a SFB command is executed, the function is disabled.
- 2nd data field (b): 1 sets the position comparison in the 3rd data field (x) to an incremental position;
 \emptyset sets the position comparison in the 3rd data field (x) to an absolute position.
- 3rd data field (x): Represents the scalable distance with which the actual feedback device position is to be compared (distance is either incremental or absolute, depending on the setting of the 2nd data field). The feedback device used per axis depends on which one is assigned with the SFB command. *Remember that encoder feedback may not be used for axis 2.*
- 4th data field (i): Represents the time (in milliseconds) the output is to stay active. If this data field is set to \emptyset , the output will stay active for as long as the actual distance equals or exceeds the distance specified in the 3rd data field. (This is valid only for the absolute mode—2nd data field set to \emptyset)
If an incremental distance is used for comparison (2nd data field set to 1), the output activates when the feedback device position is greater than or equal to the specified distance, and stays active for the specified time.
If an absolute distance is used for comparison (2nd data field set to \emptyset), the output activates when the feedback device position is greater than or equal to the specified absolute distance, and stays active for the specified time.

NOTE

The output activates only during motion; thus, issuing a PSET command to set the absolute position counter to activate the output on position will not turn on the output until the next motion occurs.

Example	Command	Description
	> OUTFEN1	Enable programmable output functions
	> OUTFNC25-H	Set OUT-A (output #25) as <i>output on position</i> output for axis 1
	> OUTFNC26-H	Set OUT-B (output #26) as <i>output on position</i> output for axis 2
	> OUTPA1,Ø,+5Ø,5Ø	Turn on OUT-A for 50 ms when the actual position of axis 1 is greater than or equal to absolute position +50
	> OUTPB1,Ø,+2.4,2ØØ	Turn on OUT-B for 200 ms when the actual position of axis 2 is greater than or equal to absolute position +2.4

Input Functions

The input functions are assigned with the `INFNCi-<a>c` command. The "i" represents the number of the input (the 24 general purpose inputs are inputs 1 through 24, and TRG-A and TRG-B are inputs 25 and 26). The "<a>" represents the number of the axis, if required. The "c" represents the letter designator of the function (A through Q). For example, the `INFNC5-2D` command configures output #5 to function as a *stop* input, stopping motion on axis #2 when activated.

NOTE

To activate the function of an input with the `INFNC` command, you must first enable the input functions with the `INFEN1` command. Because the `INFEN1` command enables the drive fault input, you should verify the fault active level (`DRFLVL`) is set properly.

A: No Function (default)	H: Position Latch (TRG-A & TRG-B Only)
B: BCD Program Select	J: Jog+ (CW)
C: Kill	K: Jog- (CCW)
D: Stop	L: Jog Speed Select
E: Pause/Continue	P: Program Select
F: User Fault	Q: Program Security

Input Status

As shown below, you can use the `INFNC` command to determine the current function and state (on or off) of one or all the inputs. The `TIN` command also reports the inputs' state, but in a binary format in which the left-most bit represents input #1 and the right-most bit represents input #26.

Example	Command	Description
	> INFNC	Query status of all inputs; response indicating default conditions is: *INFNC1-A NO FUNCTION INPUT - STATUS OFF *INFNC2-A NO FUNCTION INPUT - STATUS OFF (response continues until all 26 inputs are reported)
	> INFNC1	Query status of input #1; response indicating default conditions is: *INFNC1-A NO FUNCTION INPUT - STATUS OFF
	> INFNC1-D	Change input #1 to function as a Stop input
	> INFNC1	Query status of input #1; response should be now be: *INFNC1-D STOP INPUT - STATUS OFF
	> TIN	Query binary status report of all inputs; response indicating default conditions is: *TINØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØØØ_ØØ

Input Debounce Time

Using the Input Debounce Time (`INDEB`) command, you can change the input debounce time for all 24 general-purpose inputs (one debounce time for all 24), or you can assign a unique debounce time to each of the 2 trigger inputs.

General-Purpose Input Debounce: The input debounce time for the 24 general-purpose inputs is the period of time that the input must be held in a certain state before the 6270 recognizes it. This directly affects the rate at which the inputs can change state and be recognized.

Trigger Input Debounce: For trigger inputs, the debounce time is the time required between a trigger's initial active transition and its secondary active transition. This allows rapid recognition of a trigger, but prevents subsequent bouncing of the input from causing a false position capture.

The INDEB command syntax is INDEB<i>,<i>. The first <i> is the input number and the second <i> is the debounce time **in even increments** of milliseconds (ms). The debounce time range is 1 - 250 ms. The default debounce time is 4 ms for the 24 general-purpose inputs, and 24 ms for the 2 trigger inputs (TRG-A & TRG-B). If the first <i> is in the range 1 - 24, the specified debounce time is assigned to all 24 general-purpose inputs. If the first <i> is 25 or 26, the specified debounce is assigned only to the specified trigger input.

For example, the INDEB5,6 command assigns a debounce time of 6 ms to all 24 general-purpose inputs. The INDEB26,12 command assigns a debounce time of 12 ms only to input #26, which is trigger B (TRG-B).

No Function
(INFNCi-A)

When an input is defined as a *No Function* input (default function), the input is used as a standard input. You can then use this input to synchronize or trigger program events. To view the current state of the inputs, use the TIN command. To use the state of the inputs as a basis for conditional branching or looping statements (IF, REPEAT, WHILE, etc.), use the [IN] command (refer to the **Conditional Looping and Branching** section in the **6000 Series Software Reference Guide** for details).

Example	Command	Description
	> DEF prog1	Begin definition of program prog1
	- INFEN1	Enable input functions
	- INFNC1-A	No function for input 1
	- INFNC2-A	No function for input 2
	- INFNC3-D	Input 3 is a stop input
	- A1Ø	Set acceleration
	- V1Ø	Set velocity
	- D5	Set distance
	- WAIT(IN=b1XX)	Wait for input 1
	- GO1	Initiate motion
	- IF(IN=bX1)	If input 2
	- TLDT	Transfer LDT position
	- NIF	End IF statement
	- END	End prog1
	> RUN prog1	Initiate program prog1

BCD Program Select
(INFNCi-B)

Inputs can be defined as *BCD program select* inputs.

This allows you to execute defined programs (DEF command) by activating the program select inputs. Program select inputs are assigned BCD weights. The table to the right shows the BCD weights of the 6270's inputs when inputs 1- 8 are configured as program select inputs. The inputs are weighted with the least weight on the smallest numbered input.

Input #	BCD Weight
Input 1	1
Input 2	2
Input 3	4
Input 4	8
Input 5	10
Input 6	20
Input 7	40
Input 8	80

If inputs 6, 9, 10 and 13 are selected instead of inputs 5, 6, 7 and 8, then the weights would be as follows:

Input #6	=	10
Input #9	=	20
Input #10	=	40
Input #13	=	80

Since 100 programs can be defined, a maximum of 8 inputs are required to select all possible programs. Up to 400 programs may be defined if you have the -M (extended memory) option.

The program number is determined by the order in which the program was downloaded to the 6270. The program number can be obtained through the TDIR command.

If the inputs are configured as in the above table, activating inputs 2 and 3 will execute program #6. Activating inputs 1, 4, and 6 will execute program #29.

To execute programs using the program select lines, enable the INSELP command. Once enabled, the 6270 will continuously scan the input lines and execute the program selected by the active program select lines. To disable scanning for program select inputs, enter !INSELPØ or place INSELPØ in a program that can be selected.

Once enabled (INSELP1), the 6270 will run the program number that the active program select inputs and their respective BCD weights represent. After executing and completing the selected program, the 6270 will scan the inputs again. If a program is selected that has not been defined, no program will be executed.

The INSELP command also determines how long the program select input must be maintained before the 6270 executes the program. This delay is referred to as *debounce time* (but is not affected by the INDEB setting). The following examples demonstrate how to select programs via inputs.

Example	Command	Definition
	> RESET	Return 6270 to power-up conditions
	> ERASE	Erase all programs
	> DEF prog1	Begin definition of program prog1
	- TLDT	Transfer position of LDTs
	- END	End program
	> DEF prog2	Begin definition of program prog2
	- TREV	Transfer software revision
	- END	End program
	> DEF prog3	Begin definition of program prog3
	- TSTAT	Transfer statistics
	- END	End program
	> INFNC1-B	Input 1 is a BCD program select input
	> INFNC2-B	Input 2 is a BCD program select input
	> INFEN1	Enable input functions
	> INSELP1,50	Enable scanning of inputs, levels must be maintained for 50ms

You can now execute programs by making a contact closure from an input to ground to activate the input.

- Activate input 1 to execute program #1
- Activate input 2 to execute program #2
- Activate input 1 & 2 to execute program #3

Kill

(INFNCi-C)

An input defined as a *Kill* input will stop motion at the rate set with the hard limit (LHAD & LHADA) commands. The program currently in progress will also be terminated, the commands currently in the command buffer will be eliminated, and the drive will be left in the enabled state (DRIVE1).

*Disabling the Drive
or Valve on a Kill*

If your application requires you to *disable* (*shut down* or *de-energize*) the valve or drive in a Kill situation, set the 6270 to the *Disable Drive on Kill* mode with the KDRIVE1 command. In this mode, a kill command or kill input will shut down the valve or drive immediately. If you are using a drive, the shutdown outputs are activated and the motor will then be allowed to *free wheel* (without control from the drive) to a stop. If you are using a hydraulic valve, the kill will set the command output to zero, causing the valve to return immediately to the neutral (*null*) position and hold the load at that position. To re-enable the valve or drive, issue the DRIVE11 command.

Stop

(INFNCi-D)

An input defined as a *Stop* input will stop motion on any one or all axes. Deceleration is controlled by the programmed (AD/ADA) deceleration ramp. After the Stop input is received, further program execution is dependent upon the COMEXS command setting:

COMEXS0: Upon receiving a stop input, program execution will be terminated and every command in the buffer will be discarded.

COMEXS1: Upon receiving a stop input, program execution will pause, and all commands following the command currently being executed will remain in the command buffer (*but the move in progress will not be saved*).

You can resume program execution (but not the move in progress) by issuing an immediate Continue (!C) command or by activating a pause/resume input (i.e., a general-purpose input configured as a pause/continue input with the INFNCi-E command—see below). *You cannot resume program execution while the move in progress is decelerating.*

COMEXS2: Upon receiving a stop input, program execution will be terminated, but the INSELP value is retained. This allows external program selection, via inputs defined with the INFNCi-B or INFNCi-aP commands, to continue.

Pause/Continue
(INFNCi-E)

An input defined as a *Pause/Continue* input will affect motion and program execution depending on the COMEXR command setting, as described below. In both cases, when the input is activated, the current command being processed will be allowed to finish executing before the program is paused.

COMEXR0: Upon receiving a pause input, only program execution will be paused; any motion in progress will continue to its predetermined destination. Releasing the pause input or issuing a !C command will resume program execution.

COMEXR1: Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. Releasing the pause input or issuing a !C command will resume motion and program execution. *You cannot resume program execution while the move in progress is decelerating.*

User Fault
(INFNCi-F)

An input defined as a *User Fault* input will set error status bit 7 (reported by TER and [ER]), and act as a Kill (K) command. Once this bit has been set, the error program (ERRORP) will be initiated if the specific error condition was enabled (ERRORxxxx xx1). Within the error program, a response to the fault condition can be initiated.

Position Latch
(INFNCi-H)

Certain applications (such as coordinate measurement machines) require latching the current position upon receiving an input. *This function can only be assigned to the two trigger inputs (inputs 25 & 26).*

NOTE

When configured as position latch inputs, the triggers cannot be affected by the input enable (INEN) command.

Trigger input specs and circuit drawings provided in Chapter 6.

When a trigger input defined as a *Position Latch* input is activated, the commanded position and the position of all feedback devices on both axes are captured at one time. The position information is stored in registers and is available at the next system update through the use of transfer and assignment/comparison commands (see table below).

Captured Information	Transfer Command	Assignment/Comparison Command
Commanded Position	TPCC	PCC
LDT Position	TPCL	PCL
Encoder Position	TPCE	PCE
ANI Position	TPCA	PCA
ANI Voltage	TCA	CA

Position Latch Accuracy is Greater for Selected Feedback Source (SFB)

If you are capturing the position/value of a feedback source (LDT, encoder, or ANI) that is currently selected with the SFB command, the captured position is interpolated from the last sampled position and velocity of the feedback device and the time elapsed since the last sample. The position sample rate is determined by the SSFR and INDAX command settings (system update rate). The position capture accuracy is $\pm 50 \mu s * \text{current velocity}$.

If you are capturing the position of a device that is not selected with the SFB command, the last sampled position is simply stored as the captured position.

Exceptions:

- Regardless of the SFB selection, if TRG-A is used to capture the encoder position on axis #1, the position is latched in hardware within ± 1 encoder count (at max. encoder frequency) after the input is activated. Encoder position capture with TRG-B is affected by the SFB selection as noted above.
- The captured commanded position is always interpolated from the last sampled position (of the feedback device selected with the SFB command) and position error, and the time elapsed since the last sample.

☞
Input debounce time
programmable with
INDEB command.

Each *position latch* input has a 24-ms debounce time. Therefore, the maximum rate that the input can capture positions is 40 times per second. However, if your application requires a shorter debounce time, you can change it with the INDEB command (refer to the *Input Debounce Time* section provided earlier in this chapter).

System status bits #25 and #26, reported with the TSS and SS commands, are set to 1 when the position has been captured on the respective trigger input (TRG-A and TRG-B, respectively). As soon as the captured information is transferred (TPCC, TPCL, TPCE, TPCA, or TCA) or assigned/compared (PCC, PCL, PCE, PCA, or CA), the respective system status bit is cleared, but the information is still available from the register until it is overwritten by a new latch from the trigger input.

The captured *position* values (commanded, LDT, encoder, and ANI) are affected by the current state of distance scaling (SCLD), position offset (PSET), feedback polarity (LDTPOL, ENCPOL, or ANIPOL), and commanded direction polarity (CMDDIR). However, the captured ANI voltage value (TCA and CA) is not affected and is reported only in volts.

Jogging the Motor

(INFNCi-aJ)
(INFNCi-aK)
(INFNCi-aL)

In some applications, you may want to manually move (*jog*) the load. You can configure the jog function with the INFNC command.

You must define the jogging velocity with the Jog Velocity High (JOGVH) and Jog Velocity Low (JOGVL) commands. The acceleration and deceleration of the JOG move can be configured using JOGA, and JOGAD respectively. (If you are using S-curve profiles, you must also specify JOGAA and JOGADA.)

You can define three different inputs per axis for jogging: extend CW (positive direction) Jog input (INFNCi-aJ), retract CCW (negative direction) Jog Input (INFNCi-aK), and Jog Speed Select High/Low (INFNCi-aL). You must also enable the jogging feature with the JOG command.

Once you set up these parameters, you can attach a switch to the jog inputs that you defined and perform jogging. The following example shows how you can define a program to set up jogging.

Step 1	Command	Description
	> DEF prog1	Begin definition of program prog1
	- JOGA25	Set jog acceleration
	- JOGAD25	Set jog deceleration
	- JOGVL.5	Sets low-speed jog velocity
	- JOGVH5	Sets high-speed jog velocity
	- INFEN1	Enable input functions
	- INFNC1-1J	Sets input 1 as a CW jog input
	- INFNC2-1K	Sets input 2 as a CCW jog input
	- INFNC3-1L	Sets input 3 as a speed-select input
	- JOG1	Enables Jog function for axis 1
	- END	End program definition

Step 2 Activate input 1 to move the load in the positive direction at a velocity of 0.5 (until input 1 is released).

Step 3 Activate input 2 to move the load in the negative direction at a velocity of 0.5 (until input 2 is released).

Step 4 Activate input 3 to switch to high-speed jogging.

Step 5 Repeat steps 2 and 3 to perform high-speed jogging.

One-to-One Program Select (INFNCi-aP)

Inputs can be defined as *One-to-One Program Select* inputs (INFNCi-aP). This allows programs defined by the DEF command to be executed by activating an input. Different from BCD Program Select inputs, One-to-One Program Select inputs correspond directly to a specific program number. The program number is determined by the order in which the program was downloaded to the 6270. The program number can be obtained through the TDIR command.

To execute programs using the program select lines, enable the `INSELP` command for one-to-one program selection. Once enabled, the 6270 will continuously scan the input lines and execute the program selected by the active program select line. To disable scanning of the program select lines, enter `!INSELPØ`, or place `INSELPØ` in a program that can be selected.

Command	Description
> RESET	Return 6270 to power-up conditions
> ERASE	Erase all programs
> DEF proga	Begin definition of program proga
- TLDT	Transfer position of LDT
- END	End program
> DEF progb	Begin definition of program progb
- TREV	Transfer software revision
- END	End program
> DEF progc	Begin definition of program progc
- TSTAT	Transfer statistics
- END	End program
> INFNC1-1P	Input 1 will select proga
> INFNC2-2P	Input 2 will select progb
> INFNC3-3P	Input 3 will select progc
> INFEN1	Enable input functions
> INSELP2,5Ø	Enable scanning of inputs

You can now execute programs by making a contact closure from an input to ground to activate the input:

- Activate input 1 to execute program #1 (proga)
- Activate input 2 to execute program #2 (progb)
- Activate input 3 to execute program #3 (progc)

Program Security (INFNCi-Q)

Issuing the `INFNCi-Q` command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input.

The program security feature denies you access to the `DEF`, `DEL`, `ERASE`, `MEMORY`, and `INFNC` commands until you activate the program access input. Being denied access to these commands effectively restricts altering the user memory allocation. If you try to use these commands when program security is active (program access input is not activated), you will receive the error message `*ACCESS DENIED`. *The `INFNCi-Q` command is not saved in battery-backed RAM, so you may want to put it in the start-up program (STARTP).*

For example, once you issue the `INFNC22-Q` command, input #22 is assigned the program access function and access to the `DEF`, `DEL`, `ERASE`, `MEMORY`, and `INFNC` commands will be denied until you activate input #22.

Thumbwheel Interface

You can connect the 6270's programmable I/O to a bank of thumbwheel switches to allow operator selection of motion or machine control parameters.

The 6270 allows two methods for thumbwheel use. One method uses Compumotor's TM8 thumbwheel module or IM32 input module. The other allows you to wire your own thumbwheels.

The TM8 requires a multiplexed BCD input scheme to read thumbwheel data. Therefore, a decode circuit must be used for thumbwheels. Compumotor recommends that you purchase Compumotor's TM8 module if you desire to use a thumbwheel interface. The TM8 contains the decode logic; therefore, only wiring is needed.

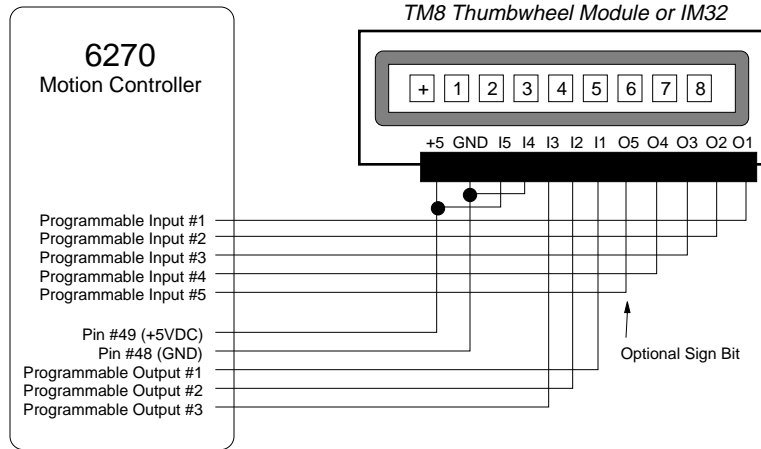
The 6270 commands that allow for thumbwheel data entry are:

Command	Description
INSTW	Establish thumbwheel data inputs (TM8)
OUTTW	Establish thumbwheel data outputs (TM8)
TW	Read thumbwheels or PLC inputs
INPLC	Establish PLC data inputs (Other thumbwheel module)
OUTPLC	Establish PLC data outputs (Other thumbwheel module)

Using the TM8 & IM32 Modules

To use Compumotor's TM8 or IM32 Modules, follow the procedures below.

Step 1 Wire your TM8 or IM32 module to the 6270 as shown below.



Step 2 Configure your 6270 as follows:

Command	Description
> OUTTW1,1-3,0,10	Configure thumbwheel output set 1 as follows: outputs 1-3 are strobe outputs, 10 ms strobe time per digit read. The minimum strobe time recommended for the TM8 module is 10 ms.
> INSTW1,1-4,5	Configure thumbwheel input set 1 as follows: inputs 1-4 are data inputs, input 5 is a sign input.
> INLVL00000	Inputs 1-5 configured active low

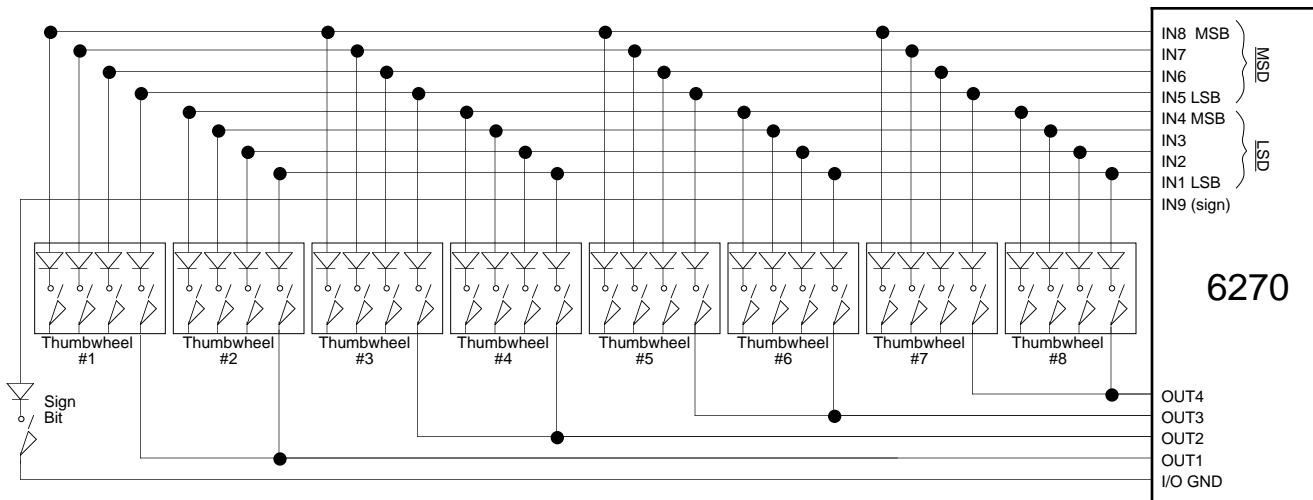
Step 3 Set the thumbwheel digits on your TM8 module to **+12345678**. To verify that you have wired your TM8 module(s) correctly and configured your 6270 I/O properly, enter the following commands:

Command	Description
> VAR1=TW1	Assign data from all 8 thumbwheel digits to VAR1
> VAR1	Displays the variable—*VAR1=+0.12345678. If you do not receive the response shown, return to step 1 and retry.

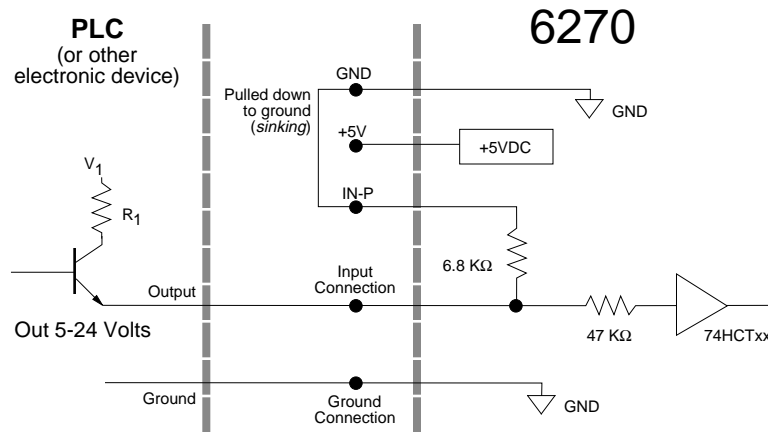
Using your own Thumbwheel Module

As an alternative to Compumotor's TM8 Module, you can use your own thumbwheels. The 6270's programming language allows direct input of BCD thumbwheel data via the programmable inputs. Use the following steps to set up and read the thumbwheel interface. Refer to the **6000 Series Software Reference Guide** for descriptions of the commands used below.

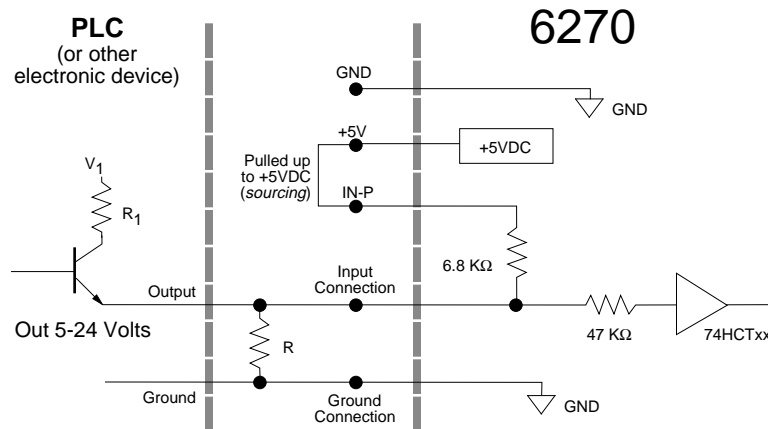
Step 1 Wire your thumbwheels according to the following schematic.



Connecting to a Sourcing Output Device



If you will be connecting to a combination of sourcing and sinking outputs, connect IN-P to +5V to accommodate sinking output devices. Then for each individual input connected to a sourcing output, wire an external resistor between the 6270's input terminal and ground (see illustration below). The resistor provides a path for current to flow from the device when the output is active.



Typical value for R = 450Ω (assuming $R_1 = 0$)

Note: The value of R may vary depending on the value of R_1 and V_1

Joystick Interface

The 6270 has three 8-bit analog input channels (CH1 - CH3). The analog inputs are configured as three discrete single-ended inputs, with an input range of 0.0V to 2.5V. These inputs can be used to control an axis with a joystick. The voltage value on the analog inputs can be read using the ANV or TANV commands. *Refer to Chapter 3 for connection procedures.*

The Daedal JS6000 joystick is compatible with the 6270. To order the JS6000, contact Daedal at (800) 245-6903 or contact your local distributor.

Joystick control can be achieved by simply connecting a joystick potentiometer to one of the analog inputs. Joystick operation is enabled with the JOY1 command.

Travel limitations in potentiometers and voltage drops along the cables may make it impossible to achieve the full 0.0V to 2.5V range at the joystick input. Therefore, you must configure the 6270 to optimize the joystick's usable voltage range. This configuration will affect the *velocity resolution*. The velocity resolution is determined by the following equation:

$$\frac{\text{maximum velocity set with the JOYVH or the JOYVL command}}{\text{voltage range between the joystick's no-velocity region (center deadband) and its maximum-velocity region (end deadband)}}$$

To establish the velocity resolution, you must define the full-scale velocity and the usable voltage.

Define Full-Scale Velocity

You must define the full-scale velocity for your application with the JOYVH and JOYVL commands. Both commands establish the maximum velocity that can be obtained per axis by deflecting the potentiometer fully CW or fully CCW. The JOYVH command establishes the *high* velocity range (selected if the joystick select input is high—sinking current). The JOYVL command establishes the *low* velocity range (selected if the joystick select input is low—not sinking current).

The JOYAXL and JOYAXH commands define which analog channels are to be used with which joystick axes when the joystick select input is low or high, respectively.

Define Usable Voltage

Use the commands described in the following table to establish the joystick's usable velocity range.

The analog-to-digital converter is an 8-bit converter with a voltage range of 0.0V to 2.5V. With 8 bits to represent this range, there are 256 distinct voltage levels from 0.0V to 2.5V. 1 bit represents 2.5/256 or 0.00976 volts/bit.

Command	Name	Purpose
JOYEDB	End Deadband	This command defines voltage levels (shy of the 0.0V and 2.5V endpoints) at which maximum velocity occurs. Specifying an end deadband effectively decreases the voltage range of the analog input to compensate for joysticks that cannot reach the 0.0V and 2.5V endpoints.
JOYCTR (or JOYZ)	Center Voltage*	This command defines the voltage level for the center of the analog input range (the point at which zero velocity will result). As an alternative, you can use the JOYZ command, which reads the current voltage on the joystick input and considers it the center voltage. You can check the center voltage by typing in JOYCTR[cr].**
JOYCDB	Center Deadband	This command defines the voltage range on each side of the center voltage in which no motion will occur (allows for minor drift or variation in the joystick center position without causing motion).

* Because the center voltage can be set to a value other than the exact center of the potentiometer's voltage range, and because there could be two different velocity resolutions, the CW velocity resolution may be different than CCW velocity resolution.

** Because of finite voltage increments, the 6270 may not report back exactly what you specified with the JOYCTR command.

Joystick Control Inputs

The table below lists the 6270's four joystick control inputs and their active levels and what the active levels affect.

Joystick Input Bit	Active Level	Effect of Active Level
Axis select	0	Selects JOYAXL
	1	Selects JOYAXH
Velocity select	0	Selects JOYVL
	1	Selects JOYVH
Joystick release	0	Exit Joystick Mode (equiv. to JOY00 command). To use the joystick again, issue the JOY11 command.
	1	Stay in Joystick Mode
Joystick trigger (general purpose)	0 or 1	Interpreted by user program (status is reported with the TINO and INO commands)
Joystick auxiliary (general purpose)	0 or 1	Interpreted by user program (status is reported with the TINO and INO commands)

Typical Applications

A typical joystick application is two-axis, in which a high velocity range is required to move to a region, then a low velocity range is required for a fine search. After the search is completed it is necessary to record the load positions, then move to the next region. The joystick trigger input can be used to indicate that the position should be read. The joystick release is used to exit the joystick mode and continue with the motion program.

Joystick Set Up Example

The following table describes the requirements of the application (rotary) described above, and how the 6270 is configured to satisfy those requirements. The resulting joystick voltage configuration is illustrated below. Given: one analog input channel is used for each axis.

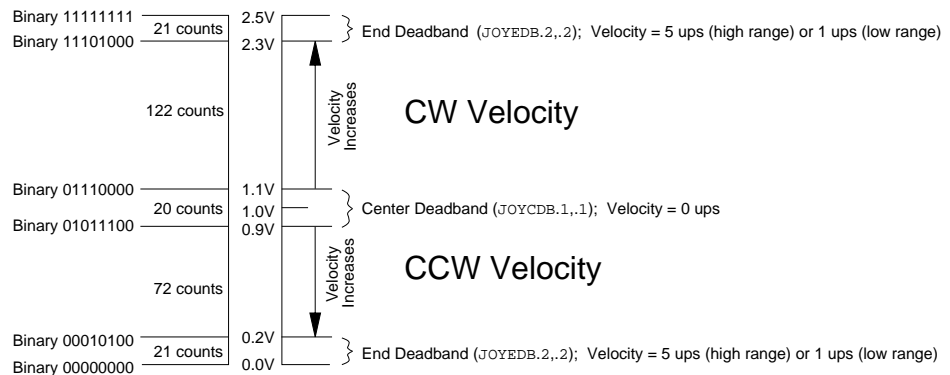
Requirement	Configuration
Set max. high-range velocity to 5 units/sec (on both axes)	Type in the JOYVH5, 5 command
Set max. low-range velocity to 1 units/sec (on both axes)	Type in the JOYVL1, 1 command
No velocity when voltage is at 1.0V	Set center voltage with JOYCTR1, 1, command, or set voltage level at both analog inputs to 1.0V and type in JOYZ11
Joystick cannot reliably rest at 1.0V, but can rest within $\pm 0.1V$ of 1.0V	Set center deadband of 0.1V with JOYCDB.1, .1 command (0.1V is the system default)
Joystick can only produce maximum of 2.3V and minimum of 0.2V	Set end deadband to get max. velocity at 2.3V or 0.2V with the JOYEDB.2, .2 command. Voltage range: CW = 1.1V to 2.3V (1.2V total) CCW = 0.9V to 0.2V (0.7V total) Voltage resolution: see below

The high-range velocity resolutions (at 5 units/sec max.) are calculated as follows:

$$\text{CW: } \frac{5 \text{ units/sec (ups)}}{\text{voltage range of } 1.2V \text{ (122 counts)}} = 0.041 \text{ ups/count}; \text{ CCW: } \frac{5 \text{ ups}}{\text{voltage range of } 0.7V \text{ (72 counts)}} = 0.069 \text{ ups/count}$$

The low-range velocity resolutions (at 1 units/sec max.) are calculated as follows:

$$\text{CW: } \frac{1 \text{ ups}}{\text{voltage range of } 1.2V \text{ (122 counts)}} = 0.008 \text{ ups/count}; \text{ CCW: } \frac{1 \text{ ups}}{\text{voltage range of } 0.7V \text{ (72 counts)}} = 0.014 \text{ ups/count}$$



Analog Voltage Override

Before you actually wire the analog inputs, you can simulate their activation in software by using the ANVO command. For instance, ANVO1.2, 1.6, 1.8 overrides the hardware analog input channels—1.2V on channel 1, 1.6V on channel 2, and 1.8V on channel 3. The ANVO values are used in any command or function that references the analog input channels, but only those channels for which ANVOEN is set to 1 (e.g., Given ANVOEN011, the ANVO values 1.6V and 1.8V are referenced for analog channels 2 and 3 only); otherwise, the voltage is read from the analog input.

6270-ANI Option Offers 14-Bit Analog Input

The 6270-ANI option offers two $\pm 10V$, 14-bit analog inputs (one ANI terminal found on each of the DRIVE connectors). Each input has an anti-aliasing filter and is sampled at the servo sample rate (set with the SSFR command). The voltage value of the ANI inputs can be transferred to the terminal with the TANI command, or used in an assignment or comparison operation with the [ANI] command (e.g., IF(1ANI<2.4)). The position value of the ANI inputs can be transferred to the terminal with the TPANI command, or used in an assignment or comparison operation with the [PANI] command (e.g., WAIT(1PANI>421)).

Three common applications of the ANI inputs are:

- Position command to the control loop
- Position feedback to the control loop
- A force or torque feedback signal

Programming Example

The following programming example will read the analog inputs into the 6270 and set the commanded analog output of each axis to that value. If you have a torque drive, this provides open-loop torque control.

Command	Description
> SGP0,0	Turn off servo proportional feedback gain
> SGI0,0	Turn off servo integral feedback gain
> SGV0,0	Turn off servo velocity feedback gain
> SGAF0,0	Turn off servo acceleration feedforward gain
> SGVF0,0	Turn off servo velocity feedforward gain
> SOFFS0,0	Set the offset to zero. The analog output will be 0 volts.
> L	Enter an infinite loop
VAR1=1ANI	Read value of ANI analog input #1 into variable #1
VAR2=2ANI	Read value of ANI analog input #2 into variable #2
SOFFS(VAR1),(VAR2)	Assign the voltages from ANI analog inputs #1 & #2 to the analog output for axes #1 & #2, respectively
T.01	Set time delay to 10 milliseconds
LN	End loop

4-20 mA Feedback

The analog inputs can be used to monitor a process using 4-20 mA feedback. This is accomplished by simply attaching a resistor from the ANI input to AGND (e.g., a 500 Ω , 1% resistor would facilitate a 2-10V input). In this way, the current is converted to a voltage that can be monitored with the ANI input. The PSET command can be used to set the input readings to user coordinates.

ANI as a Feedback Device

The ANI analog inputs, when selected as a feedback source with the SFB command, is assumed to provide position information. With this feedback it is possible to solve applications that require positioning to a voltage, rather than positioning to a known position. Some example applications are as follows:

- Using a potentiometer as feedback (mechanical motion is mimicked by the 6270)
- Maintaining a force while position changes due to fluid evacuating a chamber
- Opening or closing a valve as another process changes

RP240 Remote Operator Panel Interface

The 6270 is directly compatible with the Compumotor RP240 Remote Operator Panel. This section describes how to use the 6270 with the RP240. RP240 connections are demonstrated in Chapter 3, *Installation*.

NOTE

Refer to the **Model RP240 User Guide** (p/n 88-012156-01) for user information on *Hardware Specifications, Mounting Guidelines, Environmental Considerations, and Troubleshooting*.

Operator Interface Features

The RP240 is used as the 6270's *operator interface*, not a program entry terminal. As an operator interface, the RP240 offers the following features:

- Displays text and variables
- 8 LEDs can be used as programmable status lights
- Operator data entry of variables: read data from RP240 into variables and command value substitutions (see table in Appendix C of software guide)

Typically the user creates a program in the 6270 to control the RP240 display and RP240 LEDs. The program can read data and make variable assignments via the RP240's keypad and function keys.

The 6000 Series software commands for the RP240 are listed below. Detailed descriptions are provided in the **6000 Series Software Reference Guide** and the **Model RP240 User Guide**. The example below demonstrates the majority of the 6000 Series commands for the RP240.

```

DCLEAR..... Clear The RP240 Display
DJOG..... Enter RP240 Jog Mode
DLED..... Turn RP240 LEDs On/Off
DPASS ..... Change RP240 Password
DPCUR ..... Position The Cursor On The RP240 Display
[DREAD] ..... Read RP240 Data
[DREADF] ..... Read RP240 Function Key
DREADI..... RP240 Data Read Immediate Mode
DVAR..... Display Variable On RP240
DWRITE..... Display Text On The RP240 Display

```

NOTE: If you are not using an RP240, the **RP240** connector can be used as a one-way (transmit only) RS-232C port. The **DWRITE** command can then be used to send text to a remote RS-232C device, such as an SX.

Example	Command	Description
>	DEF panel1	Define program panel1
-	REPEAT	Start of repeat loop
-	DCLEARØ	Clear display
-	DWRITE"SELECT A FUNCTION KEY"	Display text "SELECT A FUNCTION KEY"
-	DPCUR2,2	Move cursor to line 2 column 2
-	DWRITE"DIST"	Display text "DIST"
-	DPCUR2,9	Move cursor to line 2 column 9
-	DWRITE"GO"	Display text "GO"
-	DPCUR2,35	Move cursor to line 2 column 35
-	DWRITE"EXIT"	Display text "EXIT"
-	VAR1 = DREADF	Input a function key
-	IF (VAR1=1)	If function key #1 hit
-	GOSUB panel2	GOSUB program panel2
-	ELSE	Else
-	IF (VAR1=2)	If function key #2 hit
-	DLED1	Turn on LED #1
-	GO1	Start motion on axis #1
-	DLEDØ	Turn off LED #1
-	NIF	End of IF (VAR1=2)
-	NIF	End of IF (VAR1=1)
-	UNTIL (VAR1=6)	Repeat until VAR1=6 (function key 6)
-	DCLEARØ	Clear display
-	DWRITE"LAST FUNCTION KEY = F"	Display text "LAST FUNCTION KEY = F"
-	DVAR1,1,Ø,Ø	Display variable 1
-	END	End of panel1
>		
>	DEF panel2	Define prog panel2
-	DCLEARØ	Clear display
-	DWRITE"ENTER DISTANCE"	Display text "ENTER DISTANCE"
-	D(DREAD)	Enter distance number from RP240
-	END	End of panel2

Using the Default Mode

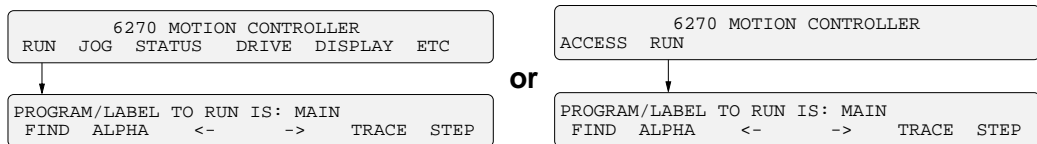
On power-up, the 6270 will automatically default to a mode in which it controls the RP240 with the menu-driven functions listed below.

- Run a stored program (RUN, STOP, PAUSE and CONTINUE functions)
- Jog (open loop and closed loop)
- Display the status of:
 - System (TSS)
 - Axis (TAS)
 - I/O (TIN and TOUT)
 - Limits (TLIM) and Enable input (TINO bit #6)
 - Joystick analog channel voltage values (TANV) and input states (TINO)
 - Position: Encoder (TPE), LDT (TLDT), ANI volts (TANI), Commanded (TPC), Current feedback device (TFB), and Position error (TPER)
 - Firmware revision levels for 6270, DSP and RP240 (TREV)
- Access RP240 menu functions with a security password
- RESET the 6270
- View and edit tuning gains and gain sets
- View numeric (VAR) and string (VARS) variables, and edit numeric variables

The flow chart on the next page illustrates the RP240's menu structure in the default operating mode (when no 6270 user program is controlling the RP240). Press the *Menu Recall* key to back up to the previous screen. The menu functions are described in detail below.

To disable these menus, the start-up program (the program assigned with the STARTP command) must contain the DCLEARØ command.

Running a Stored Program



After accessing the RUN menu, press **F1** to “find” the names of the programs stored in the 6270's memory; pressing **F1** repeatedly displays subsequent programs in the order in which they were stored in BBRAM. To execute the program, press the **ENTER** key.

To type in a program name at the location of the cursor, first select alpha or numeric characters with the **F2** function key (characters will be alpha if an asterisk appears to the right of ALPHA, or numeric if no asterisk appears). If alpha, press the up (2) or down (8) keys to move through the alphabet, if numeric, press the desired number key. Press **F3** to move the cursor to the left, or **F4** to move the cursor to the right.

Only user programs defined with DEF and END may be executed from this menu. Contouring paths cannot be executed from this menu; they must be executed from the terminal emulator with the PRUN command, or you can place the PRUN (name of path) command in a user program and then execute that program from this menu.

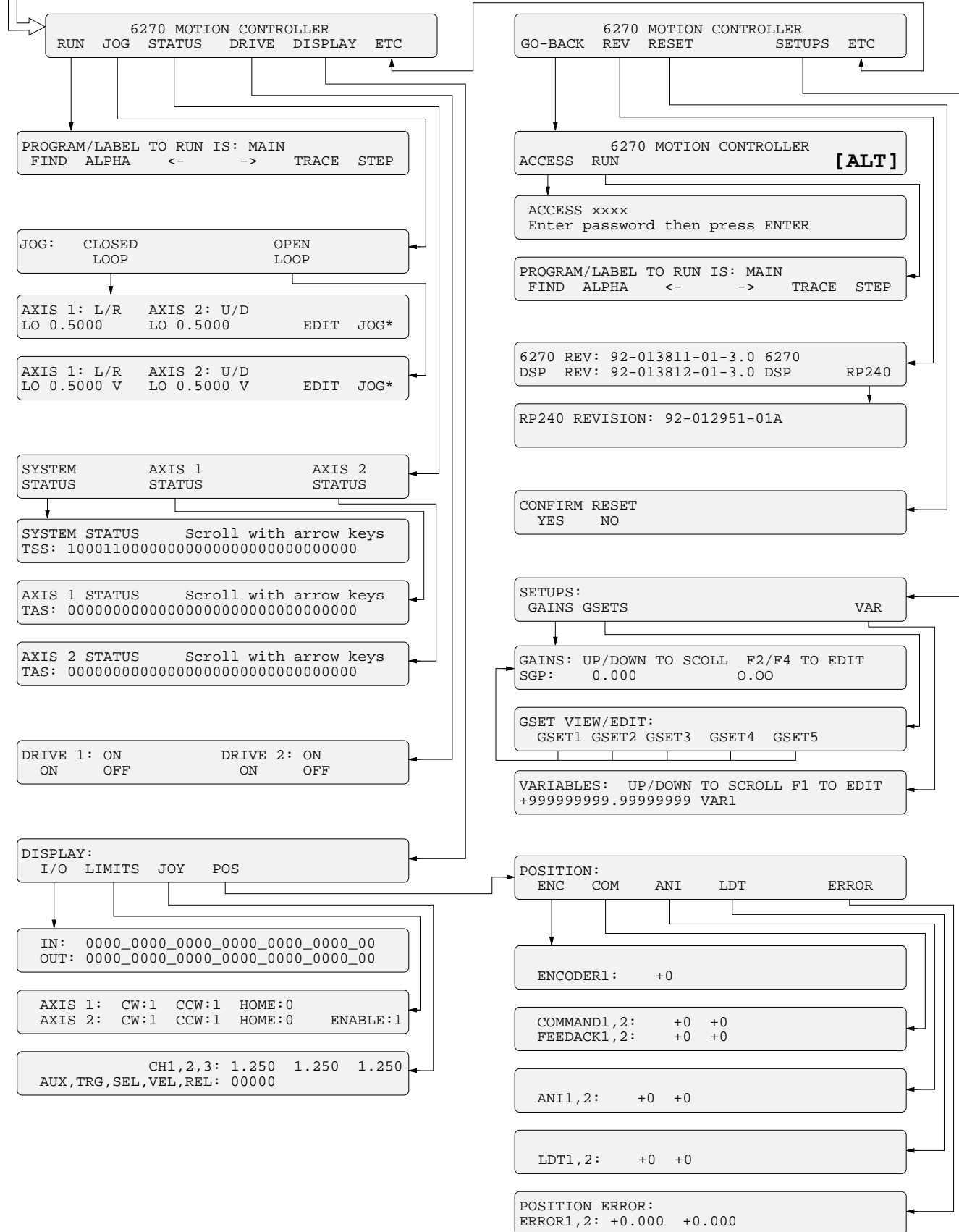
When a program is RUN and TRACE is selected (TRACE*), the RP240 display will trace all program commands as they are executed. This is different from the TRACE command in that the trace output goes to the RP240 display, not to a terminal via the RS-232C port.

HINT: If you wish to display each command as it is executed, select STEP and TRACE and press the ENTER key to step through the program.

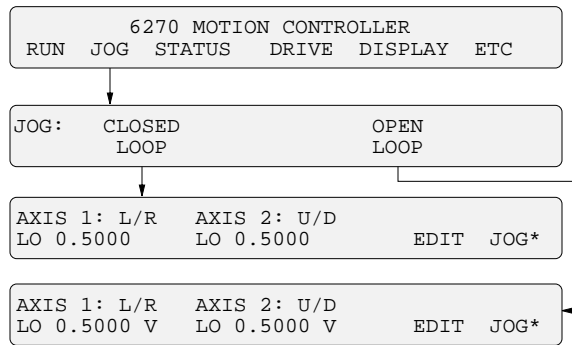
When a program is RUN and STEP is selected, step mode has been entered. This is similar to the STEP command, but when selected from the RUN menu the step mode allows single stepping by pressing the **ENTER** key. Both RP240 trace mode and step mode are exited when program execution is terminated.

Default Power-up Menu

- If you change the password (default is "6270") with the DPASS command, the menu marked with **[ALT]** becomes the power-up menu.
- Arrows indicate the menu path when you press the corresponding function key below the menu item.
- To back up to the previous menu item, press the *MENU RECALL* button.



Jogging



You can jog individual axes by pressing the RP240 arrow keys. Pressing an arrow key on the numeric keypad will start motion and releasing the arrow key will stop motion. The left and right arrow keys correspond to axis #1 negative and positive direction, respectively. The up and down arrows keys are for axis #2 negative and positive direction, respectively.

The HI and LO values in the closed-loop jog menu represent the velocity in units of inches/sec (if scaling is enabled, the value is multiplied by the programmed SCLV value). The HI and LO values in the open-loop jog menu represent volts.

To edit the jog velocity/voltage values:

1. Press the **F5** function key under EDIT (edit mode indicated with an asterisk).
2. Press the **F1** or the **F3** function keys to select the HI and LO values for axis 1 and 2, respectively (cursor appears under the first digit of the value selected).
3. Using the numeric keyboard, enter the value desired.
4. Repeat steps 2 and 3 for all values to be changed.
5. Press **ENTER** when finished editing.
6. To jog with the new velocity/voltage values, first press the **F6** function key (under JOG) to enable the arrow keys again.

Closed-loop jog acceleration and deceleration values are specified by JOGA and JOGAD commands, respectively.

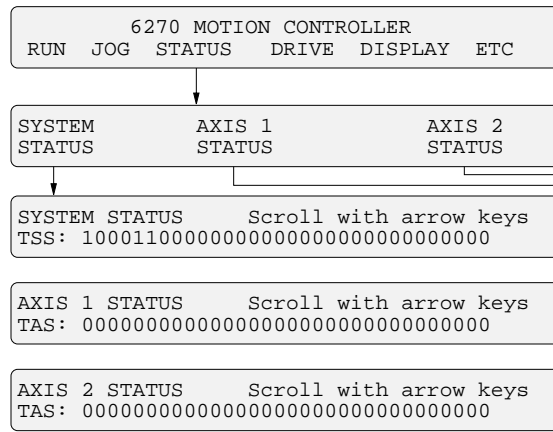
Open Loop Jogging

When using the open-loop jog mode, you should be aware of these conditions:

- **WARNING** — The hardware and software end-of-travel limits are disabled. Make sure that it is safe to operate without end-of-travel limits before using the open-loop jog function.
- Gain values (SGILIM, SGAF, SGAFN, SGI, SGIN, SGP, etc.) set to zero (open-loop operation).
- SMPER value set to zero (position error is allowed to increase without causing a fault).
- Subsequent attempts to change gain values or SMPER will cause an error message ("NOT ALLOWED IF SFBØ").
- SOFFS and SOFFSN set to zero, but allows subsequent servo offset changes to affect motion.
- SSWG set to zero (disables the Setpoint Window Gains feature).
- Disables output-on-position (OUTPA - OUTPD) functions.
- Any subsequent changes to PSET, PSETCLR, SCLD, SCLA, SCLV, SOFFS, and SOFFN are lost when another feedback source is selected.

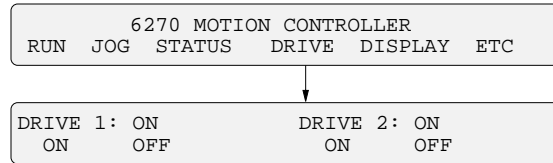
Recommendation — Use the Disable Drive On Kill mode, enabled with the KDRIVE command, so that the 6270 will shut down the valve/drive if a kill command (e.g., !K) is executed or if a kill input is activated. **CAUTION:** Shutting down a valve/cylinder system returns the valve to the null position; shutting down a rotary drive system allows the load to freewheel if there is no brake installed.

Status Reports:
System & Axis



After accessing the desired status menu, you can ascertain the function and status of each system (TSS) or axis (TAS) status bit by pressing the arrow keys on the numeric keypad.

Enabling and Disabling the Valves or Drives

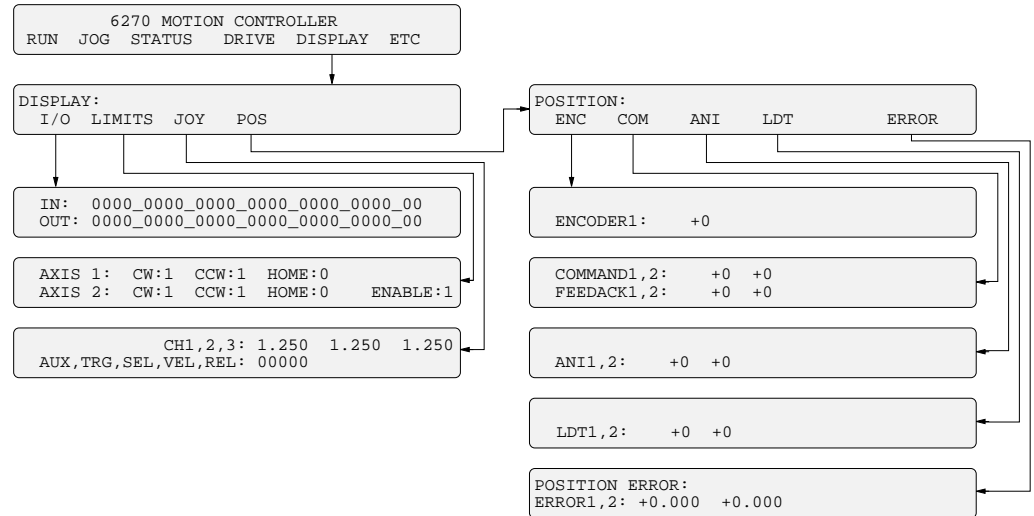


In the Drive menu, the current status of the valves/drives is displayed. To enable or disable valve/drive #1, press **F1** or **F2**; press **F4** or **F5** to enable or disable valve/drive #2. This menu offers the same functionality as the DRIVE command.

CAUTION

Shutting down a valve/cylinder system returns the valve to the null position. Shutting down a rotary drive system allows the load to freewheel if there is no brake installed.

Status Reports:
I/O, Limits, Joystick, & Position



I/O Menu:

- Input bit pattern (left to right): Bits 1-24 are the general-purpose programmable inputs, bits 25 & 26 are triggers A and B (TRG-A and TRG-B on the AUX connector).
- Output bit pattern (left to right): Bits 1-24 are the general-purpose programmable outputs, bits 25 & 26 are auxiliary outputs A and B (OUT-A and OUT-B on the AUX connector).

LIMITS Menu:

- “CW” refers to the hardware end-of-travel limit imposed when counting in the positive direction; this usually implies extension for cylinders. “CCW” refers to limit imposed when counting in the negative direction; this usually implies retraction for cylinders.

- A “1” indicates that the input is grounded, “0” indicates not grounded.
The end-of-travel limits (CW and CCW) must be grounded to allow motion (this is reversed if the active level is reversed with the LHLVL command).
The Enable input (ENBL terminal on the AUX connector) must also be grounded before motion is allowed. When not grounded, the analog output is held to zero volts and the shutdown outputs are activated.

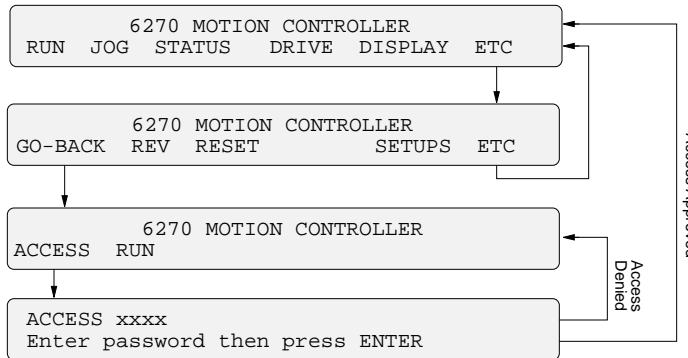
JOY Menu:

- The voltage levels present on the 3 analog channel on the 25-pin D JOYSTICK connector are listed.
- Also listed is the status of the joystick inputs (“1” indicates that the input is grounded/low, “0” indicates not grounded). AUX, TRG, SEL, VEL, and REL correspond respectively to pins 19, 18, 15, 16, and 17 on the JOYSTICK connector.

POS Menu (and sub-menus):

- The position and position error values shown are continually updated.
- Position values (except for ANI) are subject to the SCLD scaling factor (if scaling is enabled—SCALE1), PSET offset value, feedback polarity (ENCPOL and LDTPOL), and commanded direction polarity (CMDDIR). The ANI reading is always volts and is not scaled.
- “FEEDBACK” in the COM sub-menu refers to the feedback device currently selected with the SFB command (default is LDT).
- Position error = command position - actual (“feedback”) position.

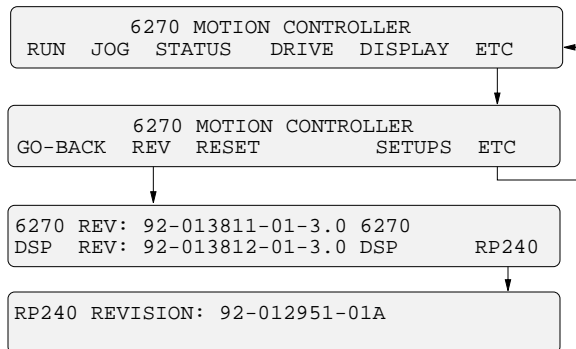
Access Security



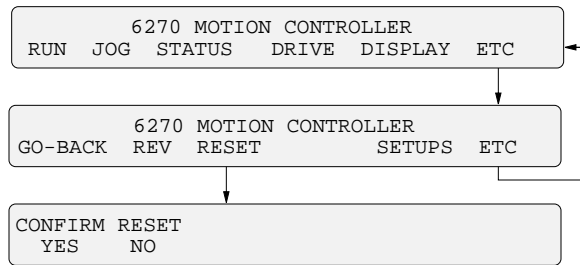
If the RP240 password is modified with the DPASS command to be other than DPASS6270 (the default password is “6270”) the access menu then becomes the new default menu after power-up or executing a RESET. After that, the new

password must then be entered to access the original default menu (see Access Approved path in illustration above). If the operator does not know the new password, all he or she can do is execute programs stored in the 6270 (RUN).

Revision Levels



Resetting the 6270

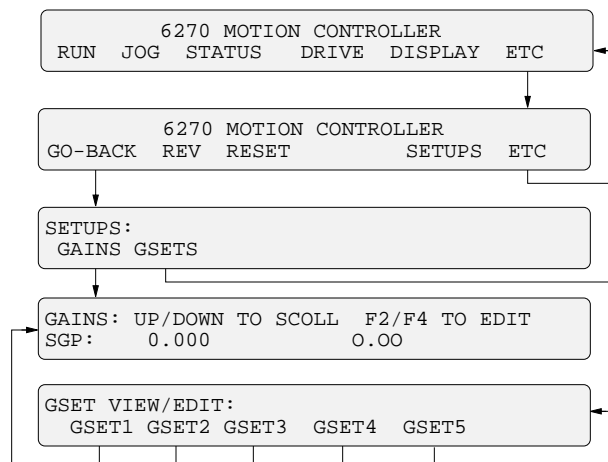


After accessing the RESET menu, press the **F1** key to execute a reset (or press **F2** to cancel and exit the menu). The reset is identical to issuing a RESET command or cycling power to the 6270. If a start-up program has been assigned with the STARTP command, that program will be executed.

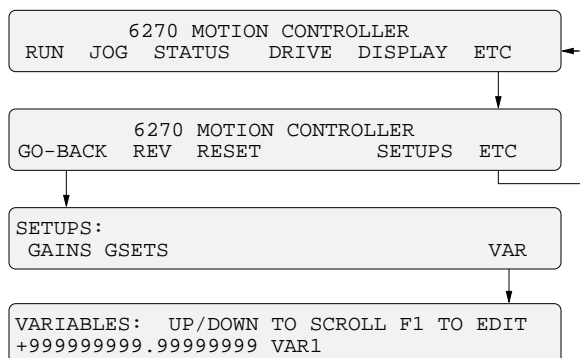
CAUTION

Executing a reset will restore all command values (those not saved in programs or variables) to their factory default setting.

Viewing and Editing Gains and Gain Sets



Viewing and Editing Variables



From the VAR menu, you can view all 150 numeric variables. The value and text of the first 25 numeric and string variables are presented side by side (e.g., the numeric value of VAR1 is displayed next to the text string in VARS1).

If no text is assigned to the corresponding string variable, the name of the variable is displayed (see example above).

To edit the displayed numeric variable, press **F2** (cursor appears) and type the new value on the numeric keypad.

Host Computer Operation

Another choice for a user interface is to use a host computer and execute a motion program using the RS-232C serial interface. A host computer may be used to run a motion program interactively from a BASIC or C program (high-level program controls the 6270 and acts as a user interface). A BASIC program example is provided below.

```
10 '      6000 Series Serial Communication BASIC Routine
12 '                      6000.BAS
14 '
16 ' *****
18 '
20 ' This program will set the communications parameters for the
22 ' serial port on a PC to communicate with a 6000 series
24 ' stand-alone product.
26 '
28 ' *****
30 '
100 '*** open com port 1 at 9600 baud, no parity, 8 data bits, 1 stop bit
110 '*** activate Request to Send (RS), suppress Clear to Send (CS), suppress
120 '*** DATA set ready (DS), and suppress Carrier Detect (CD) ***
130 OPEN "COM1:9600,N,8,1,RS,CS,DS,CD" FOR RANDOM AS #1
140 '
150 '*** initialize variables ***
160 MOVE$ = "" ' *** commands to be sent to the product ***
170 RESPONSE$ = "" ' *** response from the product ***
180 POSITION$ = "" ' *** feedback position reported ***
190 SETUP$ = "" ' *** setup commands ***
200 '
210 '*** format the screen and wait for the user to start the program ***
220 CLS : LOCATE 12, 20
230 PRINT "Press any key to start the program"
240 '
250 '*** wait for the user to press a key ***
260 PRESS$ = INKEY$
270 IF PRESS$ = "" THEN 260
280 CLS
290 '
300 '*** set a pre-defined move to make ***
310 SETUP$ = "ECHO1:ERRLVL0:LH0,0:"
320 MOVE$ = "A100,100:V2,2:D1000,1000:GO11:TFB:"
330 '
340 '
400 '*** send the commands to the product ***
410 PRINT #1, SETUP$
420 PRINT #1, MOVE$
430 '
500 '*** read the response from the TPE command ***
510 ' *** the controller will send a leading "+" or "-" in response to the TFB command to
520 ' *** indicate which direction travel has occurred. ***
530 WHILE (RESPONSE$ <> "+" AND RESPONSE$ <> "-") ' *** this loop waits for the "+"
540     RESPONSE$ = INPUT$(1, #1) ' *** or "-" characters to be returned
550 WEND ' *** before reading the position ***
560 '
570 WHILE (RESPONSE$ <> CHR$(13)) ' *** this loop reads one character at a time
580     POSITION$ = POSITION$ + RESPONSE$ ' *** from the serial buffer until a carriage
590     RESPONSE$ = INPUT$(1, #1) ' *** return is encountered ***
600 WEND
610 '
620 '*** print the response to the screen ***
630 LOCATE 12, 20: PRINT "Position is " + POSITION$
640 '
650 'END
```

Variables

HINT: Numeric and string variables may be edited via the RP240 (see RP240 Remote Operator Panel Interface section earlier in this chapter).

The 6270 has 3 types of variables (numeric, binary, and string). There are 150 numeric variables, numbered 1 - 150. There are 25 binary and string variables, numbered 1 - 25. Each type of variable is designated with a different command. The VAR command designates a numeric variable, the VARB command designates a binary variable, and the VARS command designates a string variable.

Variables do not share the same memory (i.e., VAR1, VARB1, and VARS1 can all exist at the same time and operate separately).

Numeric variables are used to store numeric values with a range of -999,999,999.00000000 to 999,999,999.99999999. Mathematical, trigonometric, and boolean operations are performed using numeric variables.

Binary variables can be used to store 32-bit binary or hexadecimal values. Binary variables can also store I/O, system, axis, or error status (e.g., the VARB2=IN.12 command assigns input bit 12 to binary variable 2). Bitwise operations are performed using binary variables.

String variables are used to store message strings of 20 characters or less. These message strings can be predefined error messages, user messages, etc.

NOTE

The programming examples in this section make use of the colon (:) command delimiter to allow entering more than one command per line.

Converting Between Binary and Numeric Variables

Using the Variable Type Conversion (VCVT) operator, you can convert numeric values to binary values, and vice versa. The operation is a signed operation as the binary value is interpreted as a two's complement number. Any *don't cares* (x) in a binary value is interpreted as a zero (0).

If the mathematical statement's result is a numeric value, then VCVT converts binary values to numeric values. If the statement's result is a binary value, then VCVT converts numeric values to binary values.

<i>Numeric to Binary</i>	<p>Example</p> <pre>> VAR1=-5 > VARB1=VCVT(VAR1) > VARB1</pre>	<p>Description/Response</p> <pre>Set numeric variable value = -5 Convert the numeric value to a binary value *VARB1=1101_1111_1111_1111_1111_1111_1111_1111</pre>
<i>Binary to Numeric</i>	<p>Example</p> <pre>> VARB1=b0010_0110_0000_0000_0000_0000_0000_0000 > VAR1=VCVT(VARB1) > VAR1</pre>	<p>Description/Response</p> <pre>Set binary variable = +100.0 Convert the binary value to a numeric value *VAR1=+100.0</pre>

Performing Operations with Numeric Variables

Some Math Operations Reduce Precision

The following math operations reduce the precision of the return value: Division and Trigonometric functions yield 5 decimal places; Square Root yields 3 decimal places; and Inverse Trigonometric functions yield 2 decimal places.

Mathematical Operations

The following examples demonstrate how the 6270 can perform math operations with its numeric variables. Operator precedence occurs from left to right (e.g., VAR1=1+1+1*3 sets VAR1 to 9, not 5).

Addition (+)	<p>Example</p> <pre>> VAR1=5+5+5+5+5+5 : VAR1 > VAR23=1000.565 > VAR11=VAR1+VAR23 : VAR11 > VAR1=VAR1+5 : VAR1</pre>	<p>Response</p> <pre>+VAR1=35.0 *VAR11=+1035.565 *VAR1=+40.0</pre>
--------------	---	---

Subtraction (-)	Example > VAR3=20-10 > VAR20=15.5 > VAR3=VAR3-VAR20 : VAR3	Response *VAR3=-5.5
Multiplication (*)	Example > VAR3=10 > VAR3=VAR3*20 : VAR3	Response *VAR3=+200.0
Division (/)	Example > VAR3=10 > VAR20=15.5 : VAR20 > VAR3=VAR3/VAR20 : VAR3 > VAR30=75 : VAR30 > VAR19=VAR30/VAR3 : VAR19	Response *+15.5 *+0.64516 *+75.0 *+116.25023
Square Root	Example > VAR3=75 > VAR20=25 > VAR3=SQRT(VAR3) : VAR3 > VAR20=SQRT(VAR20)+SQRT(9) > VAR20	Response *+8.660 *+8.0

Trigonometric Operations

The following examples demonstrate how the 6270 can perform trigonometric operations with its numeric variables.

Sine	Example > RADIAN0 > VAR1=SIN(0) : VAR1 > VAR1=SIN(30) : VAR1 > VAR1=SIN(45) : VAR1 > VAR1=SIN(60) : VAR1 > VAR1=SIN(90) : VAR1 > RADIAN1 > VAR1=SIN(0) : VAR1 > VAR1=SIN(PI/6) : VAR1 > VAR1=SIN(PI/4) : VAR1 > VAR1=SIN(PI/3) : VAR1 > VAR1=SIN(PI/2) : VAR1	Response *VAR1=+0.0 *VAR1=+0.5 *VAR1=+0.70711 *VAR1=+0.86603 *VAR1=+1.0 *VAR1=+0.0 *VAR1=+0.5 *VAR1=+0.70711 *VAR1=+0.86603 *VAR1=+1.0
Cosine	Example > RADIAN0 > VAR1=COS(0) : VAR1 > VAR1=COS(30) : VAR1 > VAR1=COS(45) : VAR1 > VAR1=COS(60) : VAR1 > VAR1=COS(90) : VAR1 > RADIAN1 > VAR1=COS(0) : VAR1 > VAR1=COS(PI/6) : VAR1 > VAR1=COS(PI/4) : VAR1 > VAR1=COS(PI/3) : VAR1 > VAR1=COS(PI/2) : VAR1	Response *VAR1=+1.0 *VAR1=+0.86603 *VAR1=+0.70711 *VAR1=+0.5 *VAR1=+0.0 *VAR1=+1.0 *VAR1=+0.86603 *VAR1=+0.70711 *VAR1=+0.5 *VAR1=+0.0
Tangent	Example > RADIAN0 > VAR1=TAN(0) : VAR1 > VAR1=TAN(30) : VAR1 > VAR1=TAN(45) : VAR1 > VAR1=TAN(60) : VAR1 > RADIAN1 > VAR1=TAN(0) : VAR1 > VAR1=TAN(PI/6) : VAR1 > VAR1=TAN(PI/4) : VAR1 > VAR1=TAN(PI/3) : VAR1	Response *VAR1=+0.0 *VAR1=+0.57735 *VAR1=+1.0 *VAR1=+1.73205 *VAR1=+0.0 *VAR1=+0.57735 *VAR1=+1.0 *VAR1=+1.73205
Inverse Tangent (Arc Tangent)	Example > RADIAN0 > VAR1=SQRT(2) > VAR1=ATAN(VAR1/2) : VAR1 > VAR1=ATAN(.57735) : VAR1	Response *VAR1=+35.26 *VAR1=+30.0

Boolean Operations

The 6270 has the ability to perform Boolean operations with its numeric variables. The following examples illustrate this capability. Refer to the **6000 Series Software Reference Guide** for more information.

Boolean And (&)	Example > VAR1=5 : VAR2=-1 > VAR3=VAR1 & VAR2 : VAR3	Response *VAR3=+0.0
Boolean Or ()	Example > VAR1=5 : VAR2=-1 > VAR3=VAR1 VAR2 : VAR3	Response *VAR3=+1.0
Boolean Exclusive Or (^)	Example > VAR1=5 : VAR2=-1 > VAR3=VAR1 ^ VAR2 : VAR3	Response *VAR3=+1.0
Boolean Not (~)	Example > VAR1=5 > VAR3=~(VAR1) : VAR3 > VAR1=-1 > VAR3=~(VAR1) : VAR3	Response *VAR3=+0.0 *VAR3=+1.0

Performing Operations with Binary Variables

The 6270 has the ability to perform bitwise functions with its binary variables. The following examples illustrate the bit manipulation capabilities of the 6270.

Bitwise And (&)	Example > VARB1=b1101 : VARB1	Response *VARB1=1101_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
	Example > VARB1=VARB1 & bXXX1 1101 : VARB1	Response *VARB1=XX01_XX0X_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
	Example > VARB1=h0032 FDA1 & h1234 43E9 : VARB1	Response *VARB1=0000_0000_1100_0000_0010_1000_0101_1000
Bitwise Or ()	Example > VARB1=h32FD : VARB1	Response *VARB1=1100_0100_1111_1011_0000_0000_0000_0000
	Example > VARB1=VARB1 bXXX1 1101 : VARB1	Response *VARB1=11X1_1101_1111_1X11_XXXX_XXXX_XXXX_XXXX
	Example > VARB1=h0032 FDA1 h1234 43E9 : VARB1	Response *VARB1=1000_0100_1100_0110_1111_1111_0111_1001
Bitwise Exclusive Or (^)	Example > VARB1=h32FD ^ bXXX1 1101 : VARB1	Response *VARB1=XXX1_1001_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
	Example > VARB1=h0032 FDA1 ^ h1234 43E9 : VARB1	Response *VARB1=1000_0100_0000_0110_1101_0111_0010_0001
Bitwise Not (~)	Example > VARB1=~(h32FD) : VARB1	Response *VARB1=0011_1011_0000_0100_1111_1111_1111_1111
	Example > VARB1=~(b1010 XX11 0101) : VARB1	Response *VARB1=0101_XX00_1010_XXXX_XXXX_XXXX_XXXX_XXXX
Shift Left to Right (>>)	Example > VARB1=h32FD >> h4 : VARB1	Response *VARB1=0000_1100_0100_1111_1011_0000_0000_0000
	Example > VARB1=b1010 XX11 0101 >> b11 : VARB1	Response *VARB1=0001_010X_X110_101X_XXXX_XXXX_XXXX_XXXX
Shift Right to Left (<<)	Example > VARB1=h32FD << h4 : VARB1	Response *VARB1=0100_1111_1011_0000_0000_0000_0000_0000
	Example > VARB1=b1010 XX11 0101 << b11 : VARB1	Response *VARB1=0XX1_1010_1XXX_XXXX_XXXX_XXXX_XXXX_X000

Teach Mode

The Teach Mode is simply a method of storing (*teaching*) variable data and later using the stored data as a source for motion program parameters. The variable data can be any value that can be stored in a numeric (VAR) variable (e.g., position, acceleration, velocity, etc). The variable data is stored into a *data program*, which is an array of *data elements* that have a specific address from which to write and read the variable data. Data programs do not contain 6000 Series commands.

The information below describes the principles of using the data program in a teach mode application. Following that is a teach mode application example in which the joystick is used to teach position data to be used in a motion program.

Teach Mode Basics

The basic process of using a data program for teach mode applications is as follows:

1. Initialize a data program.
2. Teach (store/write) variable data into the data program.
3. Read the data elements from the data program into a motion program.

Initialize a Data Program

This is accomplished with the DATSIZ command. The DATSIZ command syntax is DATSIZ*i*<, *i*>. The first integer (*i*) represents the number of the data program (1 - 50). You can create up to 50 separate data programs. The data program is automatically given a specific program name (DATP*i*). The second integer represents the total number of data elements (up to 6,500) you want in the data program. Upon issuing the DATSIZ command, the data program is created with all the data elements initialized with a value of zero.

The data program has a tabular structure, where the data elements are stored 4 to a line. Each line of data elements is called a *data statement*. Each element is numbered in sequential order from left to right (1 - 4) and top to bottom (1 - 4, 5 - 8, 9 - 12, etc.). You can use the TPROG DATP*i* command ("*i*" represents the number of the data program) to display all the data elements of the data program.

For example, if you issue the DATSIZ1, 13 command, data program #1 (called DATP1) is created with 13 data elements initialized to zero. The response to the TPROG DATP1 command is depicted below. Each line (*data statement*) begins with DATA=, and each data element is separated with a comma.

```
*DATA=0.0,0.0,0.0,0.0
*DATA=0.0,0.0,0.0,0.0
*DATA=0.0,0.0,0.0,0.0
*DATA=0.0
```

Each data statement, comprising four data elements, uses 39 bytes of memory. The memory for each data statement is subtracted from the memory allocated for user programs (see MEMORY command).

Teach the Data to the Data Program

The data that you wish to write to the data elements in the data program must first be placed into numeric variables (VAR). Once the data is stored into numeric variables, the data elements in the data program can be edited by using the Data Pointer (DATPTR) command to move the data pointer to that element, and then using the Data Teach (DATTECH) command to write the datum from the numeric variable into the element.

When the DATSIZ command is issued, the internal data pointer is automatically positioned to data element #1. Using the default settings for the DATPTR command, the numeric variable data is written to the data elements in sequential order, incrementing one by one. When the last data element in the data program is written, the data pointer is automatically set to data element #1 and a warning message (*WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1) is displayed. The warning message does not interrupt program execution.

The DATPTR command syntax is DATPTR*i*, *i*, *i*. The first integer (*i*) represents the data program number (1 through 50). The second integer represents the number of the data element to point to (1 through 6500). The third integer represents the number of data elements by which the pointer will increment after writing each data element from the DATTC command, or after recalling a data element with the DAT command.

The DATTC command syntax is DATTC*i*<, *i*, *i*, *i*>. Each integer (*i*) represents the number of a numeric variable. The value of the numeric variable will be stored into the data element(s) of the currently active data program (i.e., the program last specified with the last DATSIZ or DATPTR command). As indicated by the number of integers in the syntax, the maximum number of variable values that can be stored in the data program per DATTC command is 4. Each successive value from the DATTC command is stored to the data program according to the pattern established by the third integer of the DATPTR command.

As an example, suppose data program #1 is configured to hold 13 data elements (DATSIZ1, 13), the data pointer is configured to start at data element #1 and increment 1 data element after every value stored from the DATTC command (DATPTR1, 1, 1), and the values of numeric variables #1 through #3 are already assigned (VAR1=2, VAR2=4, VAR3=8). If you then enter the DATTC1, 2, 3 command, the values of VAR1 through VAR3 will be assigned respectively to the first three data elements in the data program, leaving the pointer pointing to data element #4. The response to the TPROG DATP1 command would be as follows (the text is highlighted to illustrate the final location of the data pointer after the DATTC1, 2, 3 command is executed):

```
*DATA=2.0, 4.0, 8.0, 0.0
*DATA=0.0, 0.0, 0.0, 0.0
*DATA=0.0, 0.0, 0.0, 0.0
*DATA=0.0
```

If you had set the DATPTR command to increment 2 data elements after every value from the DATTC command (DATPTR1, 1, 2), the data program would be filled differently and the data pointer would end up pointing to data element #7:

```
*DATA=2.0, 0.0, 4.0, 0.0
*DATA=8.0, 0.0, 0.0, 0.0
*DATA=0.0, 0.0, 0.0, 0.0
*DATA=0.0
```

Recall the Data from the Data Program

After storing (*teaching*) your variables to the data program, you can use the DATPTR command to point to the data elements and the DAT*i* ("i" = data program number) data assignment command to read the stored variables to your motion program. *You cannot recall more than one data element at a time; therefore, if you want to recall the data in a one-by-one sequence, the third integer of the DATPTR command must be a 1 (this is the default setting).*

Summary of Related 6000 Series Commands

NOTE: A detailed description of each command is provided in the **6000 Series Software Reference Guide**.

- DATSIZ**..... Establishes the number of data elements a specific data program is to contain. A new **DATPi** program name is automatically generated according to the number of the data program ($i = 1$ through 50). The memory required for the data program is subtracted from the memory allocated for user programs (see **MEMORY** command).
- DATPTR**..... Moves the data pointer to a specific data element in any data program. This command also establishes the number of data elements by which the pointer increments after writing each data element from the **DATTCH** command and after recalling each data element with the **DAT** command.
- DATTCH**..... Stores the variable data into the data program specified with the last **DATSIZ** or **DATPTR** command. After the data is stored, the data pointer is incremented the number of times entered in the third integer of the **DATPTR** command. *The data must first be assigned to a numeric variable before it can be taught to the data program.*
- TDPTR**..... Responds with a 3-integer status report (i, i, i): First integer is the number of the active data program (the program # specified with the last **DATSIZ** or **DATPTR** command); Second integer is the location number of the data element to which the data pointer is currently pointing; Third integer is the increment set with the last **DATPTR** command.
- [**DPTR**].... From the currently active data program, uses the number of the data pointer's location in a numeric variable assignment operation or a conditional statement operation.
- [**DATPi**].. The name of the data program created after issuing the **DATSIZ** command. The integer (i) represents the number of the data program. Data programs can be deleted just like any other user program (e.g., **DEL DATP1**).
- [**DATi**].... From the data program specified with i , assigns the numeric value of the data element (currently pointed to by the data pointer) to a specified variable parameter in a 6000 series command (e.g., **D(DAT3), (DAT3)**).

Teach Mode Application Example

In this example, 2 axes of the 6270 are used to move a 2-axis stage. This example illustrates a common method of teaching a path by using the joystick to move the load into position, teach the position (triggered by the Joystick Release input), then move to the next position. Five positions will be taught from each axis (2 axes at one trigger), for a total of 10 data elements in the data program. After all 10 positions are taught to the data program, the 6270 will automatically move both axes to a home position, move to each position that was taught, and then return to the home position.

For the sake of brevity, this example is limited to teaching 10 position data points; however, in a typical application, many more points would be taught. Also, it is assumed that end-of-travel and home limits are wired and a homing move has been programmed.

What follows is a suggested method of programming the 6270 for this application. To accomplish the teach mode application, a program called MAIN is created, comprising three subroutines: SETUP (to set up for teach mode), TEACH (to teach the positions), and DOPATH (to implement a motion program based on the positions taught).

The joystick operation in this example is based on setting the Joystick Axes Select input (pin #15 on the Joystick connector) to high to select analog input channels #1 and #2 (pins #1 and #2) for joystick use, and using the Joystick Release input (pin #17) to trigger the position teach operation.

Step 1 Initialize a Data Program.

- > DEL DATP1 Delete data program #1 (DATP1) in preparation for creating a new data program #1
- > DATSIZ1,10 Create data program #1 (named DATP1) with an allocation of 10 data elements. Each element is initialized to zero.

Step 2 Define the SETUP Subroutine. Note that the SETUP subroutine need only run once.

- > DEF SETUP Begin definition of the subroutine called SETUP
- JOYVH3,3 Set the high velocity speed to 3 units/sec
- JOYVL.2,.2 Set the low velocity to 0.2 units/sec
- JOYAXH1,2 When axes select input is set high, apply analog input 1 to axis 1 and apply analog input 2 to axis 2
- VAR1=0 Set variable #1 equal to zero
- VAR2=0 Set variable #2 equal to zero
- DRIVE11 Enable the drives for both axes
- MA11 Enable the absolute positioning mode for both axes
- END End definition of the subroutine called SETUP

Step 3 Define the TEACH Subroutine.

- > DEF TEACH Begin definition of the subroutine called TEACH
- HOM11 Home both axes (absolute position counter is set to zero after homing move)
- DATPTR1,1,1 Select data program #1 (DATP1) as the current active data program, and move the data pointer to the first data element. After each DATTCH value is stored to DATP1, increment the data pointer by 1 data element.
- REPEAT Set up a repeat/until loop
- JOY11 Enable joystick mode on both axes. At this point, you can start moving the axes into position with the joystick. While using the joystick, command processing is stopped here until you activate the joystick release input. Activating the joystick release input disables the joystick mode and allows the subsequent commands to be executed (assign the current positions to the variables and then store the positions in the data program).
- VAR1=1PM Set variable #1 equal to the position of motor 1
- VAR2=2PM Set variable #2 equal to the position of motor 2

- DATTCH1,2 Store variable #1 and variable #2 into consecutive data elements.
(The first time through the repeat/until loop, variable #1 is stored into data element #1 and variable #2 is stored into data element #2. The data pointer is automatically incremented once after each data element and ends up pointing to the third data element in anticipation of the next DATTCH command.)
- WAIT(INO.5=b1) Wait for the joystick release input to be de-activated
- UNTIL(DPTR=1) Repeat the loop until the data pointer wraps around to data element #1
(data program full)
- END End definition of the subroutine called TEACH

Step 4 Define the DOPATH Subroutine.

- > DEF DOPATH Begin definition of the subroutine called DOPATH
- HOM11 Move both axes to the home position (absolute counters set to zero)
- A50,50 Set up the acceleration
- V3,3 Set up the velocity
- DATPTR1,1,1 Select data program #1 (DATP1) as the current active data program, and set the data pointer to the first data element. Increment the data pointer one element after every data assignment with the DAT command. *If you wanted to move only axis 1 down the taught path, you would set the increment (third integer) to a 2, thus accessing only the axis 1 stored positions.*
- REPEAT Set up a repeat/until loop
- D(DAT1),(DAT1) The position of axis 1 and axis 2 are recalled into the distance command
- GO11 Move to the position
- T.5 Wait for 0.5 seconds
- UNTIL(DPTR=1) Repeat the loop until the data pointer wraps around to data element #1
(all data elements have been read)
- HOM11 Move both axes back to the home position
- END End definition of the subroutine called DOPATH

Step 5 Define the MAIN Program (Include SETUP, TEACH, and DOPATH).

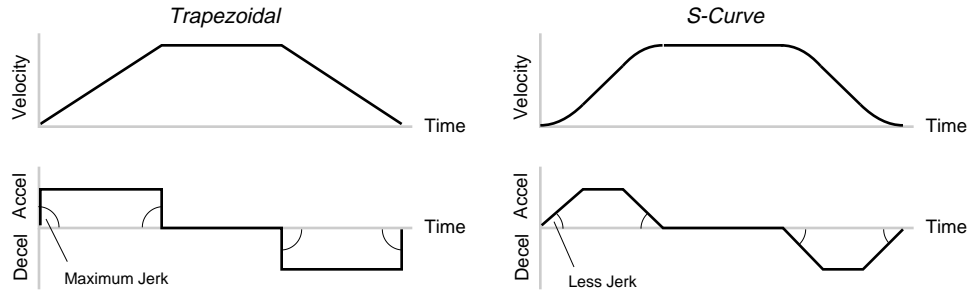
- > DEF MAIN Begin definition of the program called MAIN
- SETUP Execute the subroutine called SETUP
- TEACH Execute the subroutine called TEACH
- DOPATH Execute the subroutine called DOPATH
- END End definition of the program called MAIN

Step 6 Run the MAIN Program and Teach the Positions with the Joystick.

1. Enter the MAIN command to execute the teach mode program and set the joystick's *axis select* input to high.
2. Use the joystick to move to the position to be taught.
3. Once in position, activate the *joystick release* input to teach the positions. Two positions (one for each axis) are taught each time you activate the joystick release input.
4. Repeat steps 2 and 3 for the remaining four teach locations. After triggering the joystick release input the fifth time, the 6270 will home the axes, repeat the path that was taught, and then return both axes to the home position.

S-Curve Profiling

The 6270 allows you to perform *S-curve* move profiles, in addition to the usual trapezoidal profiles. *S-curve* profiling provides smoother motion control by reducing the *jerk* (rate of change) in acceleration and deceleration portions of the move profile (see drawing below).



S-curves improve position tracking.

Because *S-curve* profiling reduces jerk, it improves position tracking performance in servo systems, especially in linear interpolation applications.

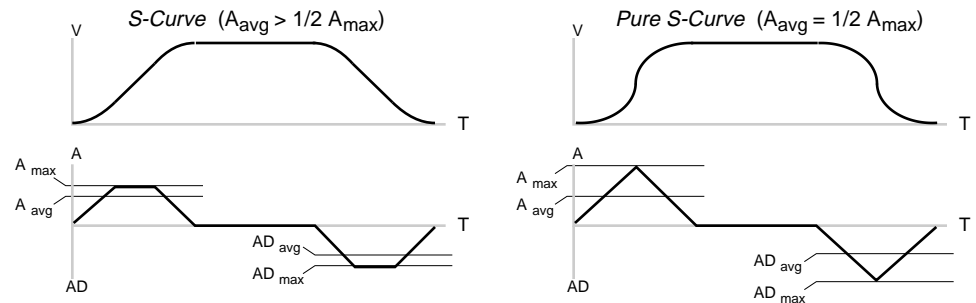
S-curve programming requirements.

To program an *S-curve* profile, you must use the *average accel/decel* commands provided in the 6000 Series programming language. For every maximum accel/decel command (e.g., A, AD, HOMA, HOMAD, JOGA, JOGAD, etc.) there is an *average* command for *S-curve* profiling (see table below).

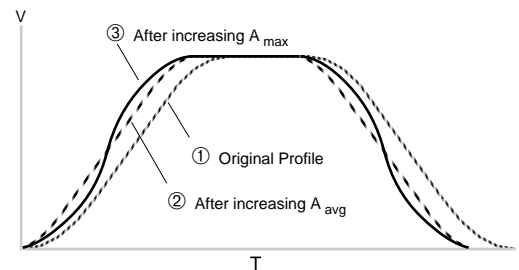
Maximum Accel/Decel Commands:		Average (S-Curve) Accel/Decel Commands:	
Command	Function	Command	Function
A	Acceleration	AA	Average Acceleration
AD	Deceleration	ADA	Average Deceleration
HOMA	Home Acceleration	HOMAA	Average Home Acceleration
HOMAD	Home Deceleration	HOMADA	Average Home Deceleration
JOGA	Jog Acceleration	JOGAA	Average Jog Acceleration
JOGAD	Jog Deceleration	JOGADA	Average Jog Deceleration
JOYA	Joystick Acceleration	JOYAA	Average Joystick Acceleration
JOYAD	Joystick Deceleration	JOYADA	Average Joystick Deceleration
LHAD	Hard Limit Deceleration	LHADDA	Average Hard Limit Deceleration
LSAD	Soft Limit Deceleration	LSADDA	Average Soft Limit Deceleration
PA	Path Acceleration	PAA	Average Path Acceleration
PAD	Path Deceleration	PADA	Average Path Deceleration

The command values for average accel/decel (AA, ADA, etc.) and maximum accel/decel (A, AD, etc.) determine the characteristics of the *S-curve*. To smooth the accel/decel ramps, you must enter average accel/decel command values that satisfy the equation $1/2 A_{max} \leq A_{avg} < A_{max}$, where A_{max} represents maximum accel/decel and A_{avg} represents average accel/decel. Given this requirement, the following conditions are possible:

- If $A_{avg} > 1/2 A_{max}$, but $A_{avg} < A_{max}$, you have achieved an S-curve profile with a variable period of constant accel/decel (see drawing below).
- If $A_{avg} = 1/2 A_{max}$, you have achieved what is called a *Pure S-curve* profile in which there is no period of constant accel/decel and jerk is at an absolute minimum (see drawing below).



- Once you enter an A_{avg} value that is \neq zero and satisfies $1/2 A_{max} \leq A_{avg} < A_{max}$, S-curve profiling is enabled, but only in the operation that uses that particular A_{avg} command. For example, entering a HOMAA command enables S-curve acceleration profiling only for homing moves, not for other functions such as jogging (which would require the JOGAA command). To return to the default trapezoidal profiling mode, enter an A_{avg} value of zero, or set $A_{avg} = A_{max}$.
- If $A_{avg} = A_{max}$, a trapezoidal profile results, but can be changed to an S-curve by specifying a new A_{avg} value less than A_{max} , or set A_{max} greater than A_{avg} .
- If $A_{avg} < 1/2 A_{max}$, or $A_{avg} > A_{max}$, when you try to initiate motion, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION-FIELD n, will be displayed.
- If $A_{avg} = 0$ or if you never enter an A_{avg} command, the 6270 defaults to trapezoidal profiling and the A_{avg} command value will always match the A_{max} command value. However, if you enter an A_{avg} deceleration of zero, you will receive the error message *INVALID DATA-FIELD n, where n is the number of the data field.
- If you never enter the maximum (A_{max}) or average (A_{avg}) decel command values (AD or ADA, HOMAD or HOMADA, etc.), the average decel value will always match, or track, the average accel value (AA, HOMAA, etc.). However, once you change the maximum decel, the average decel will no longer track the average accel.
- If you increase the A_{avg} value above the pure S-curve level ($A_{avg} > 1/2 A_{max}$), the time required to reach the target velocity and the target distance decreases; however, increasing A_{avg} also increases jerk. After increasing A_{avg} , you can reduce the jerk by increasing A_{max} (see illustration); however, increasing A_{max} requires a greater torque to achieve the commanded velocity at the mid-point of the acceleration profile.

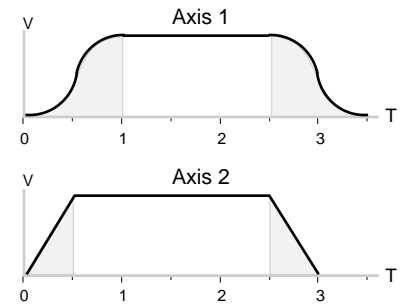


- You can calculate the profile's accel/decel time with the following equations (calculation method is identical for S-curve **and** trapezoidal):

$$\text{Time of accel or decel} = \frac{\text{Velocity}}{A_{avg}} \quad \text{or} \quad \text{Time of accel or decel} = \sqrt{\frac{2 * \text{Distance}}{A_{avg}}}$$

- Accel/decel scaling (SCLA or PSCLA) affects A_{avg} the same as it does for A_{max} , regardless of if the profile is trapezoidal or S-curve (see *Scaling* section in Chapter 5).

Example	Command	Description
>	LDTRES432	Set resolution to 432 counts/rev
>	SCALE0	Disable scaling
>	@MA0	Select incremental positioning mode
>	@D5000	Set distances to 5000 counts
>	A10,10	Set max. accel to 10 (both axes)
>	AA5,10	Set avg. accel to 5 on axis 1, and 10 on axis 2
>	AD10,10	Set max. decel to 10 (both axes)
>	ADA5,10	Set avg. decel to 5 on axis 1, and 10 on axis 2
>	V5,5	Set velocity to 5 on both axes
>	GO	Execute motion on both axes



Axis 1 executes a pure S-curve profile that takes 1 second to reach a velocity of 5 and 1 second to return to zero velocity. Axis 2 executes a trapezoidal profile that takes 0.5 seconds to reach a velocity of 5 and 0.5 seconds to return to zero velocity.

X-Y Linear Interpolation

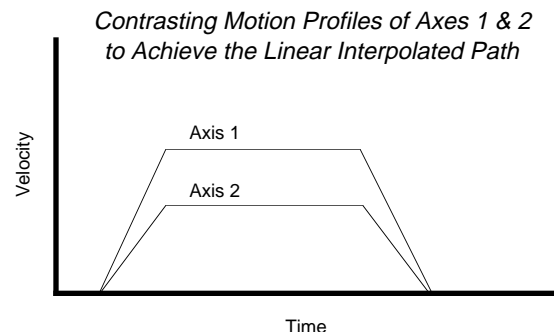
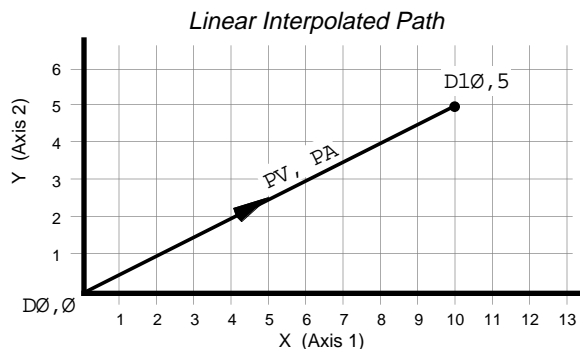
The 6270 allows you to perform *X-Y linear interpolation*, the process of moving two orthogonal (right angle) linear axes to achieve linear (straight line) motion. The task is to derive appropriate move parameters to move from a current location to a new location, where each position is specified by a set of *Cartesian coordinates*. Both axes must start, accelerate, decelerate, and stop in a synchronized manner.

The Initiate Linear Interpolated Motion (GOL) command initiates linear interpolation moves based on the parameters set with the D, PA, PAD, and PV commands. You simply enter the desired path acceleration (PA), the path deceleration (PAD), and the path velocity (PV) to arrive at the point in space (*end point*) specified with the distance (D) command; the 6270 internally calculates each axis' actual move profiles to achieve a straight-line path with these parameters.

You can scale the acceleration, velocity, and distance with the PSCLA, PSCLV, and SCLD commands, respectively (see example below).

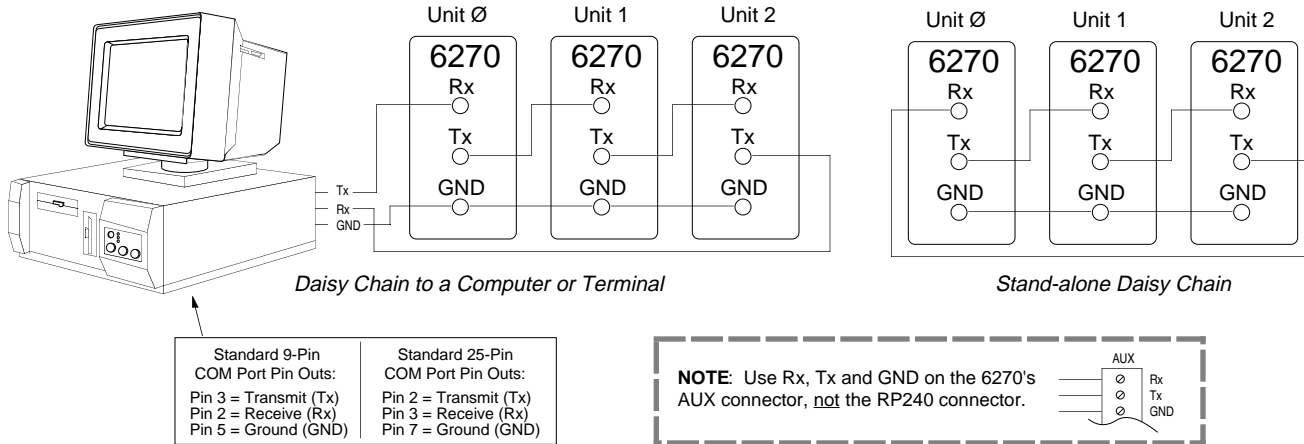
The GOL command starts motion on either or both axes. If the GOL command is issued without any arguments, motion will be started on both axes.

Example	Command	Description
>	SCALE1	Enable scaling
>	PSCLA432	Set path acceleration scale factor to 432 counts/unit
>	PSCLV432	Set path velocity scale factor to 432 counts/unit
>	@SCLD432	Set distance scale factor to 432 counts/unit on all axes
>	PA25	Set the path acceleration to 25 units/sec ²
>	PAD20	Set the path deceleration to 20 units/sec ²
>	PV2	Set the path velocity to 2 units/sec
>	D10,5	Set the distance to 10 & 5 units on axes 1 & 2, respectively
>	GOL11	Initiate linear interpolated motion on both axes (see figure below) (a GOL command could have been issued instead of a GOL11 command)



RS-232C Daisy-Chaining

Up to ninety-nine 6270s may be daisy-chained. There are two methods of daisy-chaining: one uses a computer or terminal as the controller in the chain; the other uses a 6270 as the master controller. The figure below illustrates examples of both daisy-chain types for three 6270s. *Be sure to use the Rx, Tx and GND on the AUX connector, not the RP240 connector.*



Follow these steps to implement daisy-chaining:

Step 1 To enable and disable communications on a particular 6270 unit in the chain, you must establish a unique device address using the unit's address DIP switches or the Daisy-chain Address (ADDR) command.

DIP switches: Instructions for accessing and changing these DIP switch settings are provided in the *Optional DIP Switch Settings* section in Chapter 6. Device addresses set with the DIP switches range from 0 to 7.

ADDR command: The ADDR command automatically configures unit addresses for daisy chaining by disregarding the DIP switch setting. This command allows up to 99 units on a daisy chain to be uniquely addressed.

Sending ADDR*i* to the first unit in the daisy chain sets its address to be (*i*). The first unit in turn transmits ADDR(*i* + 1) to the next unit to set its address to (*i* + 1). This continues down the daisy chain until the last unit of (*n*) daisy-chained units has its address set to (*i* + *n*).

Setting ADDR to 0 re-enables the unit's daisy-chain address configured on its internal DIP switch.

Note that a 6270 with the default device address of zero (0) will send an initial power-up start message similar to the following:

```
*PARKER 6270 - 2 AXIS MOTION CONTROLLER
*NO REMOTE PANEL
*ANALOG INPUT OPTION NOT DETECTED
*ADVANCED FOLLOWING FEATURES NOT DETECTED
```

Step 2 Connect the daisy-chain with a terminal as the master (see diagram above).

It is necessary to have the error level set to 1 for all units on the daisy-chain (ERRLVL1). When the error level is not set to 1, the 6270 sends ERROK or ERRLVL1 prompts after each command, which makes daisy-chaining impossible. Send the ERRLVL1 command from the master terminal (without an address prefix, all units will respond to ERRLVL1).

Command	Description
> ERRLVL1	Set error level to 1

After this has been accomplished a carriage return sent from the terminal will not cause any 6270 to send a prompt. Verify this. Instructions below show how to set the error level to 1 automatically on power-up by using the 6270's power-up start program (highly recommended).

After the error level for all units has been set to `ERRLVL1`, send a 6000 series command to all units on the daisy-chain by entering that command from the master terminal.

Command	Description
> <code>OUT1111</code>	Turn on outputs #1 - #4 on all units
> <code>A50,50</code>	Set acceleration to 50 for all axes (all units, both axes)

To send a 6000 series command to one particular unit on the daisy-chain, prefix the command with the appropriate unit's device address and an underline:

Command	Description
> <code>2_OUT0</code>	Turn off output #1 on unit #2
> <code>4_OUT0</code>	Turn off output #1 on unit #4

To receive data from a 6270, you **must** prefix the command with the appropriate unit's device address and an underline:

Command	Description
> <code>1_A</code>	Request acceleration information from unit #1
> <code>*A50,50</code>	Response from unit #1

Use the (E) command to enable/disable RS-232C communications for an individual unit. If all 6270 units on the daisy chain are enabled, commands without a device address identifier will be executed by all units. Because of the daisy-chain's serial nature, the commands will be executed approximately 1 ms per character later on each successive unit in the chain (assuming 9600 baud).

Units with the RS-232C disabled (E0) will not respond to any commands, except E1; however, characters are still echoed to the next device in the daisy chain.

Command	Description
> <code>3_E0</code>	Disable RS-232C on unit #3
> <code>VAR1=1</code>	Set variable #1 to 1 on all other units
> <code>3_E1</code>	Enable RS-232C on unit #3
> <code>3_VAR1=5</code>	Set variable #1 to 5 on unit #3

Verify communication to all units by using the techniques described above.

Step 3 Now that communication is established programming of the units can begin (alternately, units can be programmed individually by connecting the master terminal to one unit at a time). To allow daisy-chaining between multiple 6270s, the `ERRLVL1` command must be used to prevent units from sending error messages and command prompts. In every daisy-chained unit the `ERRLVL1` command should be placed in the program that is defined as the `STARTP` program:

Command	Description
> <code>DEF chain</code>	Begin definition of program chain
- <code>ERRLVL1</code>	Set error level to 1
- <code>GOTO main</code>	Go to program main
- <code>END</code>	End definition of program chain
> <code>STARTP chain</code>	Designates program chain as the power-up program

To define program `main` for unit #0:

Command	Description
> <code>0_DEF main</code>	Begin definition of program main on unit #0
- <code>0_GO</code>	Start motion
- <code>0_END</code>	End definition of program main on unit #0

Step 4 After all programming is completed program execution may be controlled by either a master terminal (diagram above), or by a master 6270 (diagram above).

Daisy-Chaining from a Computer or Terminal

Controlling the daisy-chain from a master computer or terminal follows the examples above:

Command	Description
> Ø_RUN main	Run program main on unit #0
> 1_RUN main	Run program main on unit #1
> 2_GO1	Start motion on unit #2 axis #1
> 3_2A	Get A command response from unit #3 axis #2

Daisy-Chaining from a Master 6270

Controlling the daisy-chain from a master 6270 (the first unit on the daisy-chain) requires stored programs in the master 6270 which can control program and command execution on the slave 6270s. The example below demonstrates the use of the WRITE command to send commands to other units on the daisy-chain.

NOTE

The last unit on the daisy-chain must have RS-232C echo disabled (ECHOØ command).

Master 6270's main program:

Command	Description
> DEF main	Program main
- L	Indefinite loop
- WHILE (IN.1 = bØ)	Wait for input #1 to go active
- NWHILE	
- GOL	Initiate linear interpolated move
- WHILE (IN.1 = b1)	Wait for input #1 to go inactive
- NWHILE	
- WRITE "2_D2ØØØ,4ØØØ"	Send message "2_D2ØØØ,4ØØØ" down the daisy chain
- WRITE "2_ACK"	Send message "2_ACK" down the daisy chain
- LN	End of loop
- END	End of program main

6270 unit #2 ack program:

Command	Description
> DEF ack	Program ack
- GO11	Start motion on both axes
- END	End of program ack

Daisy-Chaining and RP240s

RP240s cannot be placed in the 6270 daisy chain; RP240s can only be connected to the designated RP240 port on a 6270. It is possible to use only one RP240 with a 6270 daisy-chain to input data for multiple units on the chain. The example below (for the 6270 master with an RP240 connected) reads data from the RP240 into variables #1 (*data1*) & #2 (*data2*), then sends the messages 3_Ddata1,data2<CR> and 3_GO<CR>.

Command	Description
> L	Indefinite loop
VAR1=DREAD	Read RP240 data into variable #1
VAR2=DREAD	Read RP240 data into variable #2
EOTØ,Ø,Ø,Ø	Turn off <CR>
WRITE "3_D"	Send message "3_D" down the daisy chain
WRVAR1	Send variable #1 data down the daisy chain
WRITE " , "	Send message " , " down the daisy chain
EOT13,Ø,Ø,Ø	Turn on <CR>
WRVAR2	Send variable #2 data down the daisy chain
WRITE "3_GO"	Send message "3_GO" down the daisy chain
LN	End of loop