

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

Reference Guide

UMAC Quick Reference

Reference Guide for UMAC Products

3A0-UMACQR-xPRx

December 23, 2004



DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION **1**
Motion Control Applications 1
UMAC Turbo System 2
Features *3*
Installation and Setup 5
Hardware Setup *5*
Software Setup *5*
Programming UMAC 6
Online Commands *6*
Motion Programs *6*
PLC Programs *6*
UMAC Tasks 7
Single Character I/O *7*
Commutation Update *7*
Servo Update *7*
Real-Time Interrupt Tasks *8*
Background Tasks *8*

PMAC EXECUTIVE PROGRAM, PEWIN32PRO **11**
Configuring PEWIN 11
Establishing Communications *11*
Workspace Layout *11*
Quick Plot Feature 12
Saving and Retrieving PMAC Parameters 12
The Watch and Position Windows 13
Uploading and Downloading Files 13
Using MACRO Names and Include Files 13
Downloading Compiled PLCCs 13
The PID Tuning Utility 13
Auto Tuning *14*
Interactive Tuning *15*
Other Features 16

HARDWARE SETUP AND CONNECTIONS **17**
Address Configuration 17
Servo Cards *17*
IO Cards *17*
Serial Port Connections 18
Re-initializing UMAC *18*
Power Supply 19
Motor Flag Connections 19
Disabling the Overtravel Limit Flags *19*
Types of Overtravel Limits *19*
Home Sensors *20*
Checking the Flag Inputs *20*
Motor Signals Connections 21
Incremental Encoder Connection *21*
Checking the Encoder Inputs *21*
MLDT Feedback Connection *21*
DAC Output Signals *22*
Checking the DAC Outputs *22*
Pulse and Direction Stepper Signals *23*
Digital Amplifier Connections *23*
Amplifier Enable Signals *24*

Amplifier Fault Signals *24*
Digital Inputs and Outputs 25
Connection Examples 26
Digital Amplifier with Incremental Encoder *26*
Analog Amplifier with Incremental Encoder *27*
Analog Amplifier with MLDT Feedback *28*
Stepper Driver with Incremental Encoder *29*

SOFTWARE SETUP 31
Resetting UMAC 31
Motors Setup 31
Servo Loop Setup 31
Programming PMAC 31
Online Commands *31*
Buffered (Program) Commands *32*
Computational Features 33
I-Variables *33*
P-Variables *34*
Q-Variables *34*
M-Variables *35*
Arrays *36*
Operators *36*
Functions *37*
Comparators *38*
Encoder Conversion Table 38
Conversion Table Structure *38*
Further Position Processing *39*
PMAC Position Registers 39
Summary of Selected I-Variables 41
Motor Definition I-Variables *41*
Motor Safety I-Variables *41*
S Curve and Linear Acceleration Variables *42*
Rate vs. Time: Programming the Maximum Acceleration Parameters *42*
Benefits of Using S-Curve Acceleration Profiles *43*
Motor Movement I-Variables *43*
Servo Control I-Variables *44*
Channel Specific I-Variables *44*
Homing Search Moves 45
Jogging Moves 45
Indefinite Jog Commands *45*
Jogging to a Specified Position *45*
Jog Moves Specified by a Variable *45*
Jog-Until-Trigger *46*
Command and Send Statements 46

MOTION PROGRAMS 47
How PMAC Executes a Motion Program 47
Coordinate Systems 48
Axis Definitions *48*
Axis Definition Statements *49*
Writing a Motion Program 49
Running a Motion Program 50
Subroutines and Subprograms 52
Passing Arguments to Subroutines *52*
G, M, T, and D-Codes (Machine Tool Style Programs) *53*
NC Products 53
Linear Blended Moves 54

Linear Interpolated Moves Characteristics *55*
Circular Interpolation 58
Splined Moves 60
PVT-Mode Moves 61
Turbo PMAC Lookahead Function 62
Turbo PMAC Kinematic Calculations 64
Other Programming Features 65
Rotary Motion Program Buffers *65*
Internal Timebase, the Feedrate Override *65*
External Time-Base Control (Electronic Cams) *66*
Position Following (Electronic Gearing) *66*
Cutter Radius Compensation *66*
Synchronizing PMAC to other PMACs *66*
Axis Transformation Matrices *67*
Position-Capture and Position-Compare Functions *67*
Learning a Motion Program *67*

PLC PROGRAMS 69
Entering a PLC Program 70
PLC Program Structure 71
Calculation Statements 71
Conditional Statements 71
Level-Triggered Conditions *71*
Edge-Triggered Conditions *71*
WHILE Loops 72
COMMAND and SEND Statements 72
Timers 73
Compiled PLC Programs 74

TROUBLESHOOTING 75
Establishing Communications 75
Hardware Re-initialization 75
The Watchdog Timer (Red LED) 76
System Configuration 76
UMAC System Status Bits 76
Direct Access to Hardware Features 76
Motor Parameters 77
Motion Programs 78
PLC Programs 79

APPENDIX A — UMAC ERROR CODE SUMMARY 81
I6, Error Reporting Mode 81

APPENDIX B — SELECTED UMAC I-VARIABLES SUMMARY 83

APPENDIX C — SELECTED UMAC ONLINE COMMANDS 89

APPENDIX D — SELECTED UMAC MOTION PROGRAM COMMANDS 93

APPENDIX E — SELECTED UMAC PLC PROGRAM COMMANDS 95

APPENDIX F — MOTOR SUGGESTED M-VARIABLE DEFINITIONS 97

APPENDIX G — FIRST DIGITAL I/O ACCESSORY M-VARIABLES 103

INTRODUCTION

This manual introduces the most common hardware and software features of the UMAC Turbo system. It is intended for first-time users as a complement to the UMAC Turbo System manuals and related accessories. Use this quick reference manual in conjunction with the following manuals:

- Turbo PMAC User Manual
- Turbo PMAC Software Reference
- UMAC Turbo Hardware Reference
- UMAC Turbo Accessory Manuals

Motion Control Applications

A typical motion control application is composed of a computer, a motion controller, a set of amplifiers and motors, and the machine to be controlled.



Computer



Motion Controller



Amplifier



Electric Motor



Automated Machine

The computer is the interface between the user and the machine and it defines the automated tasks required for the machine as a series of motion program commands written as simple text files.

The motion programs, written as text files, are downloaded to the main memory of the motion controller for fast and continuous execution. The motion controller interprets the series of commands in the motion programs and converts them to proper electrical signals for the amplifier and motor to cause the programmed motion. The characteristics and timing of these signals will determine, for example, the distance, acceleration and velocity of motion for the different processes.

The use of an amplifier allows the standardization of the command signals from the motion controller to control virtually any kind and size of motor. The most commonly used command signal from a motion controller is a ±10V analog command signal. Usually, the amplifier interprets this signal as a torque command, which translates into an electrical current in the windings of the motor that causes the desired motion. An encoder device, usually placed in the back of the motor and mechanically engaged with the motor shaft, provides feedback information for the motion controller.

The electric motor is a device that converts electrical energy into mechanical energy. There are several kinds of motors including DC brush, AC brushless and stepper motors. It is important to know the maximum velocity and acceleration that the motor can deliver for the proper selection of the servo loop parameters in the motion controller. The accuracy of motion is determined mainly by the appropriate response of the amplifier and motor to the required motion command signals and by the resolution of the encoder feedback device.

The UMAC (Universal Motion and Automation Controller) is a motion controller system configurable to control virtually any kind of machine automation application. A single UMAC Turbo system can control up to 32 axes and thousands of digital I/O points with a great level of accuracy and simplicity of operation. The UMAC Turbo system can be configured to interface with virtually any kind of amplifier, motor and feedback device. In addition, the UMAC can use different kinds of communication methods with the host computer, including USB, Ethernet, RS-232 and PC/104 bus communications.

UMAC Turbo System

The UMAC (Universal Motion and Automation Controller) is a modular system built with a set of 3U-format Eurocards. The configuration of any UMAC System starts with the selection of the Turbo PMAC2 3U CPU board and continues with the addition of the necessary axes boards, I/O boards, communication interfaces (USB, Ethernet, etc.), and any other interface boards selected from the variety of available accessories. Accessory boards interface with virtually any kind of feedback sensor or implement almost any kind of communication method with the host computer or external devices. In addition, a PC/104 computer can be installed inside the UMAC System yielding an incredibly powerful system inside a compact industrial package.



Turbo PMAC2-3U

The Turbo PMAC2 3U CPU is based on the Motorola 56k DSP processor and a sophisticated firmware designed by Delta Tau Data Systems, Inc. This combination provides a highly accurate, flexible and powerful motion controller capable of controlling a large number of axes and I/O with simplicity of operation.



ACC-24E2 Axes

The axes interface boards are based on custom made ASIC gate array chips designed by Delta Tau Data Systems, Inc. These chips and the associated circuitry, interface between the Turbo PMAC2 3U CPU and the machine to output amplifier command signals, to input quadrature encoder feedback information, and to input flags information including end-of-travel limits and machine home sensors. Different kind of axes interface boards can be selected to control analog ±10V amplifiers, stepper drivers and direct digital PWM amplifiers.



UBUS Backplane

The I/O boards are based on custom-made ASIC chips that interface between the Turbo PMAC2 3U CPU and the machine to output or input a large number of I/O points with different electrical characteristics from which to choose.



Pack Frame

UMAC type boards are mounted inside 3U racks composed of pack frames and plug into the UBUS backplane. Each accessory board in the UMAC Turbo system has a unique settable address that maps into the Turbo PMAC2 3U CPU memory. The UBUS backplane connects the address and data lines from Turbo PMAC2 3U CPU to the different accessory boards. In addition, the 5V and ±12V lines are transmitted over the UBUS to power the different accessory boards.



UMAC Turbo

Features

- Up to 32 axes of motion control
- Analog ±10V, digital PWM or pulse and direction command signals
- Quadrature, incremental, encoder inputs
- Parallel binary feedback inputs
- Laser interferometer feedback devices inputs
- Analog feedback inputs
- Sinusoidal encoder feedback inputs with 4096 interpolation lines
- SSI encoders inputs
- Yaskawa or Mitsubishi absolute encoders inputs
- MLDTs feedback inputs
- Thousands of I/O points
- High-power, sinking, sourcing or OPTO-22 compatible I/O
- Up to 256 analog-to-digital converted inputs (12-bits or 16-bits resolution)
- Stand-alone or host commanded operation
- PC/104, USB, Ethernet or RS-232/422 communication methods supported

UMAC Products



Turbo PMAC2 3U



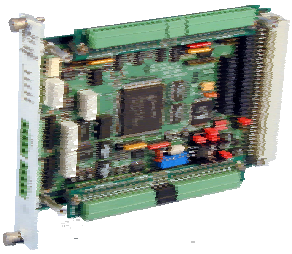
ACC-54E, USB/Ethernet interface



ACC-E1, power supply



ACC-24E2, digital interface



ACC-24E2A, analog interface



ACC-24E2S, stepper interface



ACC-11E, 24V 24 in/24 out



ACC-65E, protected 24 in/24 out



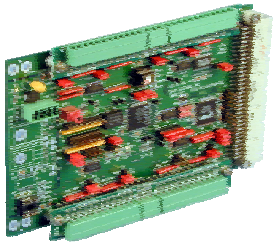
ACC-14E, 48 TTL I/O



ACC-28E, analog-to-digital converter



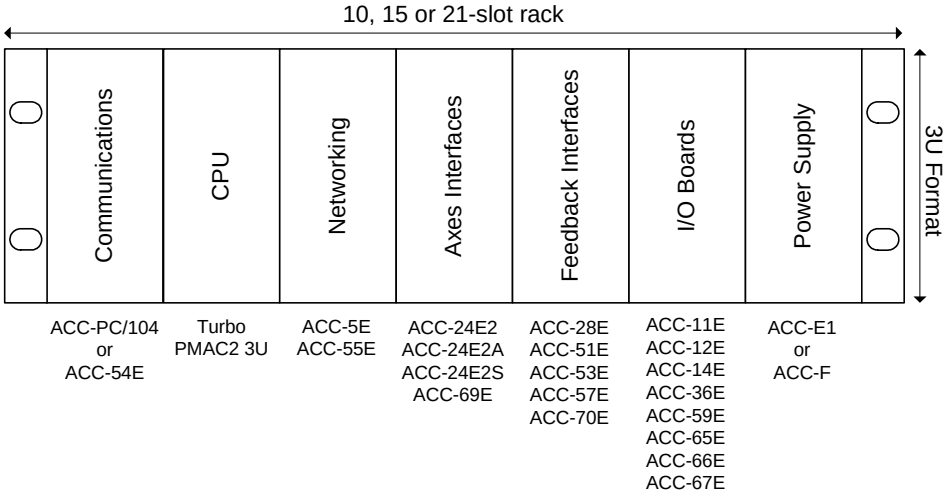
ACC-51E, sinusoidal interpolator



ACC-53E, SSI interface

Installation and Setup

When ordered from Delta Tau, the UMAC rack is provided already assembled with the selected boards internally mounted and configured properly with the appropriate addresses. The boards are installed in the rack in a particular sequence from left to right as described in the following diagram:



Hardware Setup

On some UMAC accessory boards, there are jumpers (pairs of metal prongs) called E-points. Some have been shorted together; others have been left open. These jumpers customize the hardware features of the board for a given application. Check each jumper configuration using the appropriate hardware reference for the particular accessory board being set. After all the jumpers have been properly set, the UMAC can be wired to the machine and the host computer linked with a serial cable to it.

The connections to the machine are performed directly to the UMAC rack. Terminal blocks on top, bottom and front of the rack provide the signals for the amplifiers, feedback devices and I/O points.

Software Setup

The Turbo PMAC2 3U has a set of initialization parameters (I-Variables) that determine the personality of the card for a specific application. Many of these are used to configure a motor properly. Once set up, these variables may be stored in flash memory (using the **SAVE** command) so the card is always configured properly. (PMAC loads the flash I-variable values into RAM on power up.)

The easiest way to program, set up and troubleshoot PMAC is by using the PMAC Executive Program, PEWIN32-Pro and its related add-on packages, Turbo Setup and UMAC configuration. PEWIN has the following main tools and features:

- A terminal window – This is the main channel of communication between the user and PMAC
- Watch window for real-time system information and debugging
- Position window for displaying the position, velocity and following error of all motors on the system
- Several ways to tune PMAC systems
- Interface for data gathering and plotting

In Pewin, the value of an I-Variable may be queried by typing in the name of the I-Variable. For instance, typing **I100<CR>** causes the value of the I100 to be returned. The value may be changed by typing in the name, an equals sign, and the new value (e.g. **I900=3<CR>**). Remember that if any I-Variables are changed during this setup, use the **SAVE** command before the card is powered down or reset, or the changes that were made will be lost.

Programming UMAC

Once the UMAC System is wired to the machine and the motors are properly tuned, any I/O control and motion control can be performed. There are three different ways to control I/O and motion in a UMAC System, and all these methods require Pewin Pro, the PMAC Executive Software for Windows, for communications and download to the system.

Online Commands

Online commands allow jogging motors and setting I/O points by issuing commands from the terminal window of Pewin. This mode is convenient for trying move commands and sequences to be included later in a motion or PLC program.

```
CTRL+A              ; Pressing the control and A keys together stops all motion
                    ; programs and motion commands
CTRL+D              ; Pressing the control and D keys together stops the execution
                    ; of all PLC programs
#1J^2000            ; Jogs Motor #1 2000 encoder counts in incremental mode. Press
                    ; <Enter> to execute the command.
```

Motion Programs

Motion programs are entered and downloaded using the text editor of Pewin. A motion program allows synchronizing the motion of axes and setting of I/O points with different methods of interpolation, including linear and circular interpolation.

```
CLOSE                   ; Close all open buffers
END GATHER              ; Stops gathering feature
DELETE GATHER           ; Deletes gathering buffer
UNDEFINE ALL            ; Removes all axis definitions
&1                      ; Coordinate System 1
#1->2000X               ; Motor 1 is defined as axis X with a scale factor of 2000 encoder counts
OPEN PROG 1 CLEAR       ; Open program 1 for editing
      TA100             ; Linear acceleration time is 100 msec
      TS0               ; No S-curve acceleration
      TM1000            ; Move time is 1000 msec
      INC               ; Incremental mode
      LINEAR            ; Linear interpolation mode
      X1                ; Move axis X one unit. This moves motor #1 2000 encoder counts
CLOSE
```

To run this program, type **B1R** in the terminal window.

PLC Programs

PLC programs are entered and downloaded using the text editor of PEWIN and are ideal for controlling digital I/O points, to start and stop motion programs, and to perform any function that does not require a tight axes synchronization.

```
I5 = 2                  ; Allows enabled PLCs to run
OPEN PLC 1 CLEAR        ; Open PLC 1 for editing
      P1 = P1 + 1       ; Increments variable P1 by one at each scan
      IF (P1=1000)      ; Reset P1 to zero when it reaches 1000
            P1 = 0
      ENDIF
CLOSE
```

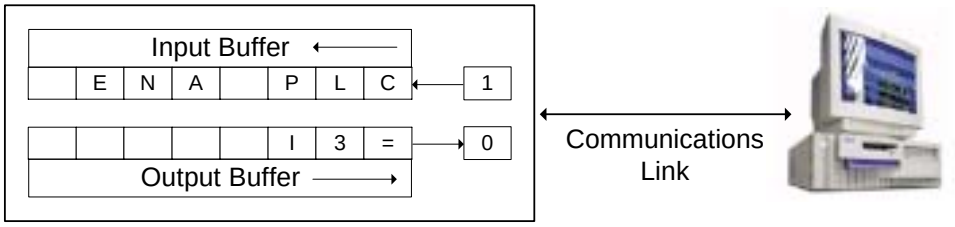
To run this program, type **ENA PLC1** in the terminal window and then press **Enter**.

UMAC Tasks

Turbo PMAC can handle all of the tasks required for machine control, constantly switching back and forth between the different tasks thousands of times per second. The major tasks involved in machine control are summarized here.

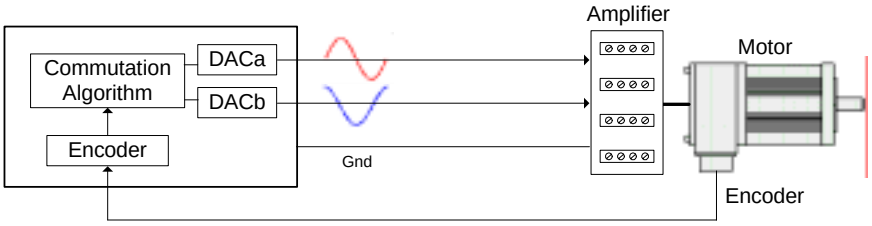
Single Character I/O

Bringing in a single character from or sending out a single character to, the serial port or host port is the highest priority in PMAC. The time this task takes is 200 nsec per character, but having it at this high priority ensures that the host cannot outrun PMAC on a character-by-character basis. This task is never a significant portion of PMAC's total calculation time. Note that this task does not include processing a full command; that happens at a lower priority (See the Background Tasks section of this guide.).



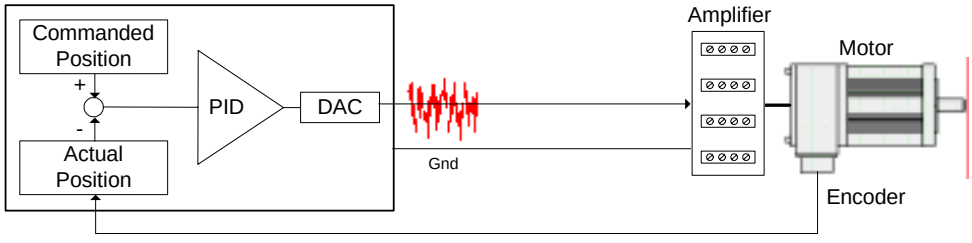
Commutation Update

If Turbo PMAC is asked to perform the commutation for a multiphase motor, it will perform commutation updates automatically at a fixed frequency (usually around 9 kHz). The commutation, or phasing, update for a motor consists of measuring and/or estimating the rotor magnetic field orientation, then apportioning the command that was calculated by the servo update among the different phases of the motor. This task occurs automatically without the need for any explicit commands.



Servo Update

In an automatic task that is essentially invisible to the Turbo PMAC user, Turbo PMAC performs a servo update for each motor at a fixed frequency (usually around 2 kHz). The servo update for a motor consists of incrementing the commanded position (if necessary) according the equations generated by the motion program or other motion command, comparing this to the actual position as read from the feedback sensor, and computing a command output based on the difference. This task occurs automatically without the need for any explicit commands.



Real-Time Interrupt Tasks

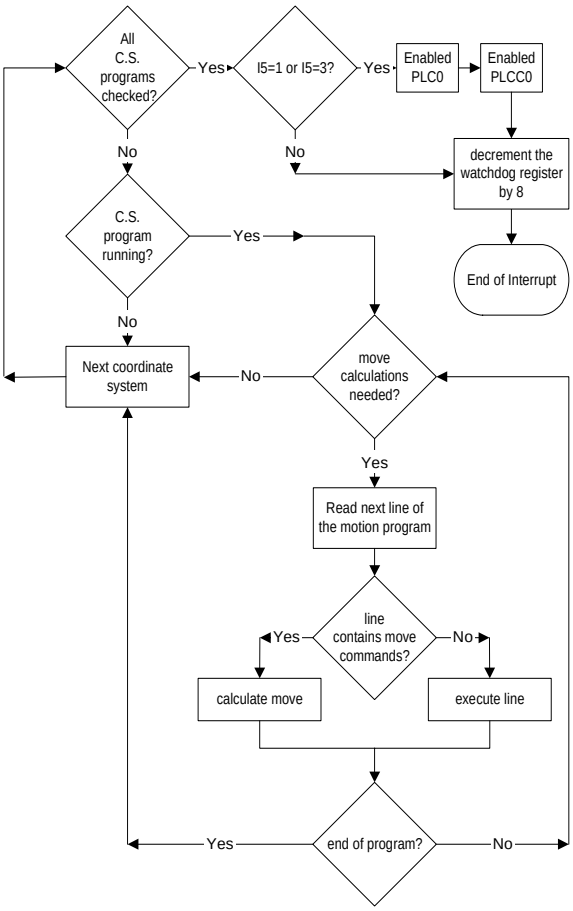
The real-time interrupt (RTI) tasks occur immediately after the servo update tasks at a rate controlled by parameter I8 (every I8+1 servo update cycles). There are two significant tasks occurring at this priority level: PLC 0 / PLCC0 and motion program move planning.

PMAC will scan the lines of each program running in the different coordinate systems and will calculate the necessary number of move commands.

The number of move commands of pre-calculation can be zero, one, or two, depending on the type of motion commands and the mode in which the program is being executed.

Non-move commands are executed immediately as they are found. The scan of any given motion program will stop as the necessary number of moves is calculated. It resumes when previous move commands are completed and more move-planning calculations are required.

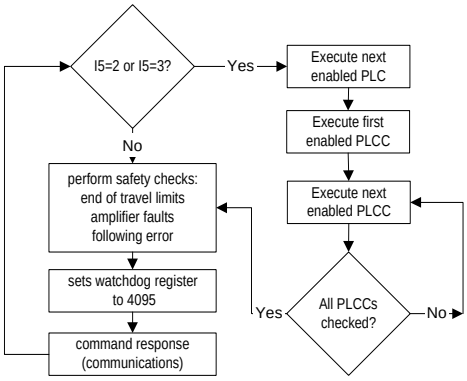
In the execution of a motion program, if PMAC finds two jumps backward (toward the top) in the program while looking for the next move command, PMAC will pause execution of the program and not try to blend the moves together. It will go on to other tasks and resume execution of the motion program on a later scan. Two statements can cause such a jump back: **`ENDWHILE`** and **`GOTO`** (**`RETURN`** does not count).



Background Tasks

During the time not taken by any of the higher-priority tasks, PMAC will be executing background tasks. There are three basic background tasks: command processing, PLC programs 1-31, and housekeeping. The frequency of these background tasks is controlled by the computational load on PMAC: the more high-priority tasks that are executed, the slower the background tasks will cycle through; and the more background tasks there are, the slower they will cycle through.

Each PLC program executes one scan (to the end or to an **`ENDWHILE`** statement) uninterrupted by any other background task (although it can be interrupted by higher priority tasks). In between each PLC program, PMAC will do its general housekeeping and respond to a host command, if any.



All enabled PLCC programs execute one scan (to the end or to an **ENDWHILE** statement) starting from lowest numbered to highest uninterrupted by any other background task (although it can be interrupted by higher priority tasks). At power-on\reset, PLCC programs run after the first PLC program runs.

The receipt of a control character from any port is a signal to PMAC that it must respond to a command. The most common control character is the carriage return (**<CR>**) which tells PMAC to treat all the preceding alphanumeric characters as a command line. Other control characters have their own meanings, independent of any alphanumeric characters received. Here PMAC will take the appropriate action to the command, or if it is an illegal command, it will report an error to the host.

Between each scan through each background PLC program, PMAC performs its housekeeping duties to keep itself properly updated. The most important of these is the safety limit checks (following error, overtravel limit, fault, watchdog, etc.) Although this happens at a low priority, a minimum frequency is ensured because the watchdog timer will trip, shutting down the card, if this frequency gets too low.

PMAC EXECUTIVE PROGRAM, PEWIN32PRO

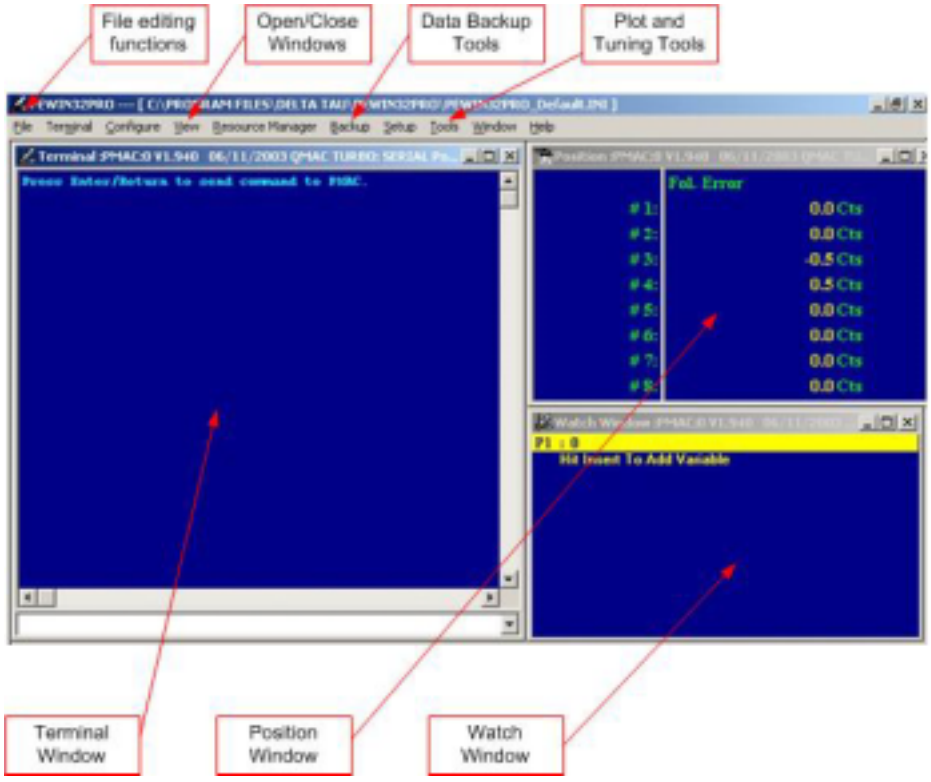
Pewin32 PRO is the PMAC Executive program for Microsoft Windows®. It is an environment rich with software tools for the development and maintenance of any application using the PMAC motion controller. These tools allow the optimization of the servo parameters to achieve maximum motor speed and accuracy and permit the customization of the motion and PLC programs inside PMAC for the application requirements. All types of communications methods are implemented for all the available communication ports, delivering a robust and reliable interchange of data with either single or multiple PMACs. A set of diagnosis tools is also available for displaying variables values, monitoring connector and motor status, and plotting motion profiles. The capability to define projects allows combining sets of files and configurations for an easy reference to each particular application.

Configuring PEWIN

Establishing Communications

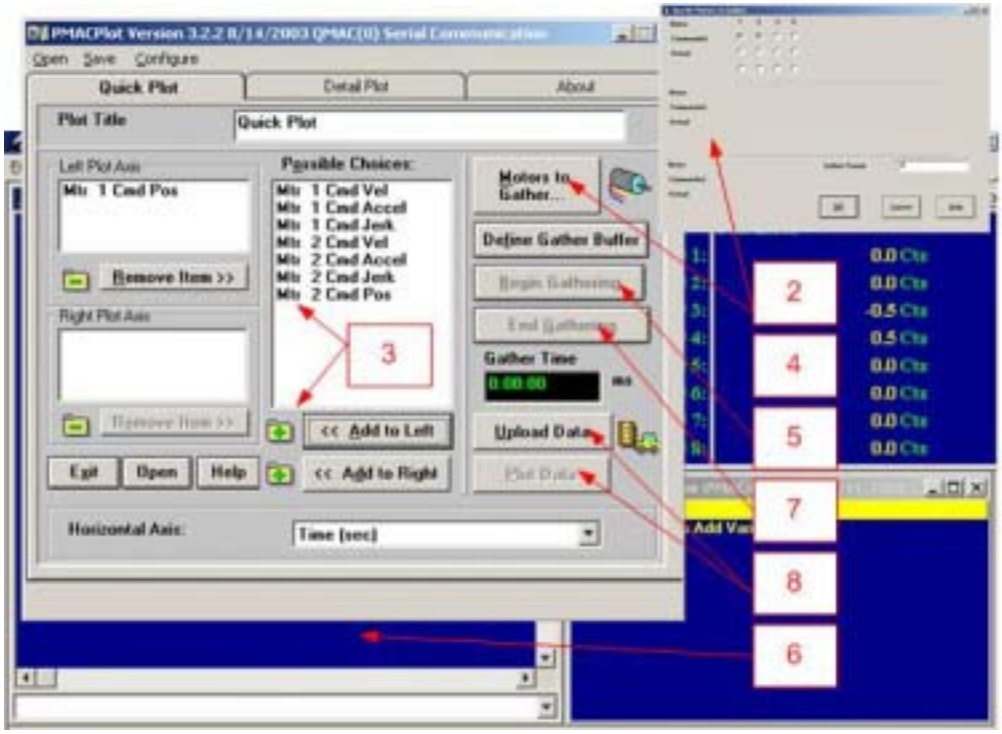
The UMAC System can communicate with the host computer using several different communication methods. This includes serial RS232, USB, Ethernet and PC/104 bus. The Pewin32 Pro installation utility includes a PDF document describing the steps required to establish communications and complete the installation process.

Workspace Layout



Quick Plot Feature

1. To run the quick plot feature, select **PMAC Plot Pro** from the **Tools** menu.
2. Select the motors and the feature to gather.
3. Select what to plot from the possible choices, and then click **Add to left** or **Add to right**.
4. Click the **Define Gather Buffer** button.
5. Click the **Begin Gathering** button.
6. Click on the terminal part of the screen and run the motion program or Jog command.
7. Click the **End Gathering** button when the motion is completed.
8. First, Click the **Upload Data** button, and then the **Plot Data** button.



The plot feature is based on the PMAC data gathering functions. It is useful for analyzing motion profiles and trajectories. For example, when using circular interpolation, the horizontal and vertical axes can plot the two motors involved. Plotting the two axes together is an important aid for understanding the set of parameters involved in a circular interpolation move.

Saving and Retrieving PMAC Parameters

It is important to save the complete set of PMAC parameters in the host computer periodically. In case of a failure or replacement, a single file created this way will allow restoring all the variables and programs necessary for the particular application. To activate this function, select **Upload Configuration** from the Backup menu. After the file is saved, verify it with the feature part of the same menu. This will confirm if the memory contents in PMAC matches the recently saved file, thus confirming a valid restoring file. To restore a configuration, select **Restore Configuration** from the same Backup menu. In addition, select **Verify Configuration** after the restore function is completed.

The Watch and Position Windows

The position window is accessed through the **Position** command from the View menu. It is a convenient way to continuously check PMAC parameters such as position velocity and following error. Using the right button of the mouse on this window checks the item selections as well as its format and update period.

The Watch function of the same View menu performs a similar function. It allows the constant display of any variable value in PMAC. Right clicking on this window allows selecting the display format from hexadecimal, decimal and binary reporting values.

Uploading and Downloading Files

These functions are accessible through the File menu. The uploading function is of great importance and allows the opening of a text editor with the contents of the requested PLC, Motion Program, M-Variables definitions or values, I-Variable values etc. This allows checking not only what commands or values PMAC has actually in memory, but also will indent **IF** conditions and **WHILE** loops, making the program flow more readable.

Using MACRO Names and Include Files

PEWIN allows using custom names in place of the common names for variables and functions that PMAC expects (P, Q, M, I):

Example:

File Downloaded

```
#define PUMP P1
OPEN PLC1 CLEAR
      PUMP = 1
      DISABLE PLC1
CLOSE
```

Uploaded Translated PMAC Code

```
OPEN PLC 1 CLEAR
      P1 = 1
      DISPLC1
CLOSE
```

The MACRO name must be defined before it can be used. In general, MACRO definitions are at the beginning of the text file to be downloaded. MACROs must be up to 255 valid ASCII characters and cannot have spaces in between (the underscore "_" is suggested in place of a space). The MACRO definitions, or any other PMAC code, can be placed in a separate file and included with a single line in the text file to be downloaded. The file name must consist of a full path in order for PEWIN to find it.

Example: `#include "c:\deltatau\files\any.pmc`

Downloading Compiled PLCCs

PLCCs are compiled by Pewin in the downloading process. Only the compiled code is downloaded to PMAC. Therefore, save the ASCII source code in the host computer separately since it cannot be retrieved from PMAC. In most cases, compiled PLCs are firmware dependent and must be recompiled when the firmware is changed in PMAC. If more than one PLCC is defined, all the PLCC code must belong to the same ASCII text file. Pewin will compile all the PLCC code present in the file and place it in the appropriate buffer in PMAC. If a single PLCC code is downloaded, all the other PLCCs that might have been present in memory will be erased, leaving only the last compiled code.

The PID Tuning Utility

This function is accessible by selecting PMAC Tuning Pro from the Tools menu. The Auto tuning feature allows finding the PID parameters with virtually no effort from the user. In most cases, the parameters are very close to optimal and in some cases require further fine-tuning by the user. In this screen, press the Page-Up or Page-Down keys on the keyboard to select the motor number.



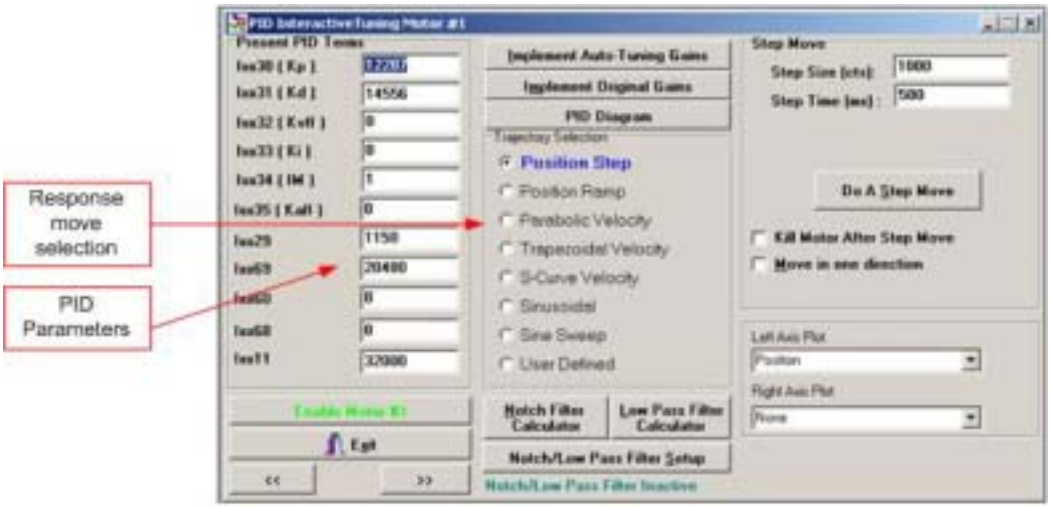
Auto Tuning

In most cases, the motors can be controlled in closed loop with a relatively small following error by simply increasing the proportional gain parameter, Ixx30, from its default value. As a rule of thumb, slowly increase the proportional gain variable until a buzzing noise in the motor is heard, and then back down 20% from that value. The auto-tuning utility provides a more efficient method of getting the motors to move in close loop with minor effort.

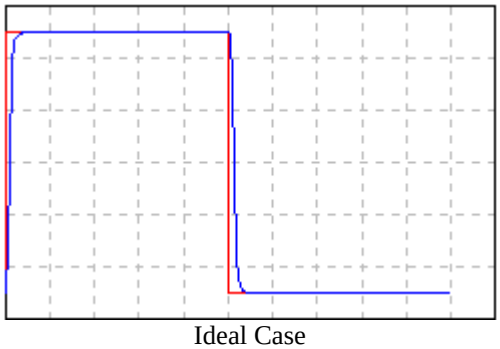


1. Make sure to read the Pewin manual section related to the safety issues of this procedure.
2. Select the type of amplifier being tuned.
3. Let the Auto Tune select the bandwidth by checking the **Auto Select Bandwidth** box.
4. Select the **Velocity Feed Forward** or **Acceleration Feed Forward** boxes as necessary.
5. Select the **Integral Action** box if necessary.
6. Start the Auto Tuning interaction by clicking the **Auto Tune** button. Most likely, the motor will move after this is clicked.

Interactive Tuning

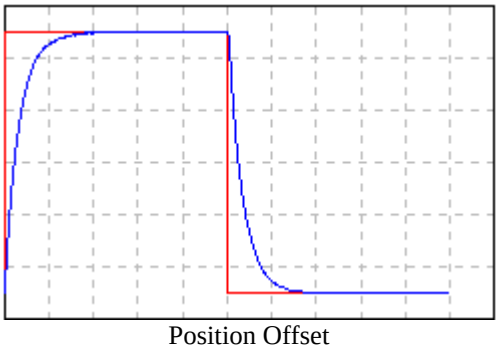


After the Auto Tuning is completed, the PID parameters can be changed for a final fine-tuning approach. Perform a step response and use the following guidelines for the selection of the appropriate I-Variables:



Ideal Case

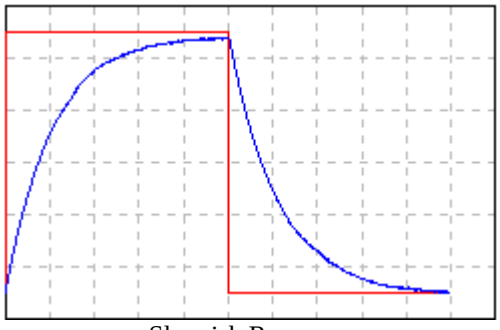
The motor closely follows the commanded position



Position Offset

Cause: friction or constant force/system limitation

Fix: Increase K_I (Ix33) and maybe use more K_P (Ix30)



Sluggish Response

Cause: Too much damping or too little proportional gain

Fix: Increase K_P (Ix30) or decrease K_D (Ix31)

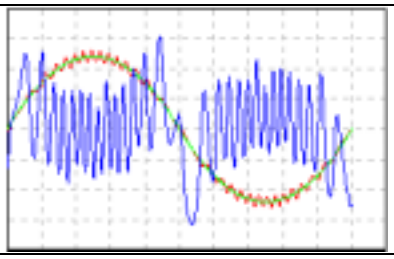


Overshoot and Oscillation

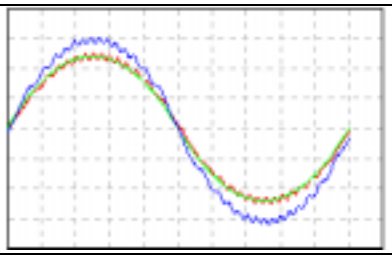
Cause: Too much damping or too little proportional gain

Fix: Increase K_P (Ix30) or decrease K_D (Ix31)

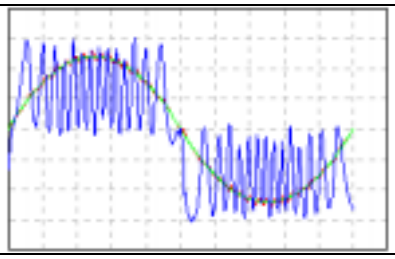
Perform a parabolic move and use the following guidelines for the selection of the appropriate I-Variables:



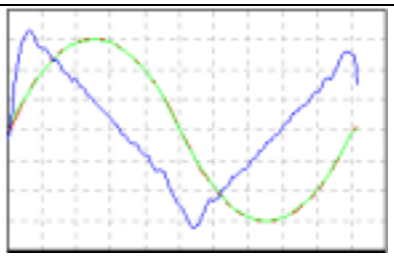
Ideal Case
The following error is reduced at minimum and is concentrated in the center, evenly along the move



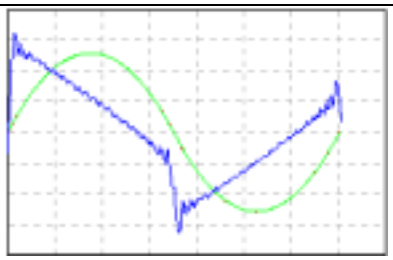
High vel/FE correlation
Cause: damping
Fix: Increase K_{vel}(Ix32)



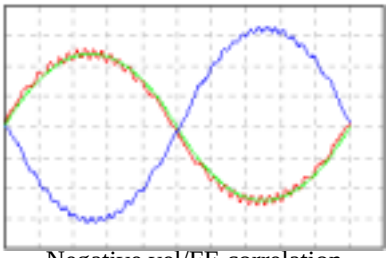
Negative vel/FE correlation
Cause: friction
Fix: Increase Integral gain (Ix33) or Friction Feedforward (Ix68)



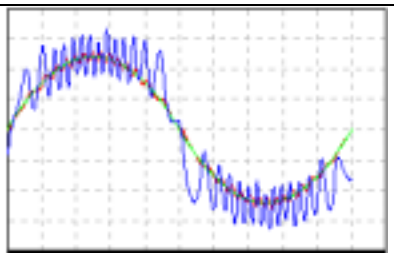
High acc/FE correlation
Cause: Integral lag
Fix: Increase K_{aff}(Ix35)



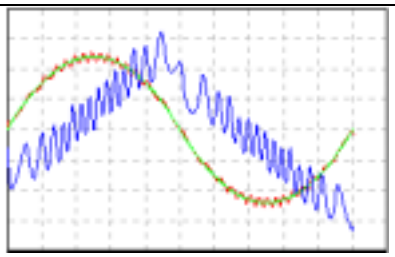
High acc/FE correlation
Cause: Physical system limitations
Fix: Use less sudden acceleration



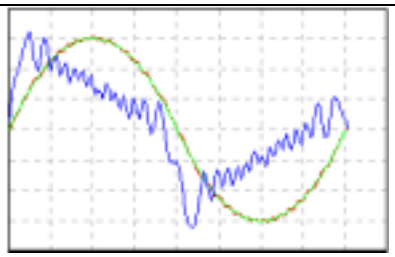
Negative vel/FE correlation
Cause: Too much velocity FF
Fix: Decrease K_{vel}(Ix32)



High vel/FE correlation
Cause: damping and friction
Fix: Increase K_{vel}(Ix32)



High acc/FE correlation
Cause: Too much acc FF
Fix: Decrease K_{aff}(Ix35)



High vel/FE and acc/FE correlation
Cause: Integral lag and friction
Fix: Increase K_{aff}(Ix35)

Other Features

- Turbo Setup32 Pro provides a step-by-step method for configuring any Turbo PMAC-type motion controller
- UMAC Config Pro provides a method for checking the hardware configuration of any existing UMAC rack
- Workspace support that allows saving all the working environment settings for next session restore (e.g., the number of windows open, their corresponding sizes and update rate)
- Project management for combining sets of files and configurations for any given application
- Organizer feature that allows sorting, setting and checking all the I, P, Q and M-Variables
- Motor, Coordinate System and Global status windows that display PMAC's status bits in real-time
- Methods for the configuration of the encoder conversion table
- Real-time status display of all PMAC's connectors
- Diagnostic routines for checking the functionality of motors and motion programs
- A real-time color text editor for PMAC motion and PLC programs

HARDWARE SETUP AND CONNECTIONS

Address Configuration

When ordered from Delta Tau, the UMAC rack is provided already assembled with the selected boards internally mounted and properly configured with the appropriate addresses. The address selection for each accessory is necessary when replacing a board or when adding a new board in the UMAC System. The System Configuration Reporting I-Variables, I4900 to I4965, provide information about the accessory boards found inside the UMAC rack on power-up or reset. The UMAC Configuration program of the Pewin32 Pro Suite uses these variables to report the configuration of any UMAC System.

Note:

The E1 jumper on the back of the Acc-Ux UBUS backplane board must be on to use the DIP-switch addressing.

Servo Cards

The typical UMAC System will use up to eight different locations to address the servo cards. These are set with DIP-switches according to the following table. The corresponding manual for each product will indicate if it uses a servo address and the switches configuration for each particular address.

S1	S2	S3	S4	S5	S6	Board #	I-Variables Range	Base Address
ON	ON	ON	ON	ON	ON	1	I7200-I7249	$078200
OFF	ON	ON	ON	ON	ON	2	I7300-I7349	$078300
ON	ON	OFF	ON	ON	ON	3	I7400-I7449	$079200
OFF	ON	OFF	ON	ON	ON	4	I7500-I7549	$079300
ON	ON	ON	OFF	ON	ON	5	I7600-I7649	$07A200
OFF	ON	ON	OFF	ON	ON	6	I7700-I7749	$07A300
ON	ON	OFF	OFF	ON	ON	7	I7800-I7849	$07B200
OFF	ON	OFF	OFF	ON	ON	8	I7900-I7949	$07B300

Note:

Only one servo type board must source the servo clock lines in a given UMAC System. It is configured through a jumper setting. Consult the particular accessory manual for details.

IO Cards

Each IO card in a given UMAC System must have a unique address and this is set with DIP-switches. The following table shows the settings for the first four IO type cards. The corresponding manual for each product will indicate if it uses an IO address and the switches configuration for each particular address.

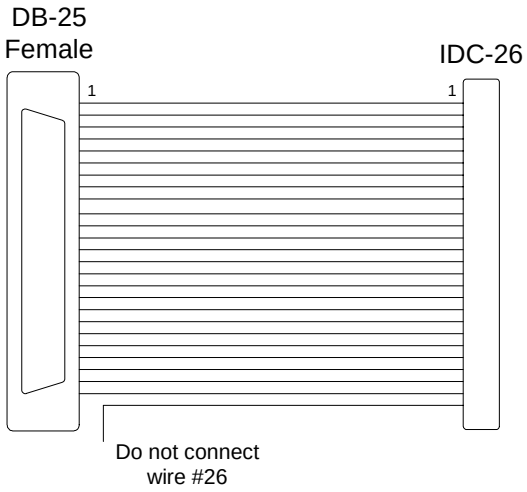
S1	S2	S3	S4	S5	S6	Board #	Address Range
ON	ON	ON	ON	ON	ON	1	Y:$078C00 to Y:$078C03
ON	ON	OFF	ON	ON	ON	2	Y:$079C00 to Y:$079C03
ON	ON	ON	OFF	ON	ON	3	Y:$07AC00 to Y:$07AC03
ON	ON	OFF	OFF	ON	ON	4	Y:$07BC00 to Y:$07BC03

Some accessories can use only a limited range of addresses, but have a set of jumpers to select which byte of the assigned address space is actually used. The following table shows an example that uses this addressing scheme.

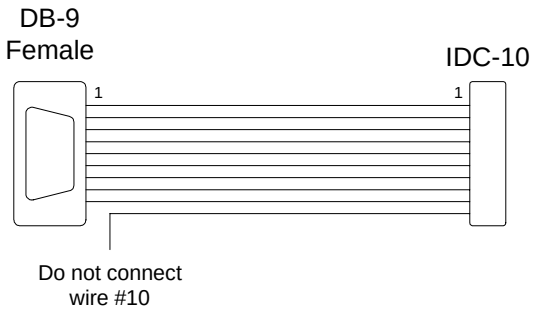
ACC-11E Jumper Settings		
Jumpers	**Setting**	**Bits used from base address**
E6A- E6H	1-2	Uses bits 0-7 from six consecutive memory locations (low byte)
E6A- E6H	2-3	Uses bits 8-15 from six consecutive memory locations (middle byte)
E6A- E6H	4-5	Uses bits 16-23 from six consecutive memory locations (high byte)

Serial Port Connections

For serial communications, use a serial cable to connect the PC's COM port to the Turbo PMAC2's 3U serial port connector. The Acc-3D cable provided connects to the Turbo PMAC2's 3U serial port with a DB-25 connector. Standard DB-9-to-DB-25 or DB-25-to-DB-9 adapters may be needed for a particular set up. The simplest way to make such a cable is to use a flat cable prepared with flat-cable type connectors as indicated in the following diagram:



If the auxiliary serial port is present, it will be provided through an IDC-10 connector. In this case, the Acc-3L cable provided by Delta Tau connects to the Turbo PMAC2's 3U auxiliary serial port with a DB-9 connector. Standard DB-9-to-DB-25 or DB-25-to-DB-9 adapters may be needed for a particular setup. The simplest way to make such a cable is to use a flat cable prepared with flat-cable type connectors as indicated in the following diagram:



Serial communications can be checked using the Windows® HyperTerminal program with 38,400 baud rate, eight data bits, one stop bit, no parity and no flow control. In this mode, set I3=1 to add a carriage return at the end of each response line.

Re-initializing UMAC

After communication is established, re-initialize UMAC for first-time use by sending the **$$$***** command in the terminal window. This command will erase all programs and reset all variables to factory defaults.

Power Supply

The typical UMAC System is provided with the internally mounted ACC-E1 power supply that can accept an AC input from 85VAC to 240VAC, and output DC voltages with up to 14A at +5V, and 1.5A each at ±15V. In this case, a standard computer type IEC/EIA male connector is present in the back panel and any regular computer type cord can be used for the power connection. In addition, a connector in the back panel provides the output power supply lines of +5V, ±15V and ground. These lines can be used to power the flags opto-isolation circuitry in case no external power supply is used.

Motor Flag Connections

When assigned for the dedicated uses, the overtravel limit flags provide important safety and accuracy functions. PLIMn and MLIMn are direction-sensitive over-travel limits that must conduct current (either sinking or sourcing) to permit motion in that direction. The home input flag is used in conjunction with home search type moves to establish a machine point of reference when an incremental type of feedback is used. The user input flag is used mostly in conjunction with the position capture feature, which allows recording the feedback information when the input is activated.

Disabling the Overtravel Limit Flags

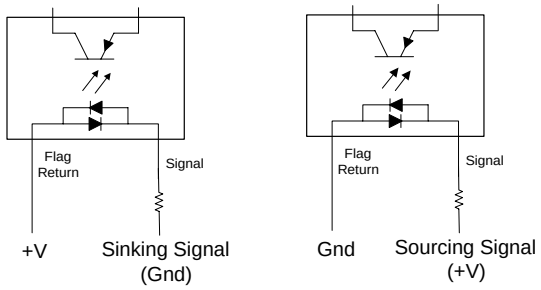
If no overtravel switches are connected to the particular motor, set bit 17 of the Ixx24 variable to 1 to disable this feature.

Example:

```
I124 = $20001            ; Disables Overtravel Limits of Motor #1
```

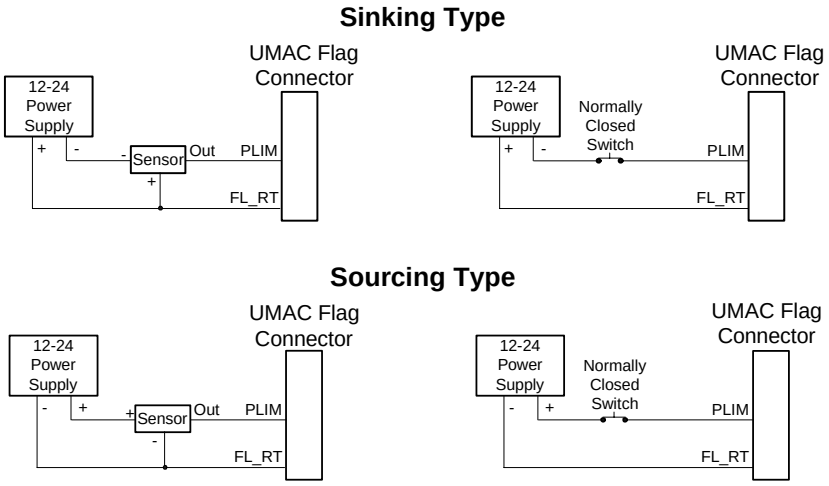
Types of Overtravel Limits

The UMAC axes boards, ACC-24Ex, have a bipolar opto-isolating circuitry (chip PS-2705-4NEC) for the flag connections. Conveniently, this allows using either a sinking or a sourcing sensor in the 5V or 12 to 24V range. This includes proximity sensors and dry (passive) normally closed contacts. If the use of 5V flags is desired, a 1kΩ SIP resistor pack (1KSIP8I) should be installed in the appropriate resistor socket onboard. In this case, the flags' opto-isolation circuits will be powered with a 5V power supply instead. Consult the particular accessory manual for details.



UMAC Flag Inputs Circuit

Example:
These examples show the connection of the most common types of end-of-travel sensors. Instead of the external power supply shown here, the power can be supplied from the back panel of the UMAC System.



Home Sensors

The location of the home sensors establishes a point of reference in the machine from which each move is related. When using incremental types of feedback, a home search type of move must be performed after each power-up or reset cycle.

In contrast with the overtravel limit inputs, the home inputs do not need to conduct current to allow motion. However, use the same type of sensors for both the limits and home inputs.

Note:

If a hardware flag is used for home reference and a quadrature encoder is used for feedback, they both must belong to the same hardware channel in the axis board.

Checking the Flag Inputs

In the Pewin terminal window, define the following M-Variables for the flags of the motors under consideration:

Flag Type	Motor #1	Motor #2	Motor #3	Motor #4
HMFL input status	M120->X:$78200,16	M220->X:$078208,16	M320->X:$078210,16	M420->X:$078218,16
PLIM input status	M121->X:$78200,17	M221->X:$078208,17	M321->X:$078210,17	M421->X:$078218,17
MLIM input status	M122->X:$78200,18	M222->X:$078208,18	M322->X:$078210,18	M422->X:$078218,18
Flag Type	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
HMFL input status	M520->X:$78300,16	M620->X:$078308,16	M720->X:$078310,16	M820->X:$078318,16
PLIM input status	M521->X:$78300,17	M621->X:$078308,17	M721->X:$078310,17	M821->X:$078318,17
MLIM input status	M522->X:$78300,18	M622->X:$078308,18	M722->X:$078310,18	M822->X:$078318,18

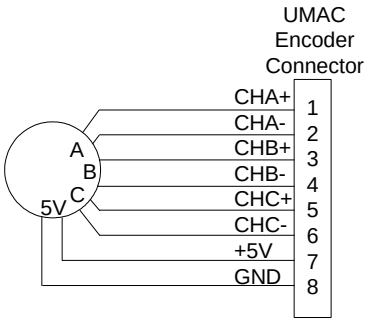
Open a Watch Window and click **Insert** to enter the appropriate M-Variable to watch. Interacting with the switch or sensor, monitor a change of value in the corresponding M-Variable. A value of zero indicates that the flag is closed or conducting current, so the motor will be able to run in that direction (see Ixx24). If the value is 1, the flag is open instead.

Motor Signals Connections

Incremental Encoder Connection

The encoder connectors in the ACC-24Ex type boards provide all the signals for a TTL quadrature incremental encoder type. Connect the A and B (quadrature) encoder channels to the appropriate terminal block pins. If it is a single-ended signal, leave the complementary signal pins floating – do not ground them. For a differential encoder, also connect the complementary signal lines. The third C channel (index pulse) is optional, and it is used mostly for a more accurate home search procedure. Jumpers on the Acc-24E2S select between amplifier enable outputs and encoder C channel inputs. An encoder loss circuitry is available in most axes boards; refer to the appropriate accessory hardware reference for details.

Example:

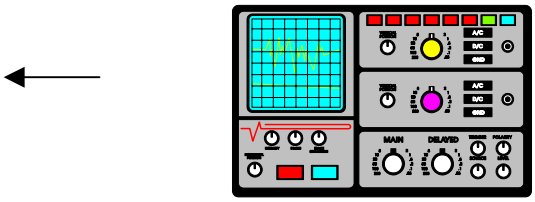


Checking the Encoder Inputs

Once the encoders have been properly wired, it is important to check their functionality and its polarity. Make sure the motor is not powered while performing this test. Activate the appropriate motor xx by setting variable Ixx00 = 1. Then, in Pewin, open a position window by selecting **Position** in the View menu. Rotate the encoder monitor to the corresponding position values. Make sure that a rotation in the positive direction increments the position value, otherwise change variable I7mn0 (I7210 for motor 1 of the first axes board) between values 3 or 7. Also, make sure that the number of counts of resolution matches the number read by PMAC when moving the appropriate distance. If necessary, for troubleshooting purposes, place an oscilloscope in the encoder inputs to check the functionality of the encoder signals.

Example:

- Channel A is pin 1 of the encoders connector
- Channel B is pin 3 of the encoders connector
- Ground is pin 8 of the encoders connector



MLDT Feedback Connection

Any channel of an Acc-24E2, Acc-24E2A or Acc-24E2S that is not being used for digital PWM or stepper PFM signals can be set up to interface an MLDT position feedback device. In most cases, MLDT position feedback devices are used with analog ±10V amplifiers. See the connections example at the end of this section for details.

DAC Output Signals

Acc-24E2A provides the ±10V DAC signals for analog type motors. If PMAC is not performing the commutation for the motor, only one analog output is required to command the motor. This analog output can be either single-ended or differential, depending on what the amplifier is expecting. For a single-ended command, connect DACA+ (pin 1) to the command input on the amplifier. Connect the amplifier's command signal return line to GND line (pin 12). In this setup, leave the DACA- pin floating; do not ground it. For a differential command, connect DACA+ (pin 1) to the plus command input on the amplifier. Connect DACA- (pin 2) to the minus command input on the amplifier. The GND line should still be connected to the amplifier common.

If using PMAC to commutate the motor, use two analog outputs for the motor. In this case, the DACB+ and DACB- lines provide the second DAC output. Each output may be single-ended or differential, as for the DC motor.

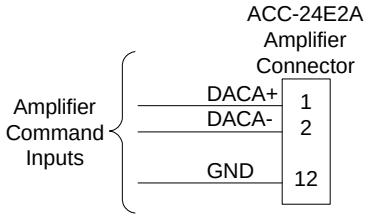
To limit the range of each signal to ±5V, use parameter Ixx69. Sign-and-magnitude mode, the output of a 0-10V and sign signals is not available in the UMAC System.

There are two options to power the Acc24-E2A DAC circuitry. If the UMAC internal power supply is used (default), jumpers E85, E87, and E88 in the ACC-24E2A board must be installed. In this case, no external power supply should be connected to the analog power terminal block of the Acc-24E2A board. If an external power supply is used, jumpers E85, E87, and E88 must be removed.

Note:

Before using the analog DAC signals, the output of the corresponding motor must be configured accordingly. This is accomplished by setting the I-Variable I7mn6=3 (I7216=3 for the first motor of the first axes board).

Example:



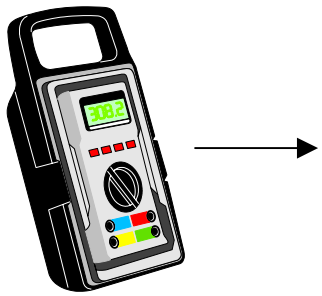
Checking the DAC Outputs

Warning:

Make sure the amplifier is not powered while performing this test.

Before powering the amplifier, check the DAC outputs operation. In the Pewin terminal window, define the following M-variables for the DACs of the motors under consideration:

	Motor #1	**Motor #2**	**Motor #3**	**Motor #4**
DAC output	M102->Y:$78202,8,16,S	M202->Y:$7820A,8,16,S	M302->Y:$78212,8,16,S	M402->Y:$7821A,8,16,S
	Motor #5	**Motor #6**	**Motor #7**	**Motor #8**
DAC output	M502->Y:$78302,8,16,S	M602->Y:$7830A,8,16,S	M702->Y:$78312,8,16,S	M802->Y:$7831A,8,16,S



Example for DAC #1.

Type the following in the terminal window:

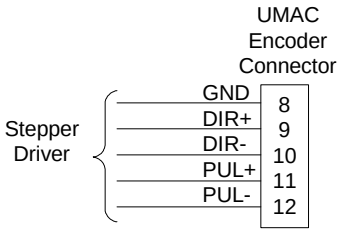
```
M102->Y:$078202,8,16,S
I100=0
I7216 = 3
M102=16383
```
<measure 5V between pins 1 and 12 of the amplifier connector>
```
M102=-16383
```
<measure -5V between pins 1 and 12 of the amplifier connector>
```
I100=1
```

Pulse and Direction Stepper Signals

Typically, the pulse and direction signals to control stepper drivers are provided by the Acc-24E2S board. However, either Acc-24E2A or Acc-24E2 can be used for this purpose also. This is the case in applications where stepper drivers, analog amplifiers and digital amplifiers are controlled with the same UMAC System. Regardless of the accessory used for connections, the setup is the same. A set of jumpers select between the pulse and direction outputs and the T, U, V and W hall-effect inputs. The signals are differential at TTL levels and are brought to the encoder connector.

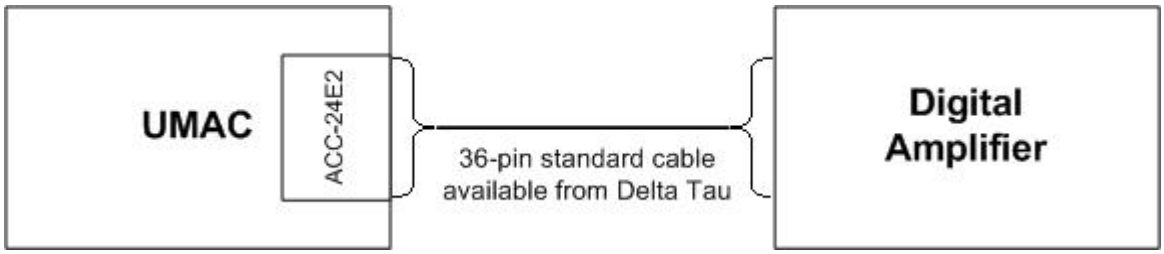
Note:

Before using the pulse and direction signals, the output of the corresponding channel and motor must be configured accordingly. This is accomplished through variables Ixx02, I7m03, I7m04, I7mn6, I7mn7, and I7mn8.



Digital Amplifier Connections

ACC-24E2 provides the necessary signals for direct PWM digital control. These signals are brought through a standard 36-pin Mini-D connector and are the direct PWM control lines, current feedback lines, and amplifier enable\fault lines. Typically, a connection from UMAC to these types of amplifiers is performed using a standard cable.

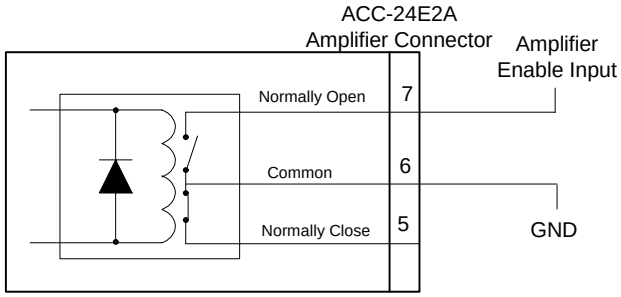


Amplifier Enable Signals

Most amplifiers have an enable/disable input that permits complete shutdown of the amplifier, regardless of the voltage of the command signal. UMAC's AENA line is meant for this purpose. For early tests, this amplifier signal should be under manual control. For troubleshooting purposes, the amplifier enable signal can be controlled manually by setting Ixx00=0 and using the properly defined Mxx14 variable.

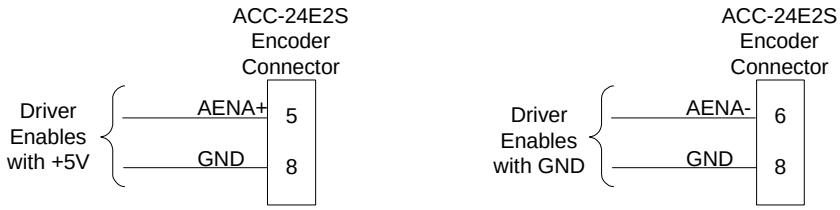
To control the amplifier enable function, the Acc-24E2A is provided with a relay with normally closed and normally open contacts. Typically, the required amplifier enable signal will be passed through the normally open contact.

Example: The amplifier is connected to the Acc-24E2A and enables with a ground connection.



Both Acc-24E2 and Acc-24E2S provide a differential amplifier enable signal at TTL levels. Jumpers on the Acc-24E2S select between amplifier enable outputs and encoder C channel inputs. In the Acc-24E2, the 36-pin amplifier connector brings the necessary amplifier enable signals automatically. To use the driver enable outputs in the Acc-24E2S, the appropriate jumpers must be set accordingly.

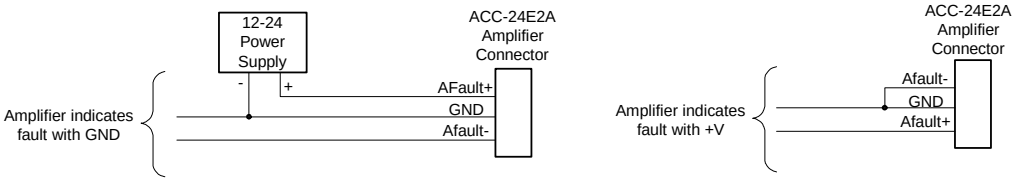
Example: These examples show the connection of single-ended stepper driver enable signals.



Amplifier Fault Signals

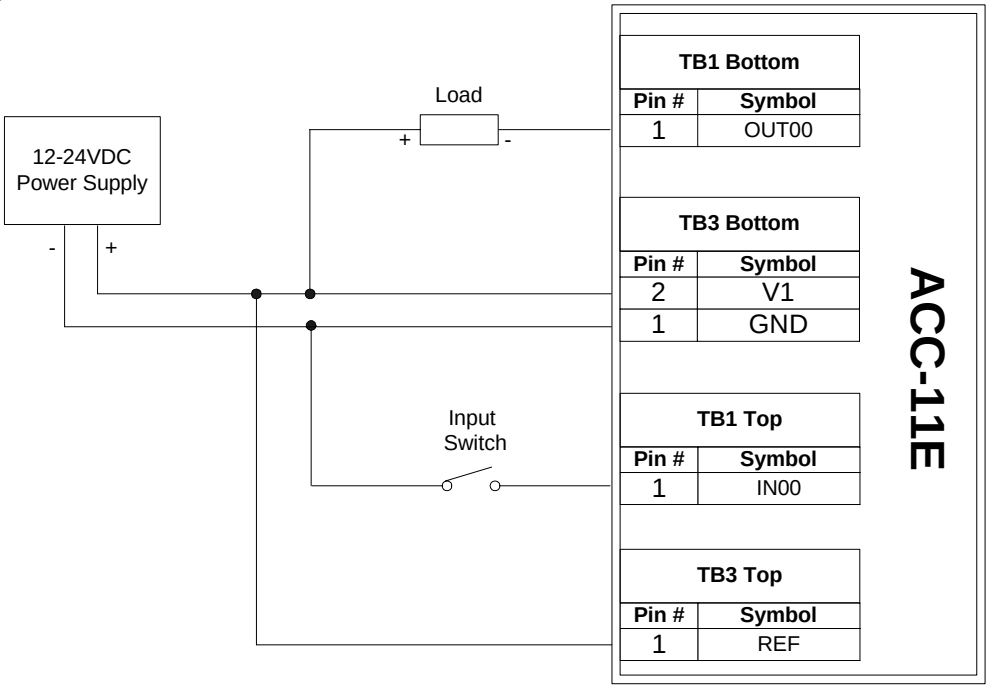
These inputs, available only on Acc-24E2 and Acc-24E2A, can take a signal from the amplifier so PMAC knows when the amplifier is having problems and can shut down action. The polarity is programmable with I-Variable Ixx24 (I124 for motor #1). The amplifier fault input is differential, but it can be used with single-ended type signals also. In the Acc-24E2, the 36-pin amplifier connector brings the necessary amplifier fault signals automatically. The amplifier fault signal can be monitored using the properly defined Mxx23 variable.

Examples: These examples show the connection of the single-ended amplifier fault signals. Instead of the external power supply shown here, the power can be supplied from the back panel of the UMAC System.

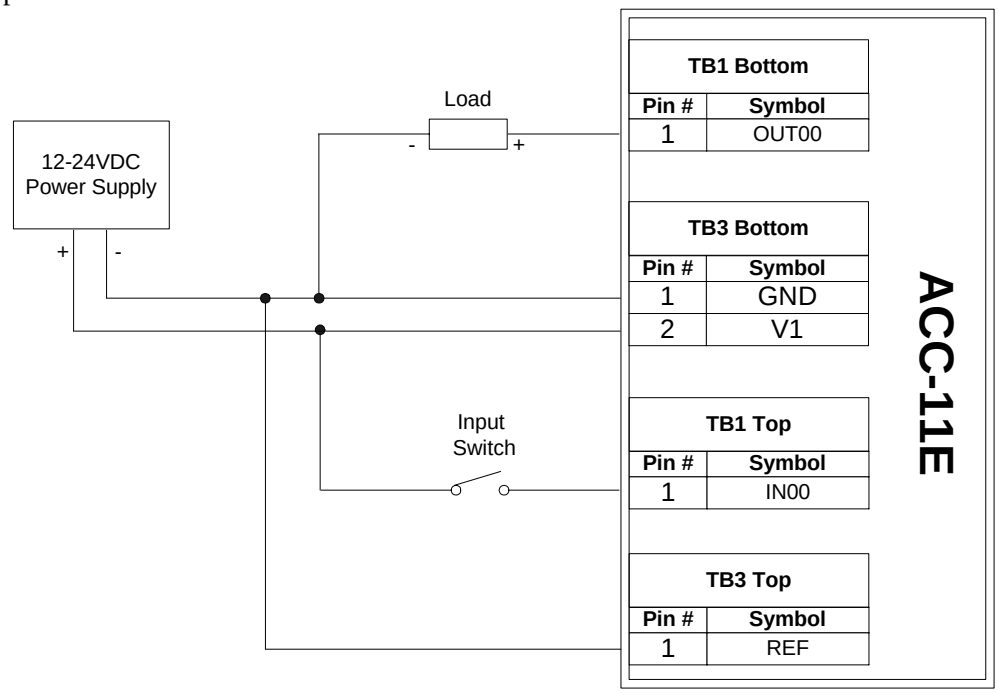


Digital Inputs and Outputs

This example shows the typical connection of an ACC-11E digital I/O board with sinking inputs and sinking outputs. The Acc-11E must be ordered with the appropriate output chips for either sinking or sourcing operation.



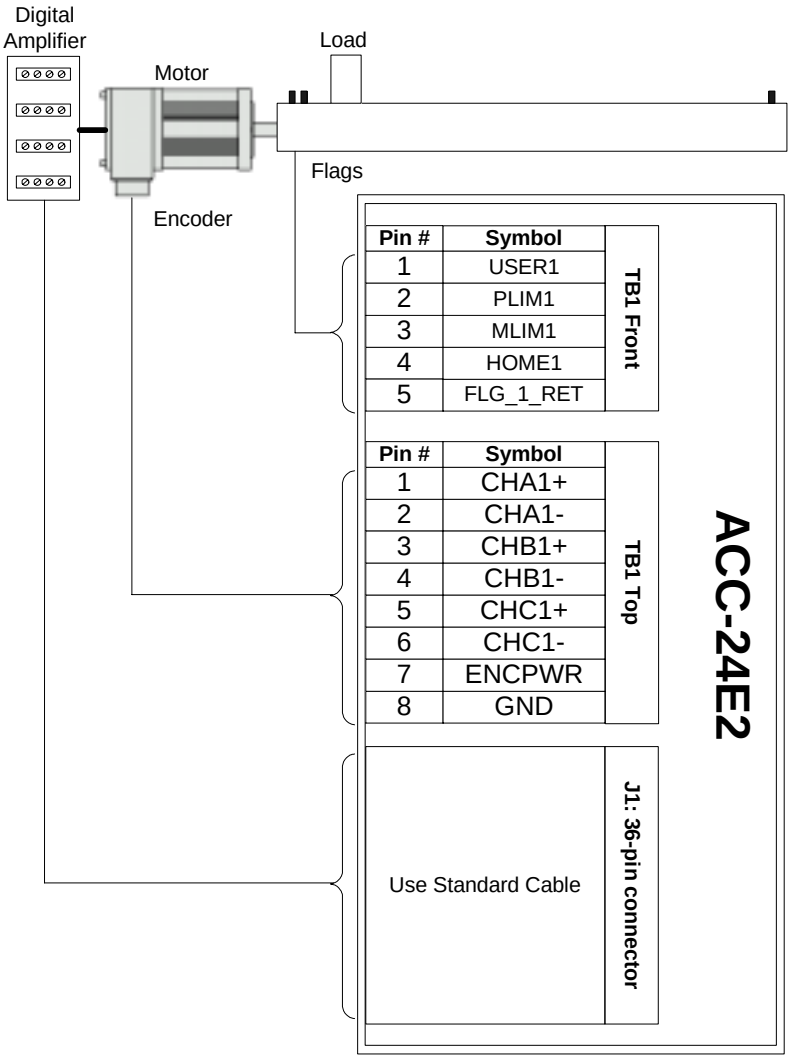
This example shows the typical connection of an Acc-11E digital I/O board with sourcing inputs and sourcing outputs. The Acc-11E must be ordered with the appropriate output chips for either sinking or sourcing operation.



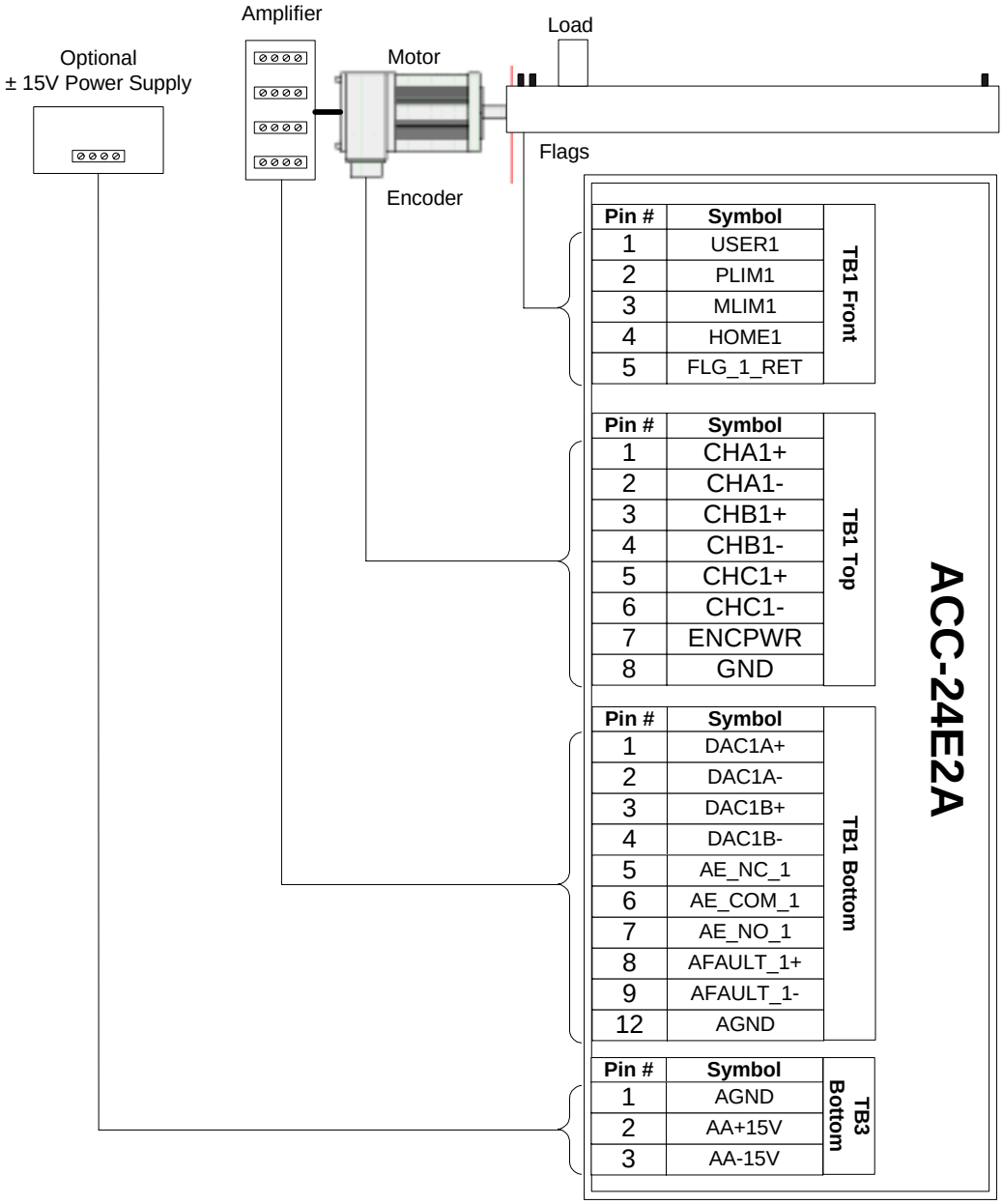
These examples can be applied to other IO accessory types. However, in some cases, the polarity of the TB3 Bottom power connector might be reversed from what is shown here.

Connection Examples

Digital Amplifier with Incremental Encoder

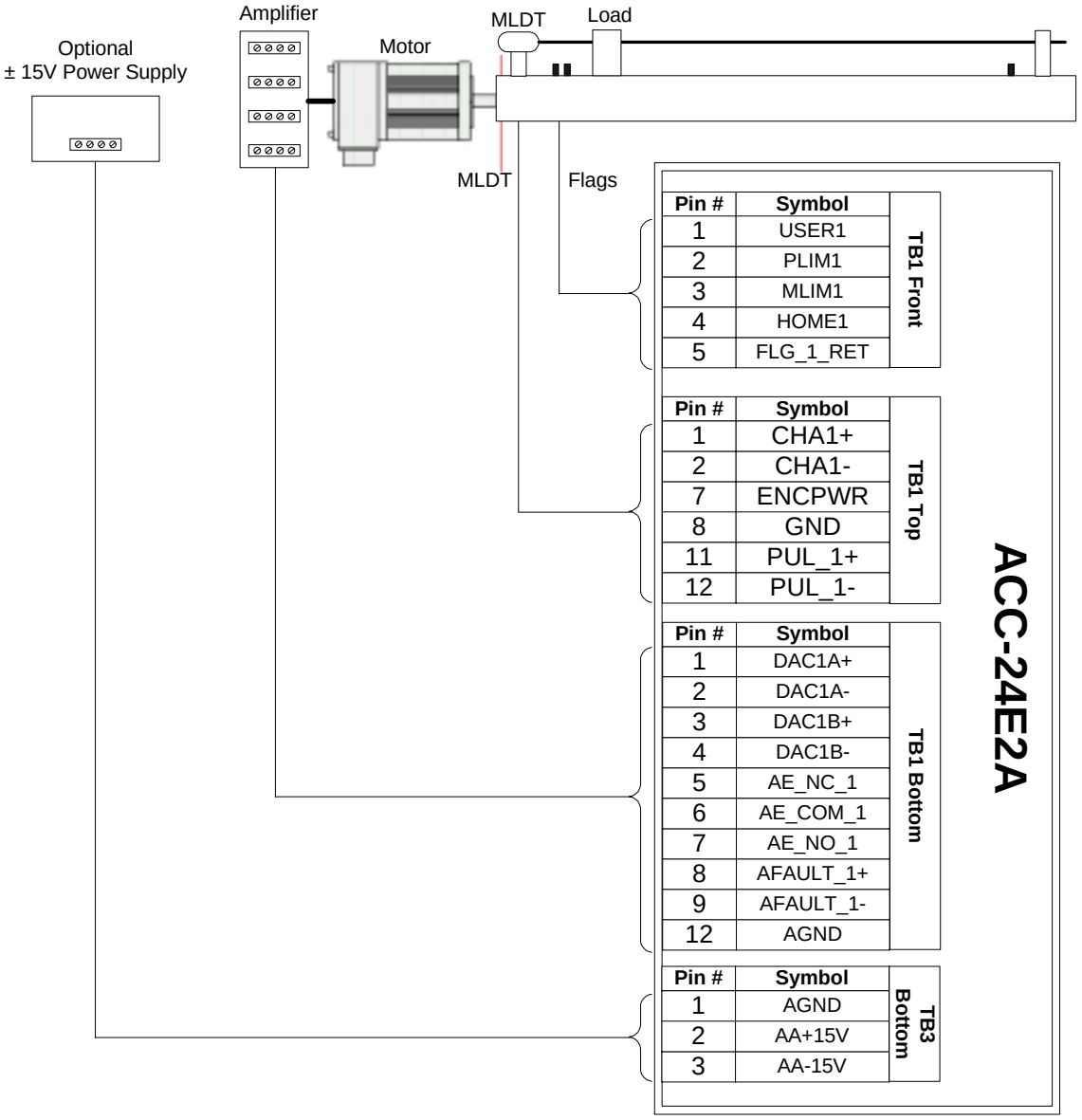


Analog Amplifier with Incremental Encoder



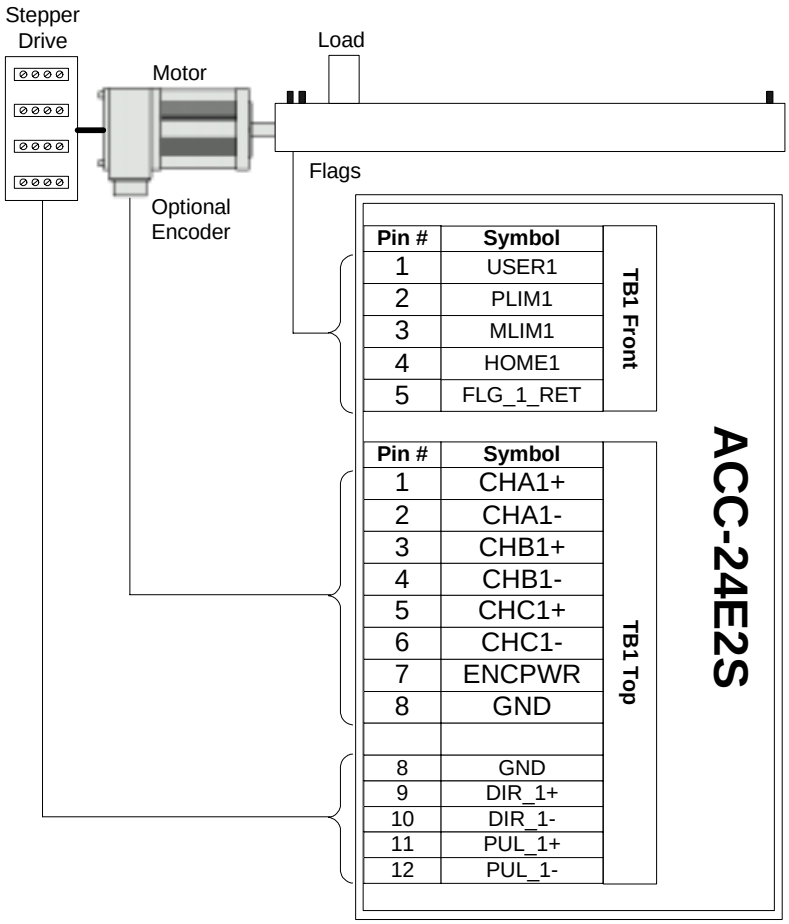
If the optional power supply is connected to the TB3 Bottom connector, jumpers E85, E87, and E88 in the Acc-24E2A axes accessory board must be removed.

Analog Amplifier with MLDT Feedback



If the optional power supply is connected to the TB3 Bottom connector, jumpers E85, E87, and E88 in the Acc-24E2A axes accessory board must be removed.

Stepper Driver with Incremental Encoder



Jumpers in the Acc-24E2S board must be configured properly to output the pulse-and-direction signals and to select between encoder C channel inputs or driver enable outputs.

SOFTWARE SETUP

The Turbo PMAC2 3U, or PMAC for short, is the CPU of the UMAC System. PMAC has a large set of Initialization parameters (I-Variables) that determine the personality of the card for a specific application. Many of these are used to configure a motor properly. The Pewin32 Pro Suite provides a set of tools for setting up the motors and programming the UMAC System.

Resetting UMAC

Perform a complete reset routine before configuring the software of a UMAC System. This will assure a clean memory configuration before starting:

```
$$$***                          ; Global Reset
P0..8191 = 0                    ; Reset P-Variables values
Q0..8191 = 0                    ; Reset Q-Variables values
M0..8191 -> *                   ; Reset M-Variables definitions
M0..8191 = 0                    ; Reset M-Variables values
UNDEFINE ALL                    ; Undefine Coordinate Systems
SAVE                            ; Save this initial clean configuration
```

Motors Setup

Each motor must be configured for the kind of output signals used (digital, analog or stepper), the feedback device used and the use of safety flags. The Turbo Setup Program, part of the Pewin32 Pro Suite, provides a step-by-step procedure for setting up the motors in a UMAC System. This is important particularly when using digital amplifiers since extra setup steps are necessary for the configuration of the current loop feedback.

Servo Loop Setup

Before the motors can be controlled in close loop, the PID gains parameters must be configured properly. The Tuning Pro software, part of the Pewin32 Pro Suite, provides a series of tools for tuning each motor of a UMAC System with minor effort. See the Pewin32 Pro section for details.

Programming PMAC

Fundamentally, PMAC is a command-driven device. PMAC performs tasks when issued ASCII command text strings, and generally, PMAC provides information to the host in ASCII text strings. When PMAC receives an alphanumeric text character over one of its ports, it does nothing but place the character in its command queue. It requires a control character (ASCII value 1 to 31) to cause it to take some actual action. The most common control character used is the carriage return (**<CR>**; ASCII value 13), which tells PMAC to interpret the preceding set of alphanumeric characters as a command and to take the appropriate action.

Once the motion parameters and programs have been set, the system can be operated as a stand-alone controller or commanded via a host computer. The **SAVE** command issued from the terminal window will store all the defined programs and parameters in flash memory for later use.

Online Commands

Many of the commands given to PMAC are on-line commands; that is, they are executed immediately by PMAC to cause some action, change some variable, or report some information back to the host. Some commands, such as **P1=1**, are executed immediately if there is no open program buffer, but are stored in the buffer if one is open. Other commands, such as **X1000 Y1000**, cannot be on-line commands; there must be an open buffer – even if it is a special rotary buffer for immediate execution. These commands will be rejected by PMAC (reporting an ERR005 if I6 is set to 1 or 3) if there is no buffer open. Still other commands, such as **J+**, are on-line commands only and cannot be entered into a program buffer (unless in the form of **CMD"J+"**, for instance).

There are three basic classes of on-line commands: motor-specific commands, which affect only the motor that is currently addressed by the host; coordinate-system-specific commands, which affect only the coordinate system that is currently addressed by the host; and global commands, which affect the card regardless of any addressing modes.

A motor is addressed by a **#n** command, where **n** is the number of the motor, with a range of 1 to 32, inclusive. This motor is the one addressed until the card receives another **#n**. For instance, the command line **#1J+#2J-** tells Motor 1 to jog in the positive direction, and Motor 2 to jog in the negative direction. There are only a few types of motor-specific commands. These include the jogging commands, a homing command, an open loop command, and requests for motor position, velocity, following error, and status.

A coordinate system is addressed by an **&n** command, where **n** is the number of the coordinate system, with a range of 1 to 16, inclusive. This coordinate system is the one addressed until the card receives another **&n** command. For instance, the command line **&1B6R&2B8R** tells Coordinate System 1 to run Motion Program 6 and Coordinate System 2 to run Motion Program 8. There is a variety of coordinate-system-specific commands. Axis definition statements act on the addressed coordinate system because motors are matched to an axis in a particular coordinate system. Since it is a coordinate system that runs a motion control program, all program control commands act on the addressed coordinate system. Q-Variable assignment and query commands are coordinate system commands also because the Q-Variables themselves belong to a coordinate system.

Some on-line commands do not depend on which motor or coordinate system is addressed. For instance, the command **P1=1** sets the value of P1 to 1 regardless of what is addressed. Among these global on-line commands are the buffer management commands. PMAC has multiple buffers and only one can be open at a time. When a buffer is open, commands can be entered into the buffer for later execution.

Control character commands (those with ASCII values 0 - 31D) are always global commands. Those that do not require a data response include carriage return **<CR>**, backspace **<BS>**, and several special-purpose characters. This allows, for instance, commands to be given to several locations on the card in a single line, and have them take effect simultaneously at the **<CR>** at the end of the line (**&1R&2R<CR>** causes both Coordinate Systems 1 and 2 to run).

Buffered (Program) Commands

As their name implies, buffered commands are not acted on immediately, but held for later execution. PMAC has many program buffers – 224 regular motion program buffers, 16 rotary motion program buffers (one for each coordinate system), 32 non-compiled PLC program buffers and 32 compiled PLC program buffers. Before commands can be entered into a buffer, that buffer must be opened (e.g. **OPEN PROG 3**, **OPEN PLC 7**). Each program command is added onto the end of the list of commands in the open buffer. To replace the existing buffer, use the **CLEAR** command immediately after opening to erase the existing contents before entering the new ones. When finished entering the program statements, use the **CLOSE** command to close the opened buffer.

Computational Features

I-Variables

I-Variables (initialization or setup variables) determines the personality of the card for a given application. They are at fixed locations in memory and have pre-defined meanings. Most are integer values, and their range varies depending on the particular variable. There are 8192 I-Variables, from I0 to I8191, and they are organized as follows:

```
I0 – I99          Global card setup
I100 – I199 Motor 1 setup
I200 – I299 Motor 2 setup
…
I3200 – I3299     Motor 32 setup
I3300 – I4799     Supplemental Motor setup
I4900 – I4999     Configuration status
I5000 – I5099     Data gathering/ADC demux setup
I5100 – I5199     Coordinate System 1 setup
I5200 – I5299     Coordinate System 2 setup
…
I6600 – I6699     Coordinate System 16 setup
I6800 – I6999     MACRO IC setup
I7000 – I7999     Servo IC setup
I8000 – I8191     Encoder conversion table setup
```

When I-Variables are described in the documentation, the following nomenclature have been used:

`xx:` This stands for motor number, and it can take values from 1 to 32.

`mn:` The m stands for servo IC number. In a UMAC System, this can take a value from 2 to 9 depending on the address given to the corresponding axes card. The n stands for the channel part of the servo IC chip. Each servo IC has four hardware channels, so n has a range from 1 to 4.

`m:` The m stands for servo IC number. In a UMAC System, this can take a value from 2 to 9, depending on the address given to the corresponding axes card.

`sx:` This represents the coordinate system number plus 50. For example, variables that refer to coordinate system 1 will be addressed by variables I5100 to I5199.

Values assigned to an I-Variable may be either a constant or an expression. The commands to do this are on-line (immediate) if no buffer is open when sent, or buffered program commands, if a buffer is open.

Examples:

```
I120 = 45
I120 = (I120+P25*3)
```

For I-Variables with limited range, an attempt to assign an out-of-range value does not cause an error. The value is rolled over automatically to within the range by modulo arithmetic (truncation). For example, I3 has a range of 0 to 3 (four possible values). Actually, the command **`I3=5`** would assign a value of 5 modulo 4 = 1 to the variable.

On the UMAC System, all of the I-Variable values must be stored in the flash memory with the **SAVE** command. After a new value is given to any I-Variable, the **SAVE** command must be issued in order for this value to survive a power-down or reset.

Default values for all I-Variables are contained in the manufacturer-supplied firmware. They can be used individually with the **`I{constant}=*`** command, or in a range with the **`I{constant}..{constant}=*`** command. Upon board re-initialization by the **`$$$***`** command or by a reset with jumper E3 of the PMAC CPU in the non-default setting, all default settings are copied from the firmware into active memory. The last saved values are not lost; they are just not used.

P-Variables

P-Variables are general-purpose user variables. They are 48-bit floating-point variables at fixed locations in Turbo PMAC's memory, but with no pre-defined use. There are 8192 P-Variables, from P0 to P8191. A given P-Variable means the same thing from any context within the card; all coordinate systems have access to all P-Variables (in contrast to Q-Variables, which are coupled to a given coordinate system below). This allows for useful information passing between different coordinate systems. P-Variables can be used in programs for any purpose desired: positions, distances, velocities, times, modes, angles, intermediate calculations, etc.

P-Variables can be located either in the main memory or in the supplemental battery-backed parameter memory (if Option 16 is ordered). If I46 is set to 0 (default) or 2, the P-Variables are located in the main memory, which has fast access (1 wait state) but whose values are not retained without a **SAVE** command copying the values to flash memory. On power-up/reset, the last saved values are copied from flash memory into the active variable registers in RAM. If I46 is set to 1 or 3, the P-variables are located in the Option 16 battery-backed RAM, which has slow access (nine wait states) but whose values are retained automatically by the battery when power is removed.

Generally, Turbo PMAC firmware has no automatic use of P-Variables. However, it can make special use of variables P0 – P32 and P101 – P132. If a command consisting simply of a constant value is sent to Turbo PMAC, that value is assigned to variable P0 (unless a special table buffer such as a compensation table or stimulus table has been defined but not yet filled – in that case the constant value will be entered into the table). For example, if the command **342<CR>** is sent to Turbo PMAC, it will interpret it as **P0=342<CR>**. This capability is intended to facilitate simple operator terminal interfaces. It is not a good idea to use P0 for other purposes, because it is easy to change this variable's value accidentally. If the application uses kinematic subroutines to convert between tool-tip (axis) positions and joint (motor) positions, variables P1 – P32 and P101 – P132 are used for the motor positions in these subroutines (Pn is Motor n position; if PVT moves are converted, P10n is Motor n velocity). If using the kinematic subroutines, make sure not to use the P-Variables employed in the subroutines for any other purpose.

Q-Variables

Q-Variables, like P-Variables, are general-purpose user variables: 48-bit floating-point variables at fixed locations in memory, with no pre-defined use. However, the meaning of a given Q-Variable (and hence the value contained in it) is dependent on which coordinate system is utilizing it. This allows several coordinate systems to use the same program (for instance, containing the line X(Q1+25) Y(Q2), but to do have different values in their own Q-Variables (which in this case, means different destination points).

Several Q-Variables have special uses. The **ATAN2** (two-argument arctangent) function uses Q0 as its second argument (the cosine argument) automatically. The **READ** command places the values it reads following letters A through Z in Q101 to Q126, respectively, and a mask word denoting which variables have been read in Q100. The **S** (spindle) statement in a motion program places the value following it into Q127. If the application uses kinematic subroutines to convert between tool-tip (axis) positions and joint (motor) positions, variables Q1 – Q10, and possibly Q11 – Q19 for the coordinate system are used for the axis data in these subroutines. (Q1 – Q9 are for axis positions; Q10 tells whether PVT moves are being converted; if PVT moves are converted, Q11 – Q19 are for axis velocities.) Therefore, since 8192 Q-Variables are shared between potentially 16 Coordinate Systems (512 variables each), the practical ranges of the Q-Variables to be used safely in motion programs are Q20 - Q99 and Q128 - Q511.

The set of Q-Variables working within a command depends on the type of command. When accessing a Q-Variable from an on-line (immediate) command from the host, the Q-Variable for the currently host-addressed coordinate system is used (with the **&n** command). When accessing a Q-Variable from a motion program statement, the Q-variable belonging to the coordinate system running the program is used. If a different coordinate system runs the same motion program, it will use different Q-Variables.

When accessing a Q-Variable from a PLC program statement, the Q-Variable for the coordinate system that has been addressed by that PLC program with the **ADDRESS** command is used. Each PLC program can address a particular coordinate system independent of other PLC programs and independent of the host addressing. If no **ADDRESS** command is used in the PLC program, the program uses the Q-Variables for C.S. 1.

M-Variables

To permit easy user access to Turbo PMAC's memory and I/O space, M-variables are provided. Typically, M-Variables are used to access general-purpose IO points, read motor registers and monitor status bits. There are 8192 M-Variables (M0 to M8191), and as with other variable types, the number of the M-Variable may be specified with either a constant or an expression: M576 or M(P1+20). The definition of an M-Variable is set using the defines-arrow (**->**) composed of the minus sign and greater than symbols. Generally, a definition must be set only once with an on-line command. The **SAVE** command must be used to retain the definition through a power-down or reset. An M-Variable is defined by assigning it to a location and defining the size and format of the value in this location. An M-Variable can be a bit, a nibble (4 bits), a byte (8 bits), 1-1/2 bytes (12 bits), a double-byte (16 bits), 2-1/2 bytes (20 bits), a 24-bit word, a 48-bit fixed-point double word, a 48-bit floating-point double word, or special formats for dual-ported RAM. The following types are the most commonly used as specified by the address prefix in the definition:

```
X:    1 to 24 bits fixed-point in X-memory
Y:    1 to 24 bits fixed-point in Y-memory
D:    48 bits fixed-point across both X- and Y-memory
DP:   32 bits fixed-point (low 16 bits of X and Y) (for use in dual-ported
      RAM)
F:    32 bits floating-point (low 16 bits of X and Y) (for use in dual-ported
      RAM)
*:    No address definition; uses part of the definition word as general-
      purpose variable
```

If an X or Y type of M-Variable is defined, also define the starting bit to use, the number of bits, and the format (decoding method). Typical M-Variable definition statements are:

```
M1->Y:$078C02,8,1      ; Unsigned one-bit wide starting at bit 8 on the Y-register
M102->Y:$78003,8,16,S  ; Signed 16-bits wide starting at bit 8 on the Y-register
M103->X:$078003,0,24,S ; Signed 24-bits wide starting at bit 0 on the X-register
M161->D:$8B            ; 48-bit fixed-point double word
M50->DP:$060401        ; Dual-Ported RAM 48-bit fixed-point double word
M51->F:$0607FF         ; Dual-Ported RAM 48-bit floating-point double word
```

There is a set of suggested M-Variables definitions that allow accessing the most commonly used registers in a UMAC System. The definitions are made to access motor position registers, status bits and general-purpose IO points. Downloading this set of M-Variables simplifies the definition process. See the Turbo PMAC Software Reference for details.

Prepare a single file with all of the M-Variable definitions and put the **M0..8191->*** command at the top of this file. This will remove all existing definitions and help to prevent mysterious problems caused by stray M-Variable definitions. The M-Variable definitions are stored as 48-bit codes at Turbo PMAC memory addresses $004000 (for M0) to $005FFF (for M8191). The Y-register contains the address of the register pointed to by the definition; the X-register contains a code that determines what part of the register is used and how it is interpreted. If another M-Variable points to the Y-register, it can be used to change the subject register. The main use of this technique is to create arrays of registers which can be used to walk through tables in memory.

Once defined, an M-Variable may be used in programs just as any other variable – through expressions. When the expression is evaluated, Turbo PMAC reads the defined memory location, calculates a value based on the defined size and format, and utilizes it in the expression. Many M-Variables have a more limited range than Turbo PMAC's full computational range. If a value outside of the range of an M-Variable is placed to that M-Variable, Turbo PMAC rolls over the value to within that range automatically and does not report any errors. For example, with a single bit M-Variable, any odd number written to the variable ends up as 1, any even number ends up as 0. If a non-integer value is placed in an integer M-Variable, Turbo PMAC rounds to the nearest integer automatically.

When using the M-Variables in a motion program, especially when used to control digital general-purpose outputs, it is important to use double-equal assignments. M1==1, for example, will indicate to PMAC that the assignment must take place at the blending point between the previous move encountered before the assignment and the next. In Linear and Circle mode moves, the blending occurs V*TA/2 distance ahead of the specified intermediate point, where V is the commanded velocity of the axis, and TA is the acceleration (blending) time. This feature is only available for M-Variables.

Arrays

It is possible to use a set of P or Q-Variables as an array. To read values from the array or assign values to it, replace the constant specifying the variable number with an expression in parentheses.

Example:

```
P1 = 10                    ; P1 is the array index variable in this case
P3 = P(P1)                 ; Same as P3 = P10
P1 = 15                    ; P1 is the array index variable in this case
P(P1) = 5                  ; Same as P15 = 5
```

Another method to use to get array capabilities is indirect M-Variables addressing.

Example: Values 31 to 40 will be assigned to variables P1 through P10

```
M34->L:$6001               ; Standard location for P1 (when I46 = 0 or 2)
M35->Y:$4022,0,24          ; Definition word of M34
OPEN PLC 15 CLEAR
P100=31
WHILE (P100!>40)           ; From 31 to 40
    M34=P100               ; Value is written to the array
    P100=P100+1            ; Next value
    M35=M35+1              ; Next Array position (next P-variable)
ENDWHILE
DISABLEPLC15               ; This PLC runs only once
CLOSE
ena PLC15                  ; Enable the PLC (I5 must be 2 or 3)
P1..10                     ; List the values of P1 to P10
```

The same concept applies for Q-Variables and M-Variables arrays when using the appropriate address locations.

Operators

PMAC operators work like those in any computer language: they combine values to produce new values. PMAC uses the four standard arithmetic operators: +, -, *, and /. The standard algebraic precedence rules are used: multiply and divide are executed before add and subtract, operations of equal precedence are executed left to right, and operations inside parentheses are executed first.

PMAC also has the % modulo operator, which produces the resulting remainder when the value in front of the operator is divided by the value after the operator. Values may be integer or floating point. This operator is useful particularly for dealing with counters and timers that roll over.

When the modulo operation is completed using a positive value X, the results can range from 0 to X (not including X itself). When the modulo operation is completed using a negative value -X, the results can range from -X to X (not including X itself). The negative modulo operation is useful when a register can roll over in either direction.

PMAC has three logical operators that do bit-by-bit operations: **&** (bit-by-bit AND), **|** (bit-by-bit OR), and **^** (bit-by- bit EXCLUSIVE OR). If floating-point numbers are used, the operation works on the fractional as well as the integer bits. **&** has the same precedence as ***** and **/**; **|** and **^** have the same precedence as **+** and **-**. The use of parentheses can override the default precedence.

Note:

These bit-by-bit logical operators are different from the simple Boolean operators AND and OR used in compound conditions.

Functions

The available functions are **SIN**, **COS**, **TAN**, **ASIN**, **ACOS**, **ATAN**, **ATAN2**, **SQRT**, **LN**, **EXP**, **ABS**, and **INT**. These functions perform mathematical operations on constants or expressions to yield new values. Whether the units for the trigonometric functions are degrees or radians is controlled by the global I-Variable I15.

SIN	This is the standard trigonometric sine function.
COS	This is the standard trigonometric cosine function.
TAN	This is the standard trigonometric tangent function.
ASIN	This is the inverse sine (arc-sine) function with its range reduced to +/-90 degrees.
ACOS	This is the inverse cosine (arc-cosine) function with its range reduced to 0 -- 180 degrees.
ATAN	This is the standard inverse tangent (arc-tangent) function.
ATAN2	This expanded arctangent function returns the angle whose sine is the expression in parentheses and whose cosine is the value of Q0 for that coordinate system. If calculating in a PLC program, make sure that the proper coordinate system has been addressed in that PLC program. It is only the ratio of the two values' magnitudes and their signs that matter in this function. It is distinguished from the standard ATAN function by the use of two arguments. The advantage of this function is that it has a full 360-degree range, rather than the 180-degree range of the single-argument ATAN function.
LN	This is the natural logarithm function (log base e).
EXP	This is the exponentiation function (e^x). Note: To implement the y^x function, use $e^{x \ln(y)}$ instead. For example, use this expression to implement the function $P1^{P2}$: `EXP(P2*LN(P1))`.
SQRT	This is the square root function.
ABS	This is the absolute value function.
INT	This is a truncation function, which returns the greatest integer less than or equal to the argument (`INT(2.5)=2, INT(-2.5)=-3`).

Functions and operators could be used either in motion programs, PLCs, or as online commands. For example, the following commands can be typed in a terminal window:

```
P1=SIN(45) P1      ; Reports the sine value of a 45° angle
I130=I130/2        ; Lower the proportional gain of Motor #1 by half
```

Comparators

A comparator evaluates the relationship between two values (constants or expressions). It is used to determine the truth of a condition in a motion or PLC program. The valid comparators for PMAC are:

```
=     (equal to)
!=    (not equal to)
>     (greater than)
!>    (not greater than; less than or equal to)
<     (less than)
!<    (not less than; greater than or equal to)
~     (approximately equal to -- within one)
!~    (not approximately equal to -- at least one apart)
```

Note:

<= and >= are not valid PMAC comparators. The comparators `!>` and `!<`, respectively, should be used instead.

Encoder Conversion Table

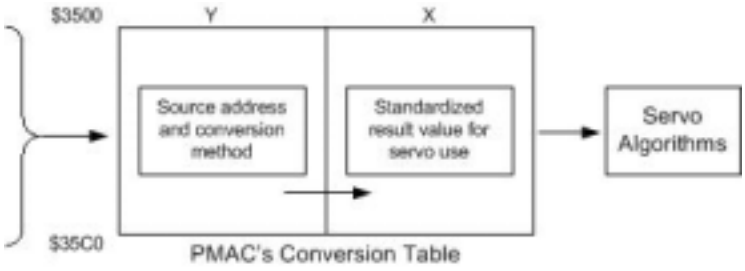
Turbo PMAC uses a two-step process to work with its feedback and master position information for the servo algorithm to provide maximum power and flexibility. For most Turbo PMAC applications with quadrature encoder feedback, this process can be virtually transparent, with no need to worry about the details. This is because the default conversion table is set to convert all the incremental quadrature channels found in the UMAC System. However, some will need to understand this conversion process in some detail to make the changes necessary to use other types of feedback, to optimize the system, or to perform special functions.

The first stage in the position processing uses the hardware registers such as encoder counters with associated timers, A/D registers, or accessory cards for parallel input. These work continually without direct software intervention with data typically latched on the servo interrupt. Beyond this point, the process is software-controlled. Turbo PMAC has an intermediate step using a software structure called the Encoder Conversion Table to pre-process the information in the latched registers. This table tells PMAC what registers to process and how to process them; it also holds the intermediate processed data. PEWIN32-Pro has a special editing screen for viewing and changing the encoder conversion table.

Conversion Table Structure

The Encoder Conversion Table has two columns, one in the X memory space of the processor, and one in the Y memory space. The X-column holds the converted data, while the Y-column holds the addresses of the source registers and the conversion methods used on the data in each of those source registers. Set up the table by writing to the Y-column and PMAC uses the Y-column data to fill up the X-column each servo cycle.

- Quadrature, incremental, encoder
- Parallel binary feedback
- Laser interferometer feedback
- Analog feedback
- 4096 sinusoidal interpolator feedback
- SSI encoder inputs
- Yaskawa or Mitsubishi absolute encoders
- MLDTs feedback inputs



Each line is setup through an I-Variable and these are numbered from I8000 to I8191. Depending on the conversion method, each entry can be one, two or three lines long using one, two or three setup I-Variables, consecutively numbered. The result of the conversion will be located at the address of the last used line. For example, if three lines are used starting with I-Variable I8000, the result will be located at X:$3502.

Further Position Processing

Once the position feedback signals have been processed by the Encoder Conversion Table (which happens at the beginning of each servo cycle), the data is ready for use by the servo loop. For each activated motor, PMAC takes the position information in the 24-bit register pointed to by Ixx03 and extends it in software to a 48-bit register that holds the actual motor position. Several other features are available for conditioning the feedback signal, if necessary.

- **Axis Position Scaling:** for running motion programs, motors are mapped, either individually or in groups, to a coordinate system with axis letters like X, Y and Z. For each individual motor, a scale factor determines the relationship between encoder counts and user units to be used in motion programs.
- **Leadscrew Compensation:** for each individual motor, it is possible to create a table to compensate for potential leadscrew imperfections. This provides added positioning accuracy, especially when moving large distances.
- **Backlash Compensation:** On reversal of the direction of the commanded velocity, a pre-programmed backlash distance is added to or subtracted from the commanded position, thus compensating for a potential backlash.

PMAC Position Registers

The PMAC Executive position window or the online **P** command reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bits 1 and 0 of Ixx06 are 1 (handwheel offset mode), minus the master position register:

```
M175->X:$00B0,4,1       ; Bit 0 of I106
M176->X:$00B0,5,1       ; Bit 1 of I106
M162->D:$008B           ; #1 Actual position (1/[Ixx08*32] cts)
M164->D:$00CC           ; #1 Position bias (1/[Ixx08*32] cts)
M167->D:$008D           ; #1 Present master ((handwheel) pos (1/[Ixx07*32] cts
                        ; of master or (1/[Ixx08*32] cts of slaved motor)
M169->D:$0090           ; #1 Compensation correction
```

$$P100 = \frac{(M162 + M164 + M169 - M175 * M176 * M167)}{I108 * 32}$$

P100 will report the same value as the online **P** command or the position window in the PMAC Executive program.

The addresses given are for Motor #1. For the registers of another motor x, add (x-1)*$80 to the appropriate motor #1 address).

```
M161->D:$0088           ; #1 Commanded position (1/[Ixx08*32] cts)
```

The motor commanded position registers contain the value in counts where the motor is commanded to move. It is set through **JOG** online commands or axis move commands (**X10**) inside motion programs.

To read this register in counts: `P161 = M161 / (I108*32)`

```
M162->D:$008B           ; #1 Actual position (1/[Ixx08*32] cts)
```

The actual position register contains the information read from the feedback sensor after it has been properly converted through the encoder conversion table and extended from a 24-bits register to a 48-bits register.

To read this register in counts: `P162 = M162 / (I108*32)`

```
M163->D:$00C7           ; #1 Target (end) position (1/[Ixx08*32] cts)
```

This register contains the most recently programmed position and it is called the target position register, if Isx13>0, PMAC is in segmentation mode and the value of M163 corresponds to the last interpolated point calculated.

To read this register in counts: `P163 = M163 / (I108*32)`

```
M164->D:$00CC                    ; #1 Position bias (1/[Ixx08*32] cts)
```

This register contains the offset specified in the axis definition command **#1->X + <offset>**

The online command **{axis}={constant}**or the motion program command **PSET** adds the specified offset to the existing M164 offset: **M164 = M164 + <new_offset>**.

To read this register in counts: `P164 = M164 / (I108*32)`

```
M165->L:$2047                    ; &1 X-axis target position (engineering units)
```

M165 contains the programmed axis position through a motion program, **X10** for example, in engineering units. It is updated also by the online command **{axis}={constant}** or the motion program command **PSET.**

```
M166->X:$009D,0,24,S             ; #1 Actual velocity (1/[Ixx09*32] cts/cyc)
```

M166 is the actual velocity register. For display purposes, use the motor-filtered actual velocity, M174

To read this register in cts/msec: `P166 = M166 * 8388608 / (I109 * 32 * I10 * (I160+1))`

```
M167->D:$008D          ; #1 Present master ((handwheel) pos (1/[Ixx07*32] cts
                       ; of master or (1/[Ixx08*32] cts of slaved motor)
```

M167 is related to the master/slave relationship set through Ixx05 and Ixx06. It contains the present number of counts from the master.

To read this register in counts: `P167 = M167 / (I108*32)`

or

`P167 = M167 / (I107*32)`

```
M169->D:$0090                    ; #1 Compensation correction
```

This contains the calculated leadscrew compensation correction according to actual position (M162) and the leadscrew compensation table set through the **DEFINE COMP** command.

To read this register in counts: `P169 = M169 / (I108*32)`

```
M172->L:$00D7                    ; #1 Variable jog position/distance (counts)
```

Contains the distance for the **J=*** command.

Example:

```
M172=2000   J=*   ;Jog to position 2000 encoder counts
```

```
M173->Y:$00CE,0,24,S             ; #1 Encoder home capture offset (counts)
```

Contains the home offset from the reset/power-on position; Important for the capture/compare features.

Example:

```
If (M117=1)
  P103=M103-M173                 ; Captured position minus offset
endif
```

```
M174->D:$00EF                    ; #1 filtered actual velocity (1/[Ixx09*32]
                                 ; cts/servo cycle)
```

This register contains the actual velocity averaged over the previous 80 real-time interrupt periods (80*[I8+1] servo cycles); good for display purposes.

To read this register in cts/msec: `P174 = M174 * 8388608 / (I109 * 32 * I10 * (I160+1))`

```
M180->D:$0091            ; #1 following error (1/[Ixx08*32] cts)
```

Following error is the difference between motor desired and measured position at any instant. When the motor is open loop (killed or enabled), following error does not exist and PMAC reports a value of 0.

$$P176 = \frac{M161 - M162 + M164 + M169 - M175 * M176 * M167}{I108 * 32}$$

To read this register in counts: `P176 = M175 / (I108*32)`

Summary of Selected I-Variables

Motor Definition I-Variables

Ixx00 – Motor xx Activate: For controlling an actual physical motor, this PMAC motor I-Variable should be set to one. If there is no physical motor associated with this PMAC motor xx, then this variable should be set to zero especially when using the encoder input or output command (DAC or stepper) for any general purpose.

Ixx02 – Motor xx Command Output Address: This variable determines which hardware channel will be used to output the command signals to the amplifier. It must be changed from the default value when using stepper type drivers. Variable Ixx96 further configures the command outputs for motor xx.

Ixx03 – Motor xx Position Loop Feedback Address: This variable determines which hardware channel will be used to input the feedback information for closing the position loop.

Ixx04 – Motor xx Velocity Loop Feedback Address: This variable determines which hardware channel will be used to input the feedback information for closing the velocity loop. It differs from variable Ixx03 when two encoders, one on the load and one in the motor, are used in double-feedback applications.

Motor Safety I-Variables

Warning:

Setting Ixx11 to zero (disabled) could lead to a dangerous motor runaway condition.

Ixx11 – Motor xx Fatal Following Error Limit: This variable sets the maximum number of 1/16 counts of allowed following error before the motor is shutdown.

Ixx13 – Motor xx + Software Position Limit: This variable determines the maximum allowed range of motion in the positive direction. Enabling this function is useful when no actual end-of-travel limit switches are used.

Ixx14 – Motor xx - Software Position Limit: This variable determines the maximum allowed range of motion in the negative direction. Enabling this function is useful when no actual end-of-travel limit switches are used.

Ixx15 – Motor xx Abort/Lim Deceleration Rate: This parameter sets the deceleration rate used when a programmed motion is aborted; either by the **A** abort command or when a maximum position limit is reached.

Ixx16 – Motor xx Maximum Velocity: This parameter sets the maximum allowed velocity for a motor performing a linear move issued from a motion program. This is observed only when variable Isx13 is zero or a special lookahead buffer has been defined with Isx20 > 0.

Ixx17 – Motor xx Maximum Acceleration: This parameter sets the maximum allowed acceleration for a motor performing a linear move issued from a motion program. This is observed only when variable Isx13 is zero or a special lookahead buffer has been defined with Isx20 > 0.

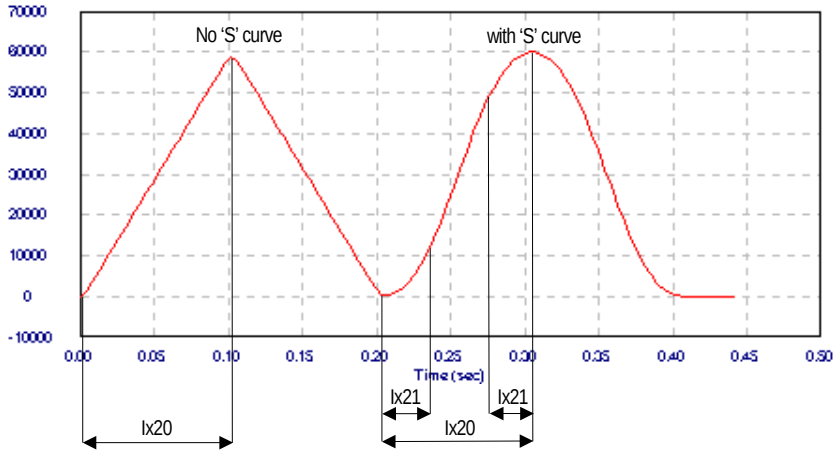
Ixx19 – Motor xx Maximum Jog/Home Acceleration: This parameter sets the maximum allowed acceleration rate for a motor performing a jog or homing move.

S Curve and Linear Acceleration Variables

The acceleration portion of a programmed move, either programmed by a jog or a motion program command, is controlled by two time parameters in units of millisecond. In the case of jog or homing commands, these two parameters are I-variables Ixx20 and Ixx21. Ixx20 determines the overall acceleration time, which is the total time required for any change in velocity. Ixx21 determines the portion of the overall acceleration ramp that is performed in S curve mode.

In every case, if two times the S curve acceleration parameter is greater than the linear acceleration parameter, then the overall acceleration time will be two times the S curve acceleration time:

```
If (2 x Ixx21)  > Ixx20 then Ixx20 = (2 x Ixx21)
```



The acceleration of either linear or circular interpolated moves programmed from a motion program is determined by a set of different parameters. However, these parameters have the same meaning as those described above:

Move Type	S Curve Acceleration Parameter	Linear Acceleration Parameter
Jog or Home commands	`Ixx21`	`Ixx20`
Linear or circular interpolation	`TS or Isx88`	`TA or Ixx87`

Rate vs. Time: Programming the Maximum Acceleration Parameters

The safety I-Variable Ixx17 determines the maximum allowed acceleration for the motor xx when no special lookahead buffer is used in Linear mode moves. (Ixx19 is used for jog or home commands.) These variables are programmed in units of encoder counts per millisecond square. However, the acceleration of a programmed move, from either jog commands or a motion program, is set in milliseconds as described above. The following relationship holds for the conversion between those different units:

$$Acceleration\ Rate = \frac{Velocity}{Linear\ Acceleration\ Time - S\ Curve\ Acceleration\ Time}$$

Examples:

Jog Commands

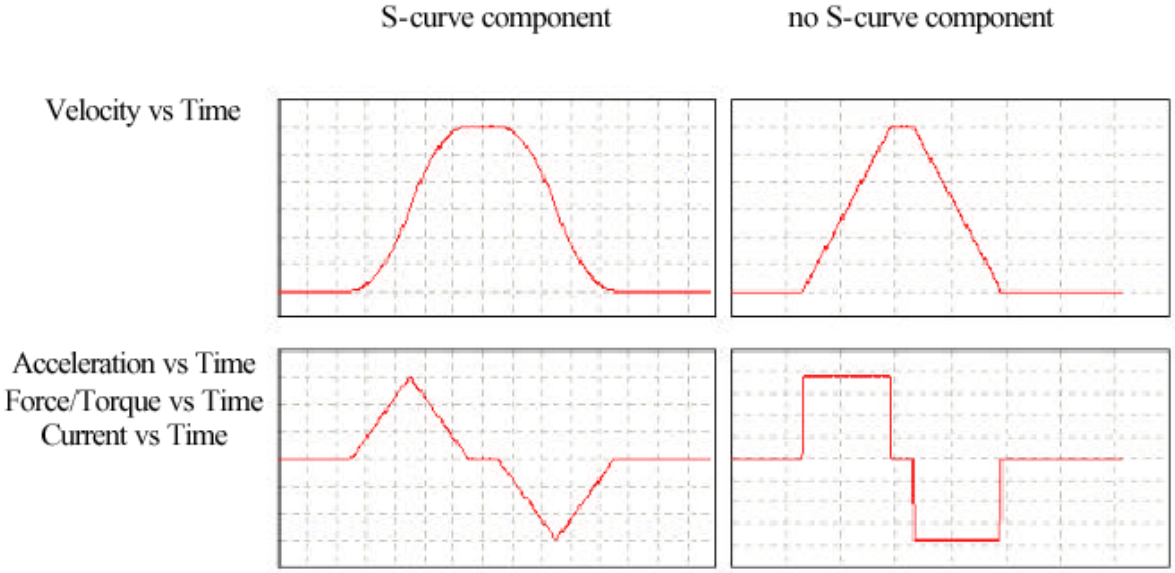
$$Ixx19 = \frac{Ixx22}{Ixx20 - Ixx21}$$

Linear Interpolated Moves

$$Ixx17 = \frac{Ixx16}{Isx87 - Isx88}$$

Benefits of Using S-Curve Acceleration Profiles

In an electric motor, the acceleration directly translates into torque and electrical current. When no S curve component is programmed, the acceleration rate, torque and current are applied immediately to the motor after it starts moving. With a programmed S curve profile, on the other hand, the acceleration is applied linearly from zero to the programmed value resulting in a smoother transition in acceleration torque and current. However, the acceleration rate in a pure S curve acceleration profile is two times that which is necessary for a pure linear acceleration profile. (See equation above.) Then, the use of S curve acceleration requires a longer overall acceleration time than when using straight linear acceleration.



Motor Movement I-Variables

Ixx20 – Motor xx Jog/Home Acceleration Time: This variable determines how long the acceleration portion of the jog moves will take, regardless if a S curve component is also programmed or not. (See diagram above.)

Ixx21 – Motor xx Jog/Home S-Curve Time: This variable determines the portion of the acceleration ramp that will be performed in S curve mode. If Ixx20 is set to zero, then the acceleration ramp will take 2*Ixx21 and will be executed in pure S curve mode.

Ixx22 – Motor xx Jog Speed: This variable sets the jog velocity. If the motor xx is moving already, a new jog command must be issued for the Ixx22 parameter to have effect.

Ixx23 – Motor xx Homing Speed and Direction: This variable is often set to the same value as Ixx22. However, what is important in this case is its sign which determines the direction the motor xx will take when searching for the home sensor.

Ixx24 – Motor xx Flag Mode Control: This variable specifies how the information in the register specified by Ixx25 is used.

Ixx25 – Motor xx Flag Address: This variable determines which set of flags motor 'xx' will use. These flags include the end-of-travel limits, the amplifier enable and fault lines and the home flag.

Note:

If a hardware flag is used for home reference and a quadrature encoder is used for feedback, they must both belong to the same hardware channel in the axes board.

Ixx26 – Motor xx Home Offset: This variable determines an offset in 1/16 of a count that PMAC will move after the home procedure is completed. This is important to move the motor away from the home sensor which could be necessary for a better reliable home search routine.

Ixx96 – Motor xx Position Capture & Trigger Mode: This variable controls how Turbo PMAC writes to the command output registers specified in Ixx02.

Ixx97 – Motor xx Position Capture & Trigger Mode: This variable controls the triggering function and the position capture function for triggered moves on motor xx. These triggered moves include homing search moves, on-line jog-until-trigger moves, and motion program **`RAPID`**-mode move-until-trigger.

Servo Control I-Variables

The servo control variables are setup in the motor tuning process. Usually, this is accomplished using a software tool like the Tuning Pro, part of the Pewin32 Pro Suite Software.

Ixx30 – Motor xx Proportional Gain: This is the most important variable in the servo control loop. It determines how strong the corrections on the servo loop will be made, based on a given following error value. The rule of thumb for the setup of this variable is to increase it until the motor starts to buzz and then back down for about 20 % of its value.

Ixx31 – Motor xx Derivative Gain: This variable acts effectively as an electronic damper. The higher Ixx31 is, the heavier the damping effect is. On a typical system with a current-loop amplifier and PMAC's default servo update time, an Ixx31 value of 2000 to 3000 will provide a critically damped step response.

Ixx32 – Motor xx Velocity Feed Forward Gain: Typically, this variable is used to minimize the tracking errors when the motor is moving with a constant velocity. If the motor is driving a current-loop (torque) amplifier, usually Ixx32 will be equal to (or slightly greater than) Ixx31 to minimize tracking error.

Ixx33 – Motor xx Integral Gain: Typically, this variable is used to minimize the steady state following error when the motor is settling on the target position. The following error in this case is due to gravity and external forces.

Ixx35 – Motor xx Acceleration Feed Forward Gain: This parameter is intended to reduce tracking error due to inertial lag.

Ixx68 – Motor xx Friction Feedforward: This parameter is intended primarily to help overcome errors due to mechanical friction.

Channel Specific I-Variables

I7mn0 – Encoder Decode Control: This variable determines how an increase in the encoder feedback counter will be interpreted when translated into position, either as an increase or a decrease in the position counter. This determines the proper direction of motion. Typical values are either 3 or 7, which determine a clock-wise or counter-clockwise direction of decoding respectively.

I7mn2 – Encoder Capture Control: This variable determines the trigger condition that completes the home search command. For example, the trigger condition could be a combination of the home sensor being activated and the encoder C channel rising high.

I7mn3 – Encoder Flag Select: This variable determines which flag will be used for the home trigger condition, selected from the home flag, the end-of-travel limits, the user flag or the amplifier fault flag.

Homing Search Moves

The purpose of a homing search move is to establish an absolute position reference when an incremental position feedback sensor is used. The move until trigger construct is ideal for finding the sensor that establishes the home position and returning to this position automatically. The trigger condition for homing-search moves, as for other triggered moves, is specified by Ixx97, Ixx24, and Ixx25. Variables Ixx20, Ixx21, Ixx23 and Ixx26 specify the move parameters of home search type commands. If no trigger is found, the pre-trigger move will continue indefinitely, or until stopped by an error condition such as hitting overtravel limits.

Note:

If a hardware flag is used for home reference and a quadrature encoder is used for feedback, they must both belong to the same hardware channel in the axes board.

A homing search move can be initiated with the on-line motor-specific **HOME** command (short form **HM**, e.g., **#1HM)**. The homing search move can be commanded also from within a motion program with the **HOMEn** command, where **n** is the motor number. Note that this command specifies a motor, unlike other motion program commands that specify an axis move. Multiple homing moves can be started together by specifying a list or range of motor numbers with the command (e.g. **HOME1,3** or **HOME2..6**). Further program execution will wait for all of these motors to finish their homing moves. Separate homing commands, even on the same line (e.g. **HOME1 HOME2**) will be executed in sequence, with the first finishing before the second starts.

Jogging Moves

Indefinite Jog Commands

J+ commands an indefinite positive jog for the addressed motor. **J-** commands an indefinite negative jog; **J/** commands an end to the jog, leaving the motor in position control after the deceleration. It is possible for the **J/** command to leave the commanded position at a fractional count which can cause dithering between the adjacent integer count values. If this is a problem, the **J!** command can be used to force the commanded position to the nearest integer count value.

Jogging to a Specified Position

Jog commands to a specified position, or of a specified distance, can be given. **J=** commands a jog to the last pre-jog position. **J={constant}** commands a jog to the (unscaled) position specified in the command. **J=={constant}** commands a jog to the (unscaled) position specified in the command and makes that position the pre-jog position. **J^{constant}** commands a jog of the specified distance from the actual position at the time of the command (**J^0** can be useful to take up remaining following error). **J:{constant}** commands a jog of the specified distance from the commanded position at the time of the command.

Jog Moves Specified by a Variable

Jogging moves to a position or a distance specified by a variable are possible. Each motor has a specific register (L:$00D7 for motor 1, L:$0157 for motor 2, etc.) that holds the position or distance to move on the next variable jog command. This register contains a floating-point value scaled in encoder counts. It should be accessed with an L-format M-Variable. The **J=*** command causes PMAC to use this value as a destination position. The **J^*** command causes PMAC to use the value as a distance from the actual position at the time of the command. The **J:*** command causes PMAC to use the value as a distance from the commanded position at the time of the command.

Each time one of these commands is given, the acceleration and velocity parameters at that time control the response to the command. To change speed or acceleration parameters of an active jog move, change the appropriate parameters, then issue another jog command.

Jog-Until-Trigger

The jog-until-trigger function permits a jog move to be interrupted by a trigger and terminated by a move relative to the position at the time of the trigger. It is very similar to a homing search move, except that the motor zero position is not altered and there is a specific destination in the absence of a trigger. The jog-until-trigger function for a motor is specified by adding a **^{constant}** specifier to the end of a regular definite jog command for the motor, where **{constant}** is the distance to be traveled relative to the trigger position before stopping in encoder counts. It cannot be used with the indefinite jog commands **J+** and **J-**. To set the trigger for motor xx to occur when an obstruction such as a hard stop is encountered, set Ixx97 to 3, specifying both following-error trigger and software capture.

Example:

```
#2J:5000^-100     ; Jog 5000 counts in the positive direction in the absence
                  ; of a trigger, but if trigger is found, jog to -100 cts
                  ; from trigger position.
```

Command and Send Statements

Using the **COMMAND** or **CMD** statement, online commands can be issued from a PLC or motion program and have the same result as if they were issued from a host computer or a terminal window. Certain online commands might not be valid when issued from a running program. For example, a jog command to a motor part of a coordinate system running a motion program will be invalid. I6 should not be set to 2 in early development so that it will be known when PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hang-up from a full response queue or from disturbing the normal host communications protocol.

Messages to a host computer connected through the PMAC port x could be issued using the **SENDx** command:

SENDS transmits the message to the main serial port.
SENDP transmits the message to the PC/104 parallel bus port.
SENDR transmits the message through the DPRAM ASCII response buffer.
SENDA transmits the message to the Option 9T auxiliary serial port.

If there is no host on the port to which the message is sent, or the host is not ready to read the message, the message is left in the queue. If several messages back up in the queue, the program issuing the messages will halt execution until the messages are read. This is a common mistake when the **SEND** command is used outside of an Edge-Triggered condition in a PLC program. On the serial port, it is possible to send messages to a non-existent host by disabling the port handshaking with I1=1.

If a program, particularly a PLC program, sends messages immediately on power-up/reset, it can confuse a host-computer program (such as the PMAC Executive Program) that is trying to find PMAC by querying it and looking for a particular response.

It is possible, particularly in PLC programs, to order the sending of messages or command statements faster than the port can handle them. Usually this happens if the same **SEND** or **CMD** command is executed every scan through the PLC. For this reason, have at least one of the conditions that causes the **SEND** or **CMD** command to execute set false immediately to prevent execution of this **SEND** or **CMD** command on subsequent scans of the PLC.

Example:

```
IF (M7000=1)                  ; input is ON
      IF (P11=0)              ; input was not ON last time
            COMMAND"#1J+"     ; jog motor
            P11=1             ; set latch
      ENDIF
ELSE
      P11=0                   ; reset latch
ENDIF
```

MOTION PROGRAMS

PMAC can hold up to 256 motion programs at one time. Any coordinate system can run any of these programs at any time, even if another coordinate system is already executing the same program. Turbo PMAC can run as many motion programs simultaneously as there are coordinate systems defined on the card (up to 16). A motion program can call any other motion program as a subprogram, with or without arguments.

PMAC's motion program language is perhaps best described as a cross between a high-level computer language like BASIC or Pascal, and G-Code (RS-274) machine tool language. In fact, it can accept straight G-Code programs directly (provided it has been set up properly). It has the calculational and logical constructs of a computer language and move specification constructs similar to machine tool languages. Numerical values in the program can be specified as constants or expressions.

Motion or PLCs programs are entered in any text file to be downloaded later to PMAC. Pewin32 Pro provides a built-in text editor for this purpose. Once the code has been written, it can be downloaded to PMAC using Pewin32 Pro. In addition, any PMAC command can be issued from any terminal window communicating with PMAC. For example, online commands can jog motors, change variables, report variables values, start and stop programs, query for status information, and even write short motion and PLC programs. In fact, the downloading process is just a sequence of valid PMAC commands sent line by line from a particular text file.

How PMAC Executes a Motion Program

A PMAC program exists to pass data to the trajectory generator routines that compute the series of commanded positions for the motors every servo cycle. The motion program must be working ahead of the actual commanded move to keep the trajectory generators fed with data. PMAC processes program lines either zero, one or two moves (including **DWELL**s and **DELAY**s) ahead. Calculating one move ahead is necessary in order to be able to blend moves together. Calculating a second move ahead is necessary if proper acceleration and velocity limiting is to be done, or a three-point spline is to be calculated (**SPLINE** mode.)

Zero Moves Ahead	Two Moves Ahead	One Move Ahead
RAPID	LINEAR with Isx13=0	LINEAR with Isx13>0
HOME	SPLINE1	CIRCLE
DWELL		PVT
"B1S" (step through program 1)		
Isx92=1 (blending disabled)		

When a **RUN** command is given, and every time the actual execution of programmed moves progresses into a new move, a flag is set saying it is time to do more calculations in the motion program for that coordinate system. If this flag is set, at the next RTI (real time interrupt), PMAC will start working through the motion program processing each command encountered. This can include multiple modal statements, calculation statements, and logical control statements. Program calculations will continue (which means no background tasks will be executed) until one of the following conditions occurs:

1. The next move, **Error! Bookmark not defined.**a **DWELL** command or a **PSET**Error! Bookmark not defined. statement is found and calculated.
2. End of, or halt to the program (e.g. **STOP**) is encountered.
3. Two jumps backward in the program (from **ENDWHILE** or **GOTO**) are performed.
4. A **WAIT** statement is encountered (usually in a **WHILE** loop).

If calculations stop on condition 1 or 2, the calculation flag is cleared and will not be set again until actual motion progresses into the next move or a new **RUN** command is given. If calculations stop on conditions 3 or 4, the flag remains set, so calculations will resume at the next RTI. In these cases, there is an empty (no-motion) loop and the motion program acts much like a PLC0 during this period. This could result in an undesired starving condition for the background cycle. If PMAC cannot finish calculating the trajectory for a move by the time execution of that move is supposed to begin, PMAC will abort the program, showing a run-time error in its status word.

Coordinate Systems

A coordinate system in PMAC is a grouping of one or more motors for synchronizing movements. A coordinate system (even with only one motor) can run a motion program; a motor cannot. Turbo PMAC can have up to 16 coordinate systems, addressed as &1 to &16, in a very flexible fashion (e.g. eight coordinate systems of one motor each, one coordinate system of eight motors, four coordinate systems of two motors each, etc.).

In general, certain motors should be moved in a coordinated fashion, put them in the same coordinate system. To move the motors independent of each other, put them in separate coordinate systems. Different coordinate systems can run separate programs at different times (including overlapping times), or even run the same program at different (or overlapping) times.

A coordinate system must be established first by assigning axes to motors in Axis Definition Statements. A coordinate system must have at least one motor assigned to an axis within that system, or it cannot run a motion program, even non-motion parts of it. When a program is written for a coordinate system, and if simultaneous motions are wanted of multiple motors, put their move commands on the same line and the moves will be coordinated.

Axis Definitions

An axis is an element of a coordinate system. It is similar to a motor, but not the same thing. An axis is referred to by letter. There can be up to nine axes in a coordinate system, selected from X, Y, Z, A, B, C, U, V, and W. An axis is defined by assigning it to a motor with a scaling factor and an optional offset (X, Y, and Z may be defined as linear combinations of three motors, as may U, V, and W). The variables associated with an axis are scaled floating-point values.

In the vast majority of cases, there will be a one-to-one correspondence between motors and axes. That is, a single motor is assigned to a single axis in a coordinate system. Even when this is the case, however, the matching motor and axis are not synonymous. The axis is scaled into engineering units and deals only with commanded positions. Except for the **PMATCH** function, calculations go only from axis commanded positions to motor commanded positions, not the other way around.

More than one motor may be assigned to the same axis in a coordinate system. This is common in gantry systems, where motors on opposite ends of the crosspiece are always trying to do the same movement. By assigning multiple motors to the same axis, a single programmed axis move in a program causes identical commanded moves in multiple motors. Commonly, this is done with two motors but up to eight motors have been used in this manner with PMAC. Remember that the motors still have independent servo loops, and that the actual motor positions will not be the same necessarily.

An axis in a coordinate system can have no motors attached to it (a phantom axis). In this case, programmed moves for that axis cause no movement, although the fact that a move was programmed for that axis can affect the moves of other axes and motors. For instance, one method to perform sinusoidal profiles on a single X-axis is to have a second, phantom Y-axis and program them together with a circular interpolation move.

Axis Definition Statements

A coordinate system is established by using axis definition statements. An axis is defined by matching a motor (which is numbered) to one or more axes (which are specified by letter).

The simplest axis definition statement is something like **#1->X**. This simply assigns motor #1 to the X-axis of the currently addressed coordinate system. When an X-axis move is executed in this coordinate system, motor #1 will make the move. The axis definition statement also defines the scaling of the axis' user units. For instance, **#1->10000X** also matches motor #1 to the X axis, but this statement sets 10,000 encoder counts to one X-axis user unit (e.g. inches or centimeters). Usually, this scaling feature is used universally. Once the scaling has been defined in this statement, the user can program the axis in engineering units without ever needing to deal with the scaling again.

Permitted Axis Names: X, Y, Z, U, V, W, A, B, C

X, Y, Z: Traditionally Main Linear Axes

- Matrix axis definition
- Matrix axis transformation
- Circular interpolation
- Cutter radius compensation

A, B, C: Traditionally Rotary Axes

- A rotates about X, B about Y, C about Z
- Position rollover (Ixx27)

U, V, W: Traditionally Secondary Linear Axes

- Matrix Axis Definition

Writing a Motion Program

1. Open a program buffer with **OPEN PROG {constant}** where **{constant}** is an integer from 1 to 32767 representing the motion program to be opened.
2. Motion program numbers 1000 and above can contain G-codes, M-codes, T-codes and D-codes for machine tool G-codes or RS-274 programming method. These buffers can be used for general PMAC code programming as long as G-codes programming is not needed in PMAC.
3. PMAC can hold up to 224 motion programs at one time. For continuous execution of programs that are larger than PMAC's memory space, a special PROG0 (the rotary motion program buffers) allows for the downloading of program lines during the execution of the program and for the overwriting of already executed program lines.
4. The **CLEAR** command empties the currently opened motion program or rotary buffer.
5. Many of the statements in PMAC motion programs are modal in nature. These include move modes, which specify what type of trajectory a move command will generate. This category includes **LINEAR**, **RAPID**, **CIRCLE**, **PVT**, and **SPLINE**.
6. Moves can be specified incrementally (distance) or absolutely (location) – individually selectable by axis – with the **INC** and **ABS** commands. Move times (**TA**, **TS**, and **TM**) and/or speeds (**F**) are implemented in modal commands. Modal commands can precede the move commands they are to affect, or they can be on the same line as the first of these move commands.
7. The move commands themselves consist of a one-letter axis-specifier followed by one or two values (constant or expression). All axes specified on the same line will move simultaneously in a coordinated fashion on execution of the line; consecutive lines execute sequentially (with or without stops in between, as determined by the mode). Depending on the modes in effect, the specified values can mean destination, distance, and/or velocity.
8. If the move times (**TA**, **TS**, and **TM**) and/or speeds (**F**) are not declared specifically in the motion program, the default parameters from the I-variables Isx87, Isx88 and Isx89 will be used instead.

Note:

Do not rely on these parameters and declare the move times in the program. This will keep the move parameters with the move commands, lessening the chances of future errors and making debugging easier.

9. In a motion program, PMAC has **WHILE** loops and **IF..ELSE** branches that control program flow. These constructs can be nested indefinitely. In addition, there are **GOTO** statements, with either constant or variable arguments. (The variable **GOTO** can perform the same function as a Case statement.) **GOSUB** statements (constant or variable destination) allow subroutines to be executed within a program. **CALL** statements permit other programs to be entered as subprograms. Entry to the subprogram does not have to be at the beginning – the statement **CALL 20.15000** causes entry into Program 20 at line **N15000**. **GOSUBs** and **CALLs** can be nested only 15 deep.
10. The **CLOSE** statement closes the currently opened buffer. This should be used immediately after the entry of a motion or rotary buffer. If the buffer is left open, subsequent statements that are intended as on-line commands (e.g. **P1=0**) will be entered into the buffer instead. It is good practice to have **CLOSE** at the beginning and end of any file to be downloaded to PMAC. When PMAC receives a **CLOSE** command, it appends a **RETURN** statement to the end of the open program buffer automatically. If any program or PLC in PMAC is structured improperly (e.g. no **ENDIF** or **ENDWHILE** to match an **IF** or **WHILE**), PMAC will report an ERR003 at the **CLOSE** command for any buffer until the problem is fixed.

Example:

```
close                   ; Close any buffer opened
delete gather           ; Erase unwanted gathered data
undefine all            ; Erase coordinate definitions in all coordinate
systems
#1->2000X               ; Motor #1 is defined as axis X

OPEN PROG 1 CLEAR       ; Open buffer to be written
LINEAR                  ; Linear interpolation
INC                     ; Incremental mode
TA100                   ; Acceleration time is 100 msec
TS0                     ; No S-curve acceleration component
F50                     ; Feedrate is 50 Units per Isx90 msec
X1                      ; One unit of distance, 2000 encoder counts
CLOSE                   ; Close written buffer, program one
```

Running a Motion Program

1. Select the Coordinate System where the motion program will be running. This is done by issuing the **&** command followed by the coordinate system number (i.e., **&1** for the coordinate system 1).
2. Use the **B{constant}** command to select the program to be run, where the **{constant}** represents the number of the motion program buffer. Use the **B** command to change motion programs and after any motion program buffer has been opened. This is not used if running the same motion program repeatedly without modification. When PMAC finishes executing a motion program, the program counter for the coordinate system is set to point to the beginning of that program automatically, ready to run it again.
3. Once pointing to the motion program to be run, issue the command to start execution of the program. For continuous execution of the program, use the **R** command (**<CTRL-R>** for all coordinate systems simultaneously). The program will execute all the way through unless stopped by command or error condition.
4. To execute just one move, or a small section of the program, use the **S** command (**<CTRL-S>** for all coordinate systems simultaneously). The program will execute to the first move **DWELL** or **DELAY**, or if it first encounters a **BLOCKSTART** command, it will execute to the **BLOCKSTOP** command.

5. When a run or step command is issued, PMAC checks the coordinate system to ensure that it is in proper working order. If it finds nothing in the coordinate system is set up properly, it will reject the command, sending a **<BELL>** command back to the host. If I6 is set to 1 or 3, it will report an error number, as well telling the reason the command was rejected. PMAC will reject a run or step command for any of the following reasons:
   - A motor in the coordinate system has both overtravel limits tripped (ERR010)
   - A motor in the coordinate system is executing a move currently (ERR011)
   - A motor in the coordinate system is not in closed-loop control (ERR012)
   - A motor in the coordinate system is not activated {Ixx00=0} (ERR013)
   - There are no motors assigned to the coordinate system (ERR014)
   - A fixed (non-rotary) motion program buffer is open (ERR015)
   - No motion program has been pointed to (ERR016)
   - After a **/** or **** stop command, a motor in the coordinate system is not at the stop point (ERR017)
6. Before starting the program, issue a **CTRL+A** command to ensure that all the motors will be potentially in closed loop and that all previous move commands are aborted. In addition, the functionality of each motor can be checked individually with jog commands before running a program. For example, **#1J^2000** will try to move motor #1 2000 encoder counts, confirming if the motor is able to run in closed loop or not.
7. All motors in the addressed coordinate system must be ready to run a motion program. Depending on the settings of Ixx24, all PMAC motors may be disabled if only one motor is having problems running in close loop.
8. No motion will occur if the feedrate override value for the current addressed coordinate system is set at zero. Check the feedrate override parameter by issuing **&1%** on the terminal window (replace 1 for the appropriate coordinate system number). If it is zero or set too low, set it to an appropriate value. The **%100** command will set it to 100%.
9. For troubleshooting purposes, change the feedrate override to a lower than 100% value. Before running the program, type **%10** to run it at a 10% rate of the programmed velocity, thus running it in slow motion. Running the program slowly will allow observing the programmed path more clearly, it will demand less calculation time from PMAC and it will prevent damages due to potentially wrong acceleration and/or feedrate parameters.
10. A motion program running in Coordinate System 1 can be stopped at any time by sending **&1A** or, for simplicity, the **CTRL+A** command will stop any running motor in PMAC.
11. If the motion of any axis becomes uncontrollable and must be stopped, a **CTRL+K** command can be issued which will kill all the motors in PMAC (disabling the amplifier enable line if connected). However, the motor will come to a stop in an uncontrollable way and may continue moving due to its own inertia.
12. In addition, a motion program can be stopped by issuing a **CTRL+Q** command. The last programmed moves in the buffer will be completed before the program quits execution. It can be resumed by issuing an **R** command.

Subroutines and Subprograms

It is possible to create subroutines and subprograms in PMAC motion programs to create well-structured modular programs with re-usable subroutines. The **GOSUBx** command in a motion program causes a jump to line label Nx of the same motion program. Program execution will jump back to the command immediately following the **GOSUB** when a **RETURN** command is encountered. This creates a subroutine.

The **CALLx** command in a motion program causes a jump to PROG x, with a jump back to the command immediately following the **CALL** when a **RETURN** command is encountered. If x is an integer, the jump is to the beginning of PROG x; if there is a fractional component to x, the jump is to line label N (**y*100,000**), where y is the fractional part of x. This structure permits the creation of special subprograms, either as a single subroutine, or as a collection of subroutines, that can be called from other motion programs.

The **PRELUDE** command allows creating an automatic subprogram call before each move command or other letter-number command in a motion program.

Passing Arguments to Subroutines

These subprogram calls are made more powerful by using the **READ** statement. The **READ** statement in the subprogram can go back up to the calling line and pick off values (associated with other letters) to be used as arguments in the subprogram. The value after an A would be placed in variable Q101 for the coordinate system executing the program. The value after a B would be placed in Q102, and so on (Z value goes in Q126). Letters N or O cannot be passed.

This structure is useful particularly for creating machine tool style programs, in which the syntax must consist solely of letter-number combinations in the parts program. Since PMAC treats the G, M, T, and D codes as special subroutine calls, the **READ** statement can be used to let the subroutine access values on the part-program line after the code.

In addition, the **READ** statement provides the capability of seeing what arguments have actually been passed. The bits of Q100 for the coordinate system are used to note whether arguments have been passed successfully; bit 0 is 1 if an A argument has been passed, bit 1 is 1 if a B argument has been passed, and so on, with bit 25 set to 1 if a Z argument has been passed. The corresponding bit for any argument not passed in the latest subroutine or subprogram call is set to 0.

Example:

```
close                       ; Close any buffer opened
delete gather               ; Erase unwanted gathered data
undefine all                ; Erase coordinate definitions in all coordinate
systems
#1->2000X                   ; Motor #1 is defined as axis X
OPEN PROG 1 CLEAR           ; Open buffer to be written
LINEAR INC TA100 TS0 F50    ;Mode and timing parameters
gosub 100 H10               ;Subroutine call passing parameter H with value 10
return                      ;End of the main program section (execution ends)
n100                        ;Subroutine section. The first subroutine is
labeled 100
read(h)                     ;Read the H parameter value passed
IF (Q100 & $80 > 0)         ;If the H parameter has been passed …
      X(Q108)               ; …use the H parameter value contained in Q108
endif
return                      ;End of the subroutine labeled 100
close                       ;End of the motion program code
```

G, M, T, and D-Codes (Machine Tool Style Programs)

PMAC permits the execution of machine tool style RS-274 (G-Code) programs by treating G, M, T, and D-codes as subroutine calls. This permits the machine tool manufacturer to customize the codes for a machine, but it requires the manufacturer to do the actual implementation of the subroutines that will execute the desired actions.

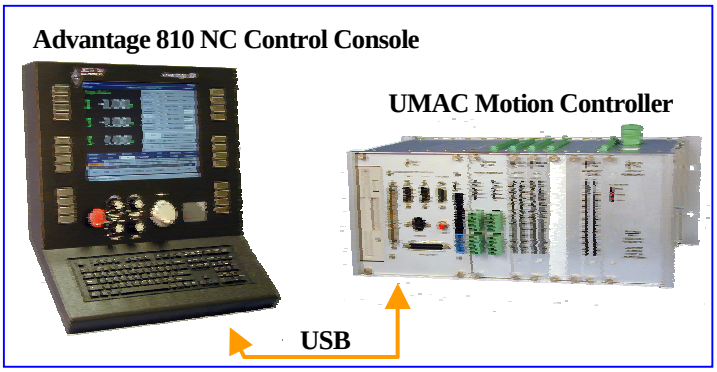
When PMAC encounters the letter G with a value in a motion program, it treats the command as a call to motion program 10n0, where n is the hundreds' digit of the value. The value without the hundred's digit (modulo 100 in mathematical terms) controls the line label within program 10n0 to which operation will jump -- this value is multiplied by 1000 to specify the number of the line label. When a return statement is encountered, it will jump back to the calling program. For example, G17 will cause a jump to N17000 of PROG 1000; G117 will cause a jump to N17000 of PROG 1010; G973.1 will cause a jump to N73100 of PROG 1090.

M-Codes operate in the same way, except they use PROG 10n1; T-codes use PROG 10n2; D-codes use PROG 10n3.

Usually, these codes have numbers within the range 0 to 99, so only PROGs 1000, 1001, 1002, and 1003 are required to execute them. To extend code numbers past 100, PROGs 1010, 1011, etc. will be required to execute them.

NC Products

The PMAC-NC software runs standard CNC parts program using a PMAC Motion controller. This software performs two important functions. It translates standard RS274 G-Codes programs into PMAC code and feeds the translated code into PMAC's memory using a perfectly synchronized communications scheme. The transfer of the program lines between the host computer and the PMAC motion controller is performed using shared DPRAM memory and either USB, PC104 or Ethernet methods. In this fashion, the size of the CNC parts program is limited only by the storage capacity in the host computer. Normally, the PMAC NC software is used with Delta Tau's Advantage line of packaged CNC systems which includes the operator control panel hardware for the machine operation.



The Advantage 810U NC controller from Delta Tau consists of two main components, the UMAC and the 810 NC operator console. This hardware combination, along with the PMAC NC software, is an excellent solution for five axes or greater, milling, turning and special purpose machines such as laser cutting and water jet cutting. A standard USBII connection links the motion controller with the 810 operator's control panel, resulting in a fast and reliable connection.

The 810 console includes all of the standard operator controls such as rotary switches for axes selection, user-definable buttons for customized IO control, feedrate override, handwheel, spindle speed and mode select. The open-architecture PC-based design runs any other Windows® compatible program, the writing of NC parts programs, reading of programs locally or through an Ethernet network or even accessing the Internet.

Linear Blended Moves

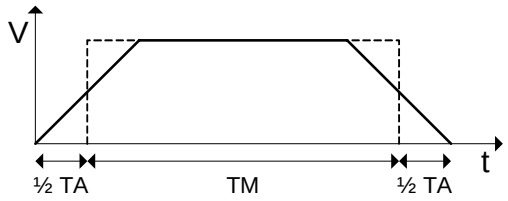
The duration of a move, or move time, is set directly by TM or indirectly from the distances and feedrate F parameters.

```
TM100          or          FRAX(X,Y)
X3 Y4
                                        TM = I5190 · √(3² + 4²) / 50 = 5000 / 50 = 100 msec
                           X3 Y4 F50    ;
```

- The acceleration time and shape is programmed with the TA and TS parameters. TA determines the overall acceleration time, which is the total time required for any change in velocity. TS determines the portion of the overall acceleration ramp that is performed in S curve mode.
- If the TA programmed results are less than twice the programmed TS, the 2*TS will be used instead.
- If the move time calculated above is less than the TA time set, the TA time will be used instead. In this case, the programmed TA (or 2*TS if TA<2*TS) will result in the minimum move time of a linearly interpolated move.
- The acceleration time TA of a blended move cannot be longer than two times the previous TM minus the previous TA, otherwise the value 2*(TM- ½ TA) will be used as the current TA instead.
- When Isx13=0, safety variables Ixx16 and Ixx17 will override these parameters if they are found to violate the programmed limits.

  If TM < TA, TM = TA
  If TA < 2*TS, TA = 2*TS
  If TA_{i+1} > 2*(TM_i- ½ TA_i), TA_{i+1} = 2*(TM_i - ½ TA_i)

Example:

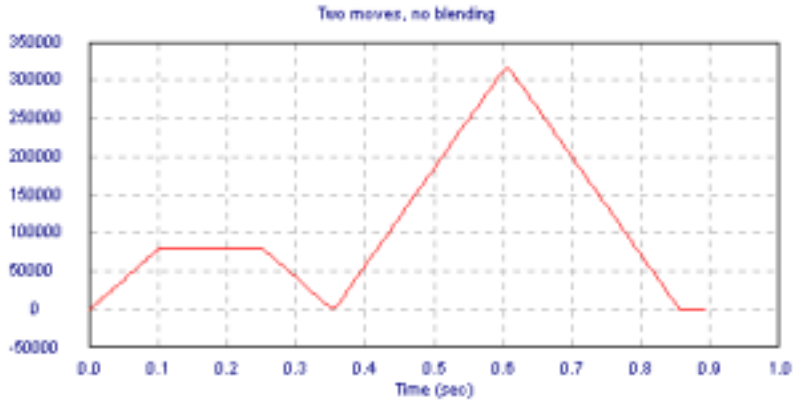


To illustrate how PMAC blends linear moves, a series of velocity Vs time profiles will be shown. The moves are defined with zero S-curve components. The concepts described here can be applied when using non-zero S-curve linear moves.

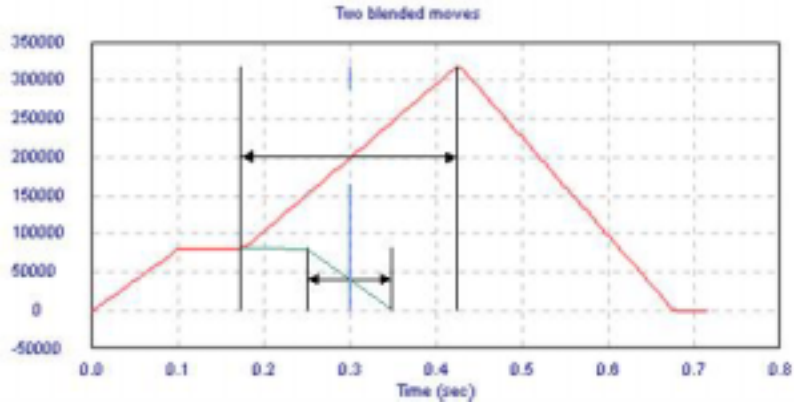
1. Consider the following motion program code:

```
close
delete gather
undefine all
&1
#1->2000x
OPEN PROG 1 CLEAR
      LINEAR          ; Linear mode
      INC             ; Incremental mode
      TA100           ; The acceleration time is 100 msec, TA1
      TS0             ; No S-curve component
      TM250           ; Move time is 250 msec, TM1
      X10             ; Move distance is 10 units, 20000 counts
      TA250           ; Acceleration \ deceleration of the blended move is
                      ; 250 msec, TA2
      X40             ; Move distance is 40 units, 80000 counts
CLOSE
```

2. The two move commands are plotted without blending, placing a **DWELL0** command in between the two moves:



3. The two moves are now plotted with the blending mode activated. To find out the blending point, trace straight lines through the middle point of each acceleration lines of both velocity profiles:

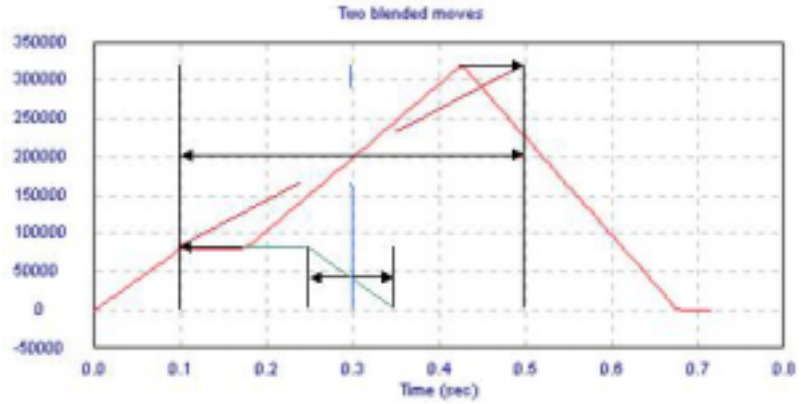


Linear Interpolated Moves Characteristics

1. The total move time is given by $\frac{TA_1}{2} + TM_1 + TM_2 + \frac{TA_2}{2} = 50 + 250 + 250 + 125 = 675\ msec$
2. The acceleration of the second blended move and be extended only up to a certain limit, 2*(TM- ½ TA).

When not using a special lookahead buffer, PMAC looks two moves ahead of actual move execution to perform its acceleration limit and can recalculate these two moves to keep the accelerations under the Ixx17 limit. However, there are cases where more than two moves, some much more than two, would have to be recalculated in order to keep the accelerations under the limit.

In these cases, PMAC will limit the accelerations as much as it can, but because the earlier moves have already been executed, they cannot be undone, and therefore the acceleration limit will be exceeded.



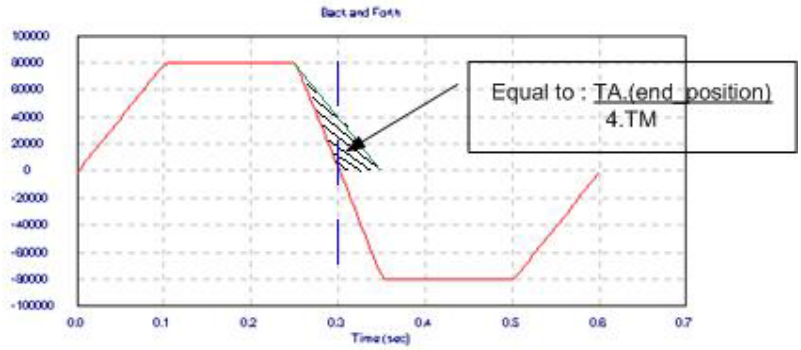
3. When performing a blended move that involves a change of direction, the programmed end point may not be reached.

   Example:

```
TA100
TM250

X10                          ; This would reach only to position =   10 − 100.10 / 4.250 = 9

X-10
```



   In order to reach the desired position, since the motor will stop when changing direction anyway, place a **DWELL0** command between moves. This command will disable blending for that particular move:

```
TA100
TM250
X10
DWELL0                       ;Temporarily disables blending between the two moves
X-10
```

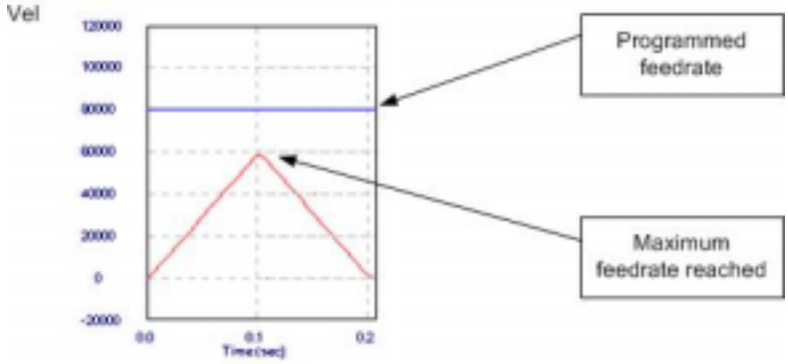
4. Since the value of TA determines the minimum time in which a programmed move can be executed, potentially it can limit the maximum velocity of motion and therefore the programmed feedrate might not be reached. This is the case for triangular (not trapezoidal) velocity profile moves types which can happen when a sequence of short distance moves is programmed.

Example:

```
close
delete gather
undefine all
&1
#1->2000X
I5190=1000
OPEN PROG 1 CLEAR
      LINEAR          ; Linear mode
      INC             ; Incremental mode
      TA100           ; Acceleration time is 100 msec, TA1
      TS0             ; No S-curve component
      F40             ; Feedrate is 40 length_units / second
                      ; TM = 3.I5190/40 = 3000/40 = 75 msec
      X3              ;
CLOSE
```

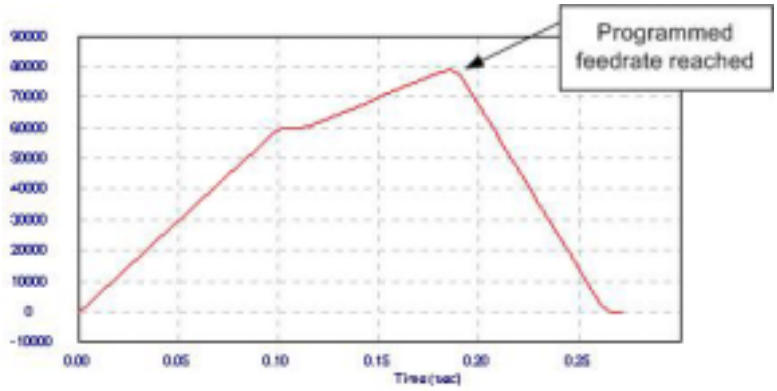
5. Since the resulting TM for the given feedrate is 75 msec and the programmed TA for this move is 100 msec, the TM used will be100 msec instead. This yields the following feedrate value instead of the programmed one:

$$F = \frac{3.I5190}{100} = \frac{3000}{100} = 30 \frac{units\ of\ distance}{second}$$



6. To be able to reach the desired velocity, a longer move could be performed split into two sections. The first move will be executed using a suitable TA to get the motor to move from rest. The second move will have a lower acceleration time TA in order to decrease the move time TM and so reach the programmed feedrate.

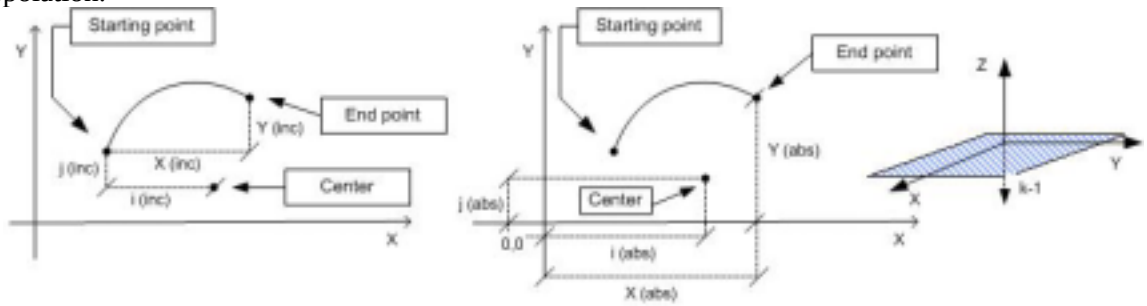
```
OPEN PROG 1
CLEAR
LINEAR
INC
TS0
F40
TA100
X3
TA75
X3
CLOSE
```



In this case, the TA parameter must be changed at the beginning and end in a series of interpolated moves. This is necessary particularly for profiles with sharp corners when more than one axis are linearly interpolated. The special lookahead buffer is an excellent solution in those cases. The lookahead feature will determine the necessary acceleration value to use in each case, maintaining the acceleration constrains limits under control. This feature allows reaching the maximum velocity along the path when possible, and reaching maximum allowed acceleration when required. This drastically increases the throughput in the machine.

Circular Interpolation

PMAC allows circular interpolation on the X, Y, and Z-axis in a coordinate system. As with linear blended moves, TA and TS control the acceleration to and from a stop and between moves. Circular blended moves can be feedrate-specified (F) or time-specified (TM), just as with linear moves. It is possible to change back and forth between linear and circular moves without stopping. The **LINEAR** command is used when linear interpolation is needed, and CIRCLE1 or CIRCLE2 is used for circular interpolation.



1. PMAC performs arc moves by segmenting the arc and performing the best cubic fit on each segment. I-Variable Isx13 determines the time for each segment. Isx13 must be set greater than zero to put PMAC into this segmentation mode so that for arc moves can be done. If Isx13 is set to zero, circular arc moves will be done in linear fashion.

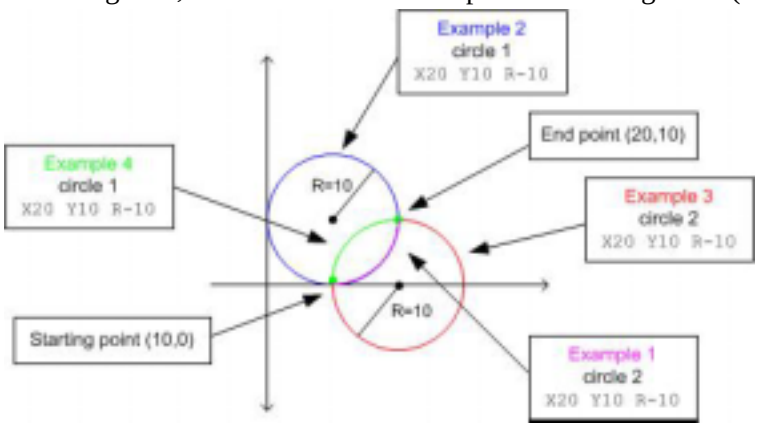
   The typical range of Isx13 for the circular interpolation mode is from 5 to 10 msec. A value of 10 msec is recommended for most applications, a lower than 10 msec Isx13 value will improve the accuracy of the interpolation (calculating points of the curve more often) but will also consume more of PMAC's total computational power.
2. When PMAC is segmenting moves automatically (Isx13 > 0), which is required for circular interpolation, the Ixx17 accelerations limits and the Ixx16 velocity limits are not observed unless a special lookahead buffer is defined.
3. Any axes used in the circular interpolation are automatically feedrate axes for circular moves, even if they were not specified in an **FRAX** command. Other axes may or may not be feedrate axes. Any non-feedrate axes commanded to move in the same move command will be linearly interpolated to finish in the same time. This permits easy helical interpolation.
4. The plane for the circular arc must be defined by the **NORMAL** command (the default – **NORMAL K-1** – defines the XY plane). This command can define planes in XYZ space only which means that the X, Y, and Z-axes can be used only for circular interpolation. Other axes specified in the same move command will be interpolated linearly to finish in the same time. The most commonly used planes are:

```
NORMAL K-1      ; XY plane -- equivalent to G17
NORMAL J-1      ; ZX plane -- equivalent to G18
NORMAL I-1      ; YZ plane -- equivalent to G19
```

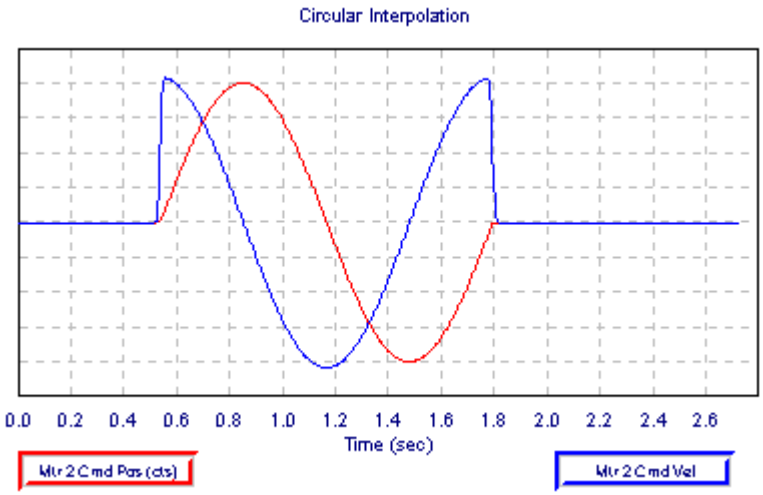
5. To put the program in circular mode, use the **CIRCLE1** program command for clockwise arcs (G02 equivalent) or **CIRCLE2** for counterclockwise arcs (G03 equivalent). **LINEAR** will restore PMAC to linear blended moves. Once in circular mode, a circular move is specified with a move command designating the move endpoint and either the vector to the arc center or the distance (radius) to the center. The endpoint may be specified either as a position or as a distance from the starting point, depending on whether the axes are in absolute (**ABS**) or incremental (**INC**) mode (individually specifiable).

```
X{Data} Y{Data} R{Data}          ;Radius of the circle is given
X{Data} Y{Data} I{Data} J{Data}  ;Center coordinates of the circle are given
```

6. If the vector method of locating the arc center is used, the vector is specified by its I, J, and K components (I specifies the component parallel to the X axis, J to the Y axis, and K to the Z axis). This vector can be specified as a distance from the starting point (i.e. incrementally), or from the XYZ origin (i.e. absolutely). The choice is made by specifying R in an ABS or INC statement (e.g. **ABS(R)** or **INC(R)**. This affects I, J, and K specifiers together. (**ABS** and **INC** without arguments affect all axes, but leave the vectors unchanged.) The default is for incremental vector specification.
7. PMAC's convention is to take the short arc path if the R value is positive, and the long arc path if R is negative:
   - If the value of R is positive, the arc to the move endpoint is the short route (<=180 degrees)
   - If the value of R is negative, the arc to the move endpoint is the long route (>=180 degrees)



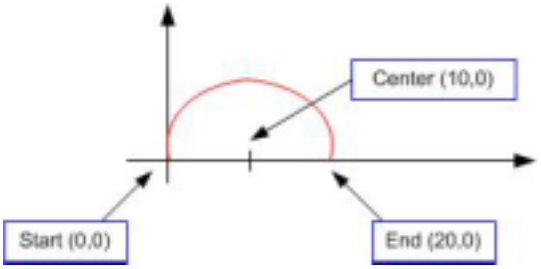
8. When performing a circular interpolation, the individual axes describe a position Vs time profile close to a sine and cosine shape. This is true for their velocity and acceleration profiles also. Therefore, circular interpolation makes an ideal feature to described trigonometric profiles. Furthermore, the period (and so the frequency) of the sine or cosine profiles are set by the total move time given by TA+TM.




```
close
delete gather
undefine all
&1
#2->2000Y    ;X is phantom
open prog1 clear
inc
inc (r)
ta300
ts0
tm1000       ;TA+TM is period
i13=10
normal k-1   ;X-Y plane
circle1      ;Clockwise
x0 y0 i10    ;Complete circle
close
b1r          ;Run this program
```

Example:

```
I5113=10        ;Move Segmentation Time
NORMAL K-1      ;XY plane
INC             ;Incremental End Point
                ;definition
INC (R)         ;Incremental Center
                ;Vector definition
CIRCLE 1        ;Clockwise circle
X20 Y0 I10 J0   ;Arc move
```



Splined Moves

Turbo PMAC can perform two types of cubic splines (cubic in terms of the position-vs.-time equations) to blend a series of points on an axis. Its **SPLINE1** mode is a uniform non-rational cubic B-spline and its **SPLINE2** mode is a non-uniform non-rational cubic B-spline. Of course, it can perform either spline for all of the axes simultaneously. Splining is particularly suited to odd (non-Cartesian) geometries, such as radial tables and rotary-axis robots, where there are odd axis profile shapes even for regular tip movements.

In **SPLINE1** mode, a long move is split into equal-time segments, each of TM or TA time (depending on the setting of global variable I42). Each axis is given a destination position in the motion program for each segment with a regular move command line like X1000Y2000. Looking at the move command before this and the move command after this, Turbo PMAC creates a cubic position-vs.-time curve for each axis so that there is no sudden change of either velocity or acceleration at the segment boundaries. The commanded position at the segment boundary may be relaxed slightly to meet the velocity and acceleration constraints.

Turbo PMAC's **SPLINE2** mode is similar to the **SPLINE1** mode, except that the requirement that the **TA** spline segment time remain constant is removed. The removal of this constraint makes the **SPLINE2** mode a non-uniform, non-rational cubic B-spline, whereas the **SPLINE1** mode is a uniform, non-rational cubic B-spline.

PVT-Mode Moves

To have more direct control over the trajectory profile, Turbo PMAC offers Position-Velocity-Time (PVT) mode moves. In these moves, the axis states are specified directly at the transitions between moves (unlike in blended moves). This requires more calculation by the host, but allows tighter control of the profile shape. For each piece of a move, the end position or distance, the end velocity, and the piece time are specified.

Turbo PMAC is put in this mode with the program statement **PVT{data}**, where **{data}** is a floating-point constant, variable, or expression, representing the piece time in milliseconds. The move time may be changed between moves, either with another **PVT{data}** statement, with a **TM{data}** statement if I42 = 0, or a **TA{data}** statement if I42 = 1. The program is taken out of this mode with another move mode statement (e.g., **LINEAR**, **RAPID**, **CIRCLE**, **SPLINE**).

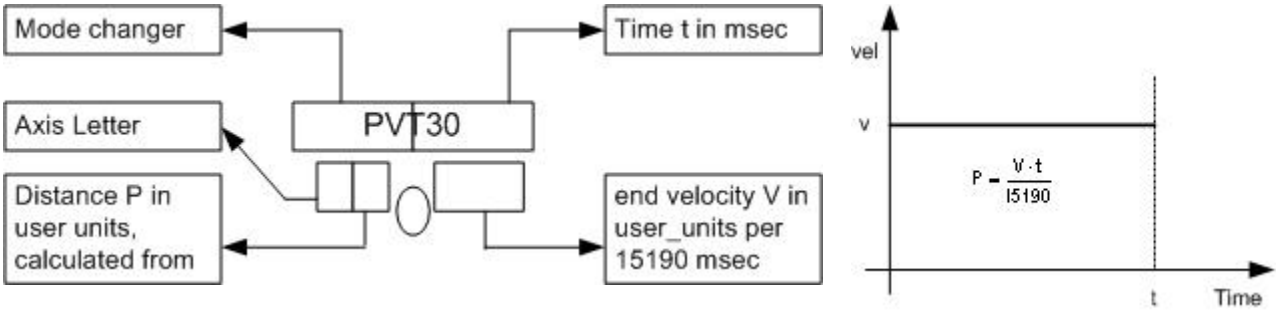
A PVT mode move is specified for each axis to be moved with a statement of the form **{axis}{data}:{data}**, where **{axis}** is a letter specifying the axis, the first **{data}** is a value specifying the end position or the piece distance, depending on whether the axis is in absolute or incremental mode, respectively, and the second **{data}** is a value representing the ending velocity.

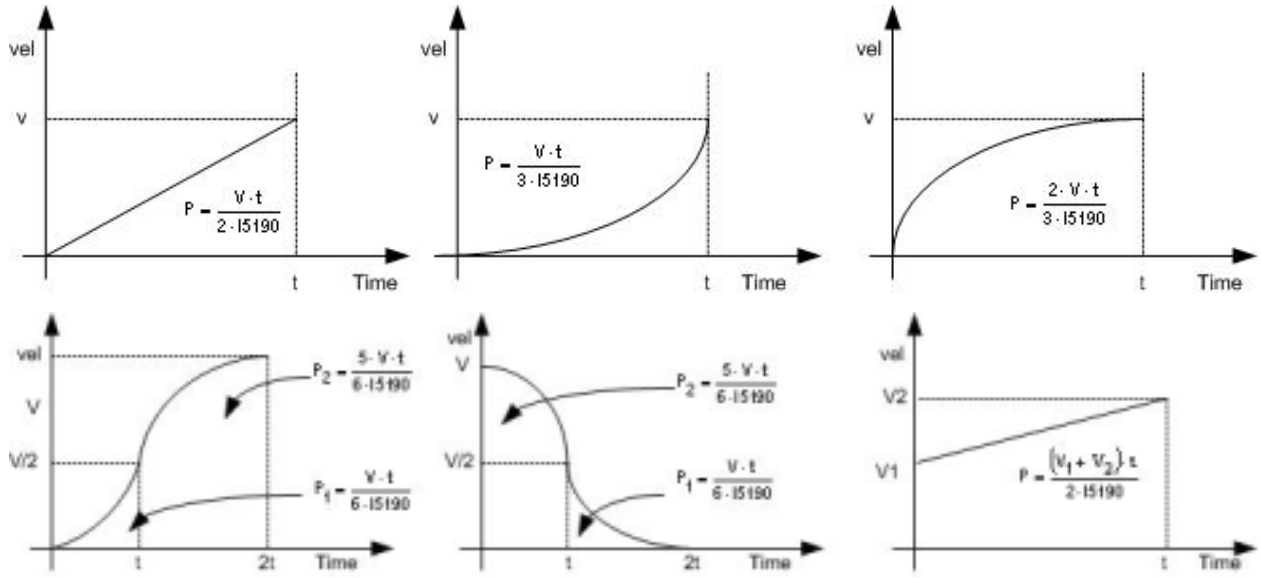
The units for position or distance are the user length or angle units for the axis, as set in the Axis Definition statement. The units for velocity are defined as length units divided by time units, where the length units are the same as those for position or distance, and the time units are defined by variable Isx90 for the coordinate system (feedrate time units). The velocity specified for an axis is a signed quantity.

From the specified parameters for the move piece, and the beginning position and velocity (from the end of the previous piece), Turbo PMAC computes only the third-order position trajectory path to meet the constraints. This results in linearly changing acceleration, a parabolic velocity profile, and a cubic position profile for the piece.

Since a non-zero end velocity for the move can be specified (directly or indirectly), it is not a good idea to step through a program of transition-point moves, and great care must be exercised in downloading these moves in real time. With the use of the **BLOCKSTART** and **BLOCKSTOP** statements surrounding a series of PVT moves, the last of which has a zero end velocity, it is possible to use a **STEP** command to execute only part of a program.

The PVT mode is the most useful for creating arbitrary trajectory profiles. It provides a building block approach to putting together parabolic velocity segments to create whatever overall profile is desired. The following diagrams show common velocity segment profiles. PVT mode can create any profile that any other move mode can.





Replace I5190 for the appropriate Isx90 variable according to coordinate system sx - 50.

Example:

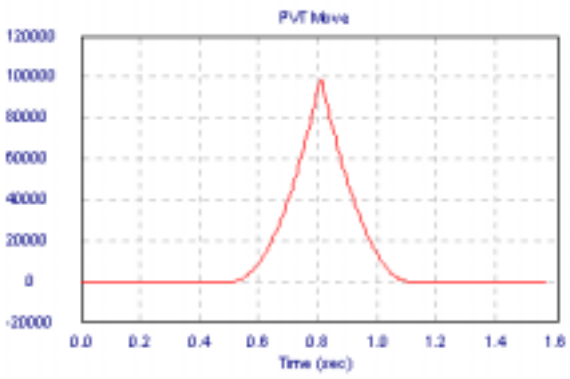
```
close delete gather undefine all
&1 #1->2000X
OPEN PROG 1 CLEAR
INC
PVT300    ;Time is 300 msec per section
```

$$P = \frac{50\ user_units}{I5190\ msec} \cdot \frac{300\ msec}{3} = \frac{15000}{3000} = 5\ user_units$$

```
X5:50     ;
```

$$P = \frac{50\ user_units}{I5190\ msec} \cdot \frac{300\ msec}{3} = \frac{15000}{3000} = 5\ user_units$$

```
X5:0      ;
CLOSE
```



Turbo PMAC Lookahead Function

Turbo PMAC can perform highly sophisticated lookahead calculations on programmed trajectories to ensure that the trajectories do not violate specified maximum quantities for the axes involved in the moves. When the lookahead function is enabled, Turbo PMAC will scan ahead in the programmed trajectories, looking for potential violations of its position, velocity, and acceleration limits. If it sees a violation, it will then work backward through the pre-computed buffered trajectories, slowing down the parts of these trajectories necessary to keep the moves within limits. Any application requiring quick reaction to external conditions should not use lookahead. In addition, any application requiring precise synchronization to external motion, such as those using PMAC's external time-base feature should not use lookahead.

The following list explains the steps required for setting up and using the lookahead function on the Turbo PMAC.

1. Assign all desired motors to the coordinate system with axis definition statements.
2. Set Ixx13 and Ixx14 positive and negative position limits, plus Ixx41 desired position-limit band, in counts for each motor in coordinate system. Set bit 15 of Ixx24 to 1 to enable desired position limits.
3. Set Ixx16 maximum velocity in counts/msec for each motor in coordinate system.

4. Set Ixx17 maximum acceleration in counts/msec^2 for each motor in coordinate system.
5. Set Isx13 segmentation time in msec for the coordinate system to minimum programmed move block time or 10 msec, whichever is less.
6. Compute maximum stopping time for each motor as Ixx16/Ixx17.
7. Select motor with longest stopping time.
8. Compute number of segments needed to look ahead as this stopping time divided by (2 * Isx13):

$$\frac{Ixx16}{2 \cdot Isx13 \cdot Ixx17}$$

9. Multiply the segments needed by 4/3 (round up if necessary) and set the Isx20 lookahead length parameter to this value:

$$Isx20 = \left\lceil \frac{4 \cdot Ixx16}{6 \cdot Isx13 \cdot Ixx17} \right\rceil$$

10. If the application involves high block rates, set the Isx87 default acceleration time to the minimum block time in msec; the Isx88 default S-curve time to 0.
11. If the application does not involve high block rates, set the Isx87 default acceleration time and the Isx88 default S-curve time parameters to values that give the desired blending corner size and shape at the programmed speeds.
12. Store these parameters to non-volatile memory with the **SAVE** command for them to be an automatic part of the machine state.
13. After each power-up/reset, send the card a **DEFINE LOOKAHEAD {# of segments},{# of outputs}** command for the coordinate system, where **{# of segments }** is equal to Isx20 plus any segments for which backup capability is desired, and **{# of outputs}** is at least equal to the number of synchronous M-Variable assignments that may need to be buffered over the lookahead length.
14. Load the motion program into the Turbo PMAC. Nothing special needs to be done to the motion program. The motion program defines the path to be followed; the lookahead algorithm may reduce the speed along the path, but it will not change the path.
15. Run the motion program, and let the lookahead algorithm do its work. The following commands will apply:
    - Set Isx21 to 4 or issue **** for a quick-stop command. This decelerates all motors at the maximum allowed rate.
    - Set Isx21 to 7 or issue **<** for a back-up command. This moves the motors up to the oldest point in the buffer.
    - Set Isx21 to 6 or issue **>** for a resume-forward command. This resumes execution from the lookahead buffer.
    - Issue **/** to quit program at the end of the block being added in the buffer. This is a block-stop command.
    - Issue **Q** to quit program execution at the end of the last calculated block.
    - Issue **H** to bring the feedrate override to zero at an Isx95 rate. This holds program execution.
    - Issue **A** to immediately abort program execution and decelerate each motor at the Ixx15 rate.
    - Issue **R** (run) or **S** (step) to resume program execution.

Turbo PMAC Kinematic Calculations

Turbo PMAC provides structures to enable easy implementation and execution of complex kinematic calculations. Kinematic calculations are required when there is a non-linear mathematical relationship between the tool-tip coordinates and the matching positions of the actuators (joints) of the mechanism, typical in non-Cartesian geometries. Most commonly, they are used in robotic applications, but can be used with other types of actuators that are not considered robotic. For example, in 4-axis or 5-axis machine tools with one or two rotary axes, they should be programmed so that the cutter-tip path will let the controller compute the necessary motor positions.

The forward-kinematic calculations use the joint (motor) positions as input, and convert them to tool-tip coordinates. The inverse-kinematic calculations use the tool-tip positions as input, and convert them to joint (motor) coordinates. Turbo PMAC implements the execution of kinematic calculations through special forward-kinematic and inverse-kinematic program buffers. Each coordinate system can have one of each of these program buffers, and the algorithms in them can be executed at the required times automatically called as subroutines from the motion program.

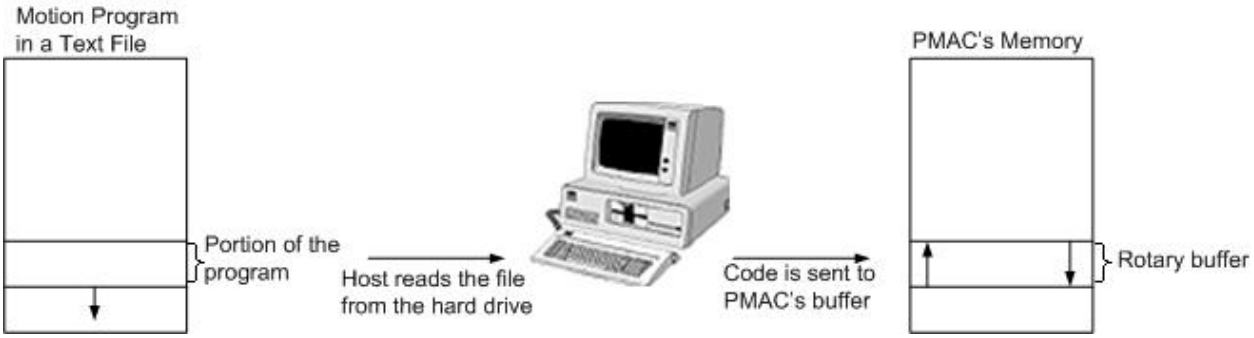
1. The on-line **OPEN FORWARD** command opens the forward-kinematic buffer for the addressed coordinate system for entry. The on-line **CLEAR** command erases any existing contents of that buffer. The forward-kinematic equations defined are placed in this buffer. The on-line **CLOSE** command stops entry into the buffer.
2. Before any execution of the forward-kinematic program, Turbo PMAC will place the present commanded motor positions for each motor xx in the coordinate system into global variable Pxx. These are floating-point values with units of counts. The program can then use these variables as the inputs to the calculations.
3. The results of the forward-kinematic equations in the program are placed in variables Q1 to Q9 for the axis letters A, B C, U, V, W, X, Y and Z respectively. Then, after any execution of the forward-kinematic program, Turbo PMAC will take the values in Q1 – Q9 for the coordinate system in the engineering units, and copy these into the 9-axis target position registers for the coordinate system. These values can be monitored through the suggested M-Variables Msx41 to Msx49 for Q1 to Q9 respectively. The basic purpose of the forward-kinematic program then is to take the joint-position values found in P1 – P32 for the motors used in the coordinate system, compute the matching tip-coordinate values, and place them in variables in the Q1 – Q9 range.
4. The on-line **OPEN INVERSE** command opens the inverse-kinematic buffer for the addressed coordinate system for entry. The on-line **CLEAR** command erases any existing contents of that buffer. The inverse-kinematic equations defined are placed in this buffer. The on-line **CLOSE** command stops entry into the buffer.
5. Before any execution of the inverse-kinematic program, Turbo PMAC will place the present axis target positions for each axis in the coordinate system into variables in the range Q1 – Q9 for the coordinate system. These are floating-point values, in engineering units. The program can then use these variables as the inputs to the calculations.
6. The results of the inverse-kinematic equations in the program are placed in variables Pxx for the corresponding motor xx in the coordinate system. After any execution of the inverse-kinematic program, Turbo PMAC will read the values in those variables Pxx that correspond to motors xx in the coordinate system with axis-definition statements of **#xx->I**. These are floating-point values, and Turbo PMAC expects to find them in the raw units of counts. Turbo PMAC will copy these values automatically into the target position registers for these motors (suggested M-Variable Mxx63), where they are used for fine interpolation. The basic purpose of the inverse-kinematic program then is to take the tip-position values found in Q1 – Q9 for the axes used in the coordinate system, compute the matching joint-coordinate values, and place them in variables in the P1 – P32 range.

7. In addition, the Turbo PMAC can support the conversion of velocities from tip space to joint space in the inverse-kinematic program to enable the use of PVT mode with kinematic calculations. With PVT-mode moves, the position calculations are completed the same as any other move mode. An additional set of velocity-conversion calculations must be done also. The commanded velocity values will be placed in variables Q11 to Q19 for the corresponding axis letters A, B C, U, V, W, X, Y and Z respectively. Turbo PMAC will set Q10 to 1 in this mode as a flag to the inverse-kinematic program to use these axis (tip) velocity values to compute motor (joint) velocity values.
8. Once the forward-kinematic and inverse-kinematic program buffers have been created for a coordinate system, Turbo PMAC will execute them automatically at the proper times, once the kinematic calculations have been enabled by setting coordinate system I-Variable Isx50 to 1. No modification to a motion program is required for access to the kinematic programs at the proper time.
9. If the special lookahead buffer for the coordinate system is active (**LINEAR** or **CIRCLE**-mode moves with the lookahead buffer defined for the coordinate system, Isx13 > 0, and Isx20 > 0), the internal spline segments computed for the joints (motors) are entered into the lookahead buffer automatically. Here they are checked continually against position, velocity, and acceleration limits for each motor. This permits Turbo PMAC to check and correct for the motion anomalies that occur near singularities automatically.

Other Programming Features

Rotary Motion Program Buffers

PMAC has a limited memory space for motion and PLCs programs. The rotary motion program buffers allow running motion programs larger than the available space in PMAC's memory.



Communication routines provided by Delta Tau have the necessary code to implement this feature in a host computer.

Internal Timebase, the Feedrate Override

Each coordinate system has its own time base that controls the speed of interpolated moves in that coordinate system.

If Isx93 is set at default, this parameter could be changed by different means:

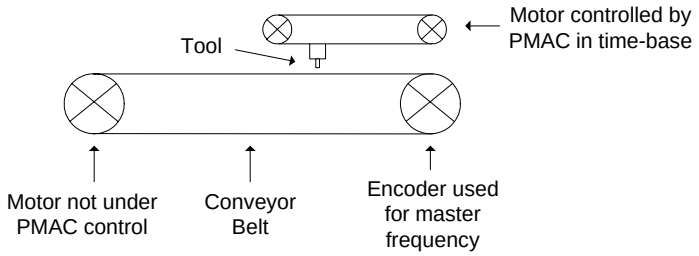
- *%n, where 0 < n < 100* `; Online or CMD command that runs all motion commands in slow-motion`
- *%n, where 100 < n ≤ 225* `; Online or CMD command that runs all motion commands in fast-motion`
- *%0* `; Online or CMD command that "freezes" all motions and timing in that C.S`
- *%100* `; Online or CMD command that restores the real-time reference (1 msec = 1 msec)`
- `M197 = I10` `; Suggested M-variable for time base change. Equal to I10 is 100%, equal to 0 is 0%.`

The variable Isx94 controls the rate at which the time base changes: $Isx94 = \frac{I10^2}{t \cdot 2^{23}}$, where t is the slew rate time in msec.

External Time-Base Control (Electronic Cams)

The motion of each coordinate system can be referenced to an external clock in the form of a frequency generated by an external encoder. At each servo cycle, the time reference used for the servo algorithms is adjusted by this external frequency source, thus controlling the rate of execution of moves and programs. The encoder register receiving the input frequency and the relationship between the input frequency and the program rate of execution must be specified. This not only varies the speed of moves in proportion to the input frequency (all the way down to zero frequency), but also keeps total position synchronization. A simple change of variable Isx93 allows switching between the internal and external time-base reference.

Example:
Material passing through a conveyor belt is processed on the fly with external time-base synchronization.



Position Following (Electronic Gearing)

PMAC has several methods of coordinating the axes under its control to axes not under its control. The simplest method is basic position following. This is a motor-by-motor function, not a coordinate system function like time-base following. An encoder signal from the master axis (which is not under PMAC's control) is fed into one of PMAC's encoder inputs. Typically, this master signal is from either an open-loop drive or a handwheel knob. Ixx05 and Ixx06 control this function.

Cutter Radius Compensation

PMAC provides the capability of performing cutter (tool) radius compensation on the moves it performs. This compensation can be performed among the X, Y, and Z axes, which should be physically perpendicular to each other. The compensation offsets the described path of motion perpendicular to the path by a programmed amount. Cutter radius compensation is valid only in **LINEAR** and **CIRCLE** move modes. The moves must be specified by **F** (feedrate), not **TM** (move time). PMAC must be in move segmentation mode (Isx13 > 0) to do this compensation. (Isx13 > 0 is required for **CIRCLE** mode anyway). Program commands **CC0**, **CC1**, **CC2**, **CCR** and **NORMAL** control this feature.

Synchronizing PMAC to other PMACs

When multiple PMACs are used together, intercard synchronization is maintained by passing the servo clock signal from the first card to the others. With careful writing of programs, this permits complete coordination of axes on different cards.

Axis Transformation Matrices

PMAC provides the capability to perform matrix transformation operations on the X, Y, and Z-axes of a coordinate system. These operations have the same mathematical functionality as the matrix forms of the axis definition statements, but these can be changed on the fly in the middle of programs; the axis definition statements can be fixed for a particular application. The matrix transformations permit translation, rotation, scaling, mirroring, and skewing of the X, Y, and Z-axes. They can be useful for English/metric conversion, floating origins, making duplicate mirror images, repeating operations with angle offsets, and more. The matrices are implemented through Q-Variables and the **DEFINE TBUF**, **TSEL**, **TINIT**, **ADIS**, **IDIS**, **AROT** and **IROT** commands.

Position-Capture and Position-Compare Functions

The position-capture function latches the current encoder position at the time of an external event into a special register. It is executed totally in hardware, without the need for software intervention (although it is set up, and later serviced, in software). This means that the only delays in the capture are the hardware gate delays (negligible in any mechanical system), so this provides an incredibly accurate capture function. The move-until-trigger functions (either jog or motion program) conveniently use the position capture feature for continuous motions until a trigger condition is reached.

Essentially, the position-compare feature is the opposite of the position-capture function. Instead of storing the position of the counter when an external signal changes, it changes an external signal when the counter reaches a certain position.

Learning a Motion Program

It is possible to have PMAC learn lines of a motion program using the on-line **LEARN** command. In this operation, the axes are moved to the desired position and the command is given to PMAC. PMAC then adds a command line to the open motion program buffer that represents this position. This process can be repeated to learn a series of points. The motors can be open loop or closed loop as they are moved around.

PLC PROGRAMS

Motion programs operate sequentially and synchronously in time, and any move command takes a specified amount to execute before succeeding program lines are executed:

Example:

```
OPEN PROG 1 CLEAR              ; Open program buffer
I5113=0                        ; Two moves ahead of calculation
LINEAR INC TA100 TS0 F50       ; Mode commands
X1                             ; First Move Command
X1                             ; Second Move Command
X1                             ; Third Move Command
M1=1                           ; This line will be executed only after the
                               ; first move is completed
CLOSE                          ; Close written buffer, program one
```

In addition to the motion programs, Turbo PMAC has 64 PLC programs that operate asynchronously and with rapid repetition (32 compiled PLC programs as well as 32 interpreted [uncompiled] PLC programs.) They are called PLC programs because they perform many of the same functions as hardware programmable logic controllers. PLC programs have most of the same logical constructs as the motion programs, but no move-type statements.

PLC programs are useful particularly for monitoring analog and digital inputs, setting outputs, sending messages, monitoring motion parameters, issuing commands as if from a host, changing gains, and starting and stopping moves. By their complete access to Turbo PMAC variables and IO, and their asynchronous nature, they become powerful adjuncts to the motion control programs.

PLC programs are numbered 0 through 31 for both the compiled and uncompiled PLCs. This means that both a compiled PLC n and an uncompiled PLC n can be stored in Turbo PMAC. PLC program 0 is a special, fast program that operates at the end of the servo-interrupt cycle with a frequency specified by variable I8 (every I8+1 servo cycles). This program is meant for a few time-critical tasks and it should be kept small because its rapid repetition can steal time from other tasks.

PLC programs 1-31 are executed in the background cycle. Each PLC program executes one scan (to the end or to an **ENDWHILE** statement) uninterrupted by any other background task (although it can be interrupted by higher priority tasks). In between each PLC program, PMAC will do its general housekeeping and respond to a host command, if any. In between each scan of each individual background interpreted PLC program, PMAC will execute one scan of all active background compiled PLCs. This means that the background compiled PLCs execute at a higher scan rate than the background interpreted PLCs. For example, if there are seven active background interpreted PLCs, each background compiled PLC will execute seven scans for each scan of a background interpreted PLC. At power-on/reset PLCC programs run after the first PLC program runs. These are the suggested uses of the PLC buffers:

- **PLC0 \ PLCC0:** PLC0 is a special fast program that operates at the end of the servo interrupt cycle with a frequency specified by variable I8 (every I8+1 servo cycles). This program is meant for a few time-critical tasks and it should be kept small because its rapid repetition can steal time from other tasks. A PLC 0 that is too large can cause unpredictable behavior and can even trip PMAC's Watchdog Timer by starving background tasks of time to execute. For faster execution, define PLCC0 instead.
- **PLC1:** This is the first code that PMAC will run on power-up, assuming that I5 was saved with a value of 2 or 3. This makes PLC1 the appropriate PLC to initialize parameters, perform commutated motors phase search and run motion programs. In addition, PLC1could disable itself and the end of execution or disable other PLCs before they start running.

- **PLC2-31:** PLC programs are useful particularly for monitoring analog and digital inputs, setting outputs, sending messages, monitoring motion parameters, issuing online commands, changing servo gains, and starting and stopping moves. Because of their complete access to all PMAC variables and IO and their asynchronous nature, they become powerful adjuncts to the motion control programs.
- **PLCC1 to PLCC31:** Compiled PLCs are convenient for their faster execution compared to regular PLCs. Since the execution rate of compiled PLCs is the same as some of the safety checks (following error limits, hardware overtravel limits, software overtravel limits, and amplifier faults), PLCCs are ideal for replacing or complementing them. However, due to their limited allocated memory space, PLCCs should be reserved for faster execution critical tasks only.

Entering a PLC Program

PLCs are programmed in a text editor and downloaded to PMAC with the Pewin32-Pro software.

Before writing the PLC, make sure that memory has not been tied up in data gathering or program trace buffers by issuing **DELETE GATHER** and **DELETE TRACE** commands.

1. Open the buffer for entry with the **OPEN PLC n** statement, where **n** is the buffer number. Next, if there is anything currently in the buffer that should not be kept, it should be emptied with the **CLEAR** statement. (PLC buffers may not be edited on the PMAC itself; they must be cleared and re-entered.) If the buffer is not cleared, new statements will be added onto the end of the buffer.
2. When finished, close the buffer with the **CLOSE** command. Opening a PLC program buffer disables that program automatically. After it is closed, it remains disabled, but it can be re-enabled again with the **ENABLE PLC n** command, where n is the buffer number from 0 to 31. In addition, I5 must be set properly for a PLC program to operate.
3. At the closing, PMAC checks to make sure all IF branches and WHILE loops have been terminated properly. If not, it reports an error, and the buffer is inoperable. Then correct the PLC program in the host and re-enter it (clearing the erroneous block in the process). This process is repeated for all of the PLC buffers to be used.

Because all PLC programs in PMAC's memory are enabled at power-on/reset, I5 should be saved as 0 in PMAC's memory when developing PLC programs. This will allow PMAC to be reset and have no PLCs running (an enabled PLC runs only if I5 is set properly) and recover more easily from a PLC programming error.

Structure Example:

```
CLOSE
DELETE GATHER
DELETE TRACE
OPEN PLC n CLEAR
      {PLC statements}
CLOSE
ENABLE PLC n
```

To erase an uncompiled PLC program, open the buffer, clear the contents, and then close the buffer again.

Example: `OPEN PLC 5 CLEAR CLOSE`

PLC Program Structure

When writing a PLC program, it is important to remember that each PLC program is effectively in an infinite loop; it will execute repeatedly until told to stop. (These are called PLCs because of the similarity in how they operate to hardware Programmable Logic Controllers – the repeated scanning through a sequence of operations and potential operations.)

Calculation Statements

Much of the action taken by a PLC is done through variable value assignment statements: **{variable}={expression}**. The variables can be I, P, Q, or M types, and the action thus taken can affect many things inside and outside the card. Perhaps the simplest PLC program consists of one line: **P1=P1+1**. Every time the PLC executes, usually hundreds of times per second, P1 will increment by one. Of course, these statements can get a lot more involved. Consider this statement:

```
P2=M162/(I108*32*10000)*COS (M262/(I208*32*100))
```

This statement could be converting radial (M162) and angular (M262) positions into horizontal position data, scaling at the same time. Because it updates this frequently, whoever needs access to this information (e.g., host computer, operator, motion program) can be assured of having current data.

Conditional Statements

Most action in a PLC program is conditional, dependent on the state of PMAC variables, such as inputs, outputs, positions, counters, etc. Action can be level-triggered or edge-triggered; both can be done, but the techniques are different.

Level-Triggered Conditions

A branch controlled by a level- triggered condition is easier to implement. Taking our incrementing variable example and making the counting dependent on an input assigned to variable M7000, we have:

```
IF (M7000=1)
      P1=P1+1
ENDIF
```

As long as the input is true, P1 will increment several hundred times per second. When the input goes false, P1 will stop incrementing.

Edge-Triggered Conditions

To increment P1 once for each time M7000 goes true (triggering on the rising edge of M7000 sometimes called a one-shot or latched). A compound condition will trigger the action, then as part of the action, set one of the conditions false, so the action will not occur on the next PLC scan. The easiest way to do this is with a shadow variable which will follow the input variable value. Action is taken only when the shadow variable does not match the input variable. The code would become:

```
IF (M7000=1)
      IF (P11=0)
            P1=P1+1
            P11=1
      ENDIF
ELSE
      P11=0
ENDIF
```

Make sure that P11 can follow M7000 both up and down. Set P11 to 0 in a level-triggered mode; this could have done as edge-triggered as well, but it does not matter as far as the final outcome of the routine is concerned, it is about the same in calculation time and it saves program lines.

WHILE Loops

Normally a PLC program executes all the way from beginning to end within a single scan. The exception to this rule occurs if the program encounters a true **WHILE** condition. In this case, the program will execute down to the **ENDWHILE** statement and exit this PLC. After cycling through all of the other PLCs, it will re-enter this PLC at the **WHILE** condition statement, not at the beginning. This process will repeat as long as the condition is true. When the **WHILE** condition goes false, the PLC program will skip past the **ENDWHILE** statement and proceed to execute the rest of the PLC program.

To increment the counter as long as the input is true and prevent execution of the rest of the PLC program, program:

```
WHILE (M7000=1)
      P1=P1+1
ENDWHILE
```

This structure makes it easier to hold up PLC operation in one section of the program, so other branches in the same program do not have to have extra conditions and they do not execute when this condition is true. Use this instead of an IF condition.

COMMAND and SEND Statements

One of the most common uses of PLCs is to start motion programs and jog motors by means of command statements.

Some **COMMAND** action statements should be followed by a **WHILE** condition to ensure they have taken effect before proceeding with the rest of the PLC program. This is true if a second **COMMAND** action statement that requires the first **COMMAND** action statement to finish will follow. (Remember, **COMMAND** action statements are processed only during the communications section of the background cycle.) For example, to stop any motion in a coordinate system and start motion program 10, the following PLC could be used:

```
M5187->Y:$00203F,17,1         ; &1 In-position bit (AND of motors)
OPEN PLC3 CLEAR
IF (M7000=1)                  ; input is ON
      IF (P11=0)              ; input was not ON last time
            P11=1             ; set latch
            COMMAND"&1A"      ; ABORT all motion
            WHILE (M5187=0)   ; wait for motion to stop.
            ENDW
            COMMAND"&1B10R"   ; start program 10
      ENDIF
ELSE
      P11=0                   ; reset latch
ENDIF
CLOSE
```

Any **SEND**, **COMMAND**, or **DISPLAY** action statement should be done only on an edge-triggered condition, because the PLC can cycle faster than these operations can process their information and the communications channels can get overwhelmed if these statements are executed on consecutive scans through the PLC.

```
IF (M7000=1)                  ; input is ON
      IF (P11=0)              ; input was not ON last time
            COMMAND"#1J+"     ; JOG motor
            P11=1             ; set latch
      ENDIF
ELSE
      P11=0                   ; reset latch
ENDIF
```

Timers

Timing commands like **DWELL** or **DELAY** are reserved only to motion programs and cannot be used for timing purposes on PLCs. Instead, each active coordinate system (those numbered from 1 to I68+1) has two timer variables running: Isx11 and Isx12. These two 24-bit registers are timers for any general-purpose use and can be used in any coordinate system. A value is written to the timer I-Variable representing the desired time in servo cycles (multiply milliseconds by 8,388,608/I10); then the PLC waits until the I-Variable is less than 0.

Example:

```
M7000->Y:$078C00,0,1          ; General-Purpose Output1 (redefine if necessary)
OPEN PLC3 CLEAR
M7000 = 0                     ; Reset Output1 before start
I5111 = 1000*8388608/I10      ; Set timer to 1000 msec, 1 second
WHILE (I5111>0)               ; Loop until counts to zero
ENDWHILE
M7000 = 1                     ; Set Output 1 after time elapsed
DIS PLC3                      ; disables PLC3 after execution
CLOSE
```

If more timers are need, use the technique in memory address X:0. This 24-bit register counts once per servo cycle. Store a starting value for this, then in each scan, subtract the starting value from the current value and compare the difference to the amount of time to wait.

Example:

```
M7000->Y:$078C00,0,1  ; General-Purpose Output1 (redefine if necessary)
M0->X:$0,24           ; Servo counter register
M85->X:$6055, 24      ; Location of P85 (I46 = 0 or 2) used as a spare register
M86->X:$6056, 24      ; Location of P86 (I46 = 0 or 2) used as a spare register
OPEN PLC 3 CLEAR
M7000=0               ; Reset Output1 before start
M85=M0                ; Initialize timer
M86=0
WHILE(M86<1000)       ; Time elapsed less than specified time?
      M86=M0-M85
      M86=M86*I10/8388608     ; Time elapsed so far in milliseconds
ENDWHILE
M7000=1               ; Set Output 1 after time elapsed
DISABLEPLC3           ; disables PLC3 after execution
CLOSE
```

Even if the servo cycle counter rolls over (start from zero again after the counter is saturated), by subtracting into another 24-bit register, rollover is handled gracefully.

Rollover Example:

M0 = 1,000
M85 = 16,777,000 (saturates at 2^{24} = 16,777,216)
M86 = 1216

Bit	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0
M85	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	0	0	0
M86	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0

← Carryout bit

Compiled PLC Programs

It is possible to compile Turbo PMAC PLC programs for faster execution. The faster execution of the compiled PLCs comes from two factors: first, from the elimination of interpretation time, and second, from the capability of the compiled PLC programs to execute integer arithmetic. Floating-point operations in compiled PLC programs run two to three times faster than in interpreted PLC programs; integer (including Boolean) operations run 20 to 30 times faster in compiled form.

Turbo PMAC can store and execute up to 32 compiled PLC programs as well as 32 interpreted (uncompiled) PLC programs for 64 PLC programs. 15K (15,360) 24-bit words of Turbo PMAC memory are reserved for compiled PLCs or 14K (14,336) words if there is a user-written servo as well. No other task may use this memory and compiled PLCs may not use any other memory.

PLCCs are compiled by Pewin32-Pro in the downloading process. Only the compiled machine code is downloaded to PMAC. Therefore, it is suggested to save the ASCII source code in the host computer separately since it cannot be retrieved from PMAC. In most cases, compiled PLCs are firmware dependent and so they must be recompiled when the firmware is changed in PMAC.

If more than one PLCC is programmed, all the PLCCs code must belong to the same ASCII text file. Pewin will compile all the PLCC code present on the file and place it in the appropriate buffer in PMAC. If a single PLCC code is downloaded, the rest of the PLCCs that might have been present in memory will be erased, leaving only the last compiled code. The Project Manager feature of the Pewin32-Pro File menu allows PLCC codes to be in different files. They will be combined by PEWIN when the complete project is downloaded to PMAC.

The use of L-Variables in a PLC program statement tells the compiler that the statement is to be executed using integer operations instead of floating-point operations. To implement integer arithmetic in a compiled PLC, define any L-Variables and substitute them in the programs for the variables that were used in the interpreted form (usually M-Variables). The compiler will interpret statements containing only L-Variables (properly defined) and integer constants as operations to be executed using integer arithmetic in compiled PLCs.

Example: Typically machine outputs 1 and 2 are referenced by the following definitions in uncompiled programs:

```
M7000->Y:$078C00,0,1          ; Machine Output 1
M7001->Y:$078C00,1,1          ; Machine Output 2
```

For the compiled PLC programs, create equivalent M-Variable definitions:

```
L7000->Y:$078C00,0,1          ; Machine Output 1
L7001->Y:$078C00,1,1          ; Machine Output 2
```

Preparation of compiled PLCs is a multi-step process. The basic steps are as follows:

1. Write and debug the PLC programs in interpreted form (simple PLCs programs).
2. Change all references to PLCs to be compiled from PLC to PLCC.
3. For integer arithmetic, define L-Variables and substitute these for the old variable names in the programs.
4. Combine all of the PLCC programs to be compiled into one file on the PC, or use the Project Manager of Pewin32-Pro.
5. Activate the compiled PLCCs. If operation is not correct, return to step 1 or 2.
6. PLCCs can be deleted using the **DELETE PLCCn** command (replace n by the appropriate number).

TROUBLESHOOTING

Establishing Communications



Serial communications can be checked using the Windows® Hyperterminal program with 38,400 baud rate, eight data bits, one stop bit, no parity and no flow control.

1. Select the appropriate COM port and try different baud rates (bits per second) if necessary.
2. In the terminal window, type **I3** and press **Enter**. PMAC should respond with some characters.
3. In this mode, set **I3=1** to add a carriage return character at the end of each response line. If there is no response, check the serial cable or try a different COM port.
4. Follow the reset procedure in the following section if communications cannot be established.

Hardware Re-initialization

1. Carefully remove the Turbo PMAC2 3U CPU board from the UMAC rack.
2. Install the jumper labeled E3 in the Turbo PMAC2 3U CPU board.
3. Replace the Turbo PMAC2 3U CPU board inside the UMAC rack and power up.
4. If the Turbo PMAC2 3U CPU board finds jumper E3 installed on power-up, the following actions occur:
   a. The installed firmware is loaded from the flash memory into active memory.
   b. The factory default I-Variables are loaded from firmware into active memory and registers. (The last saved values in flash are not lost; they are simply not used.) The last saved user programs, tables and buffers are loaded into active memory, but none will be active because of the default I-Variable settings. If the checksum for the programs and buffers does not match the data, all of these programs and buffers are completely cleared from active memory.
   c. The basic configuration of the system – memory capacity, ASIC presence, location, and type – is checked and logged. The CPU will make some decisions about default I-Variable values based on this configuration information. Counters in all ASICs are cleared.
   d. Because of the default I-Variable configuration, no motors are enabled and no programs are activated.
5. Establish communications with either PEWIN32-Pro or Windows® Hyperterminal.
6. Type the following commands in the terminal window:

```
$$$***                      ; Global Reset
P0..8191 = 0                ; Reset P-Variables values
Q0..8191 = 0                ; Reset Q-Variables values
M0..8191 -> *               ; Reset M-Variables definitions
M0..8191 = 0                ; Reset M-Variables values
UNDEFINE ALL                ; Undefine Coordinate Systems
SAVE                        ; Save this initial clean configuration
```

7. Remove the Turbo PMAC2 3U CPU board from the UMAC rack, remove the E3 jumper, replace the Turbo PMAC2 3U CPU board in the UMAC rack and try communications again.

The Watchdog Timer (Red LED)

Turbo PMAC has an on-board watchdog timer. This subsystem provides a fail-safe shutdown to guard against software and hardware malfunction. To keep it from tripping the hardware circuit for the watchdog timer requires that two basic conditions be met. First, it must see a DC voltage greater than 4.75V. If the supply voltage is below this value, the circuit's relay will trip and the card will shut down. This prevents corruption of registers due to insufficient voltage.

Second, the timer must see a square wave input (provided by the Turbo PMAC software) of a frequency greater than 25 Hz. In the foreground, the servo-interrupt routine decrements a counter (as long as the counter is greater than zero), causing the least significant bit of the timer to toggle. This bit is fed to the timer itself. At the end of each background cycle, the CPU resets the counter value to a maximum value set by variable I40 (or to 4096 if I40 is set to the default of 0).

If the card, for whatever reason, due either to hardware or software problems, cannot set and clear this bit repeatedly at 25 Hz or greater, the timer will trip and the Turbo PMAC system will shut down. When the timer trips due either to under-voltage or under-frequency, the system is latched into a reset state with a red LED indicating watchdog failure. The processor stops operating and will not communicate. All Servo and IO ICs are forced into their reset states, which force discrete outputs off and proportional outputs (DAC, PWM, PFM) to zero level.

Once the watchdog timer has tripped, power to the UMAC System must be cycled off and on to restore normal functioning.

System Configuration

After performing a hardware re-initialization, or issuing a **$$$***** command, UMAC is configured with the hardware found in the system. The System Configuration Reporting I-Variables I4900 to I4965 provide information about the accessory boards found inside the UMAC System on power-up or reset. The UMAC Configuration program, part of the Pewin32 Pro Suite, uses these variables to report the configuration of any UMAC System. Checking the configuration of the UMAC System is important in case of addressing conflicts or hardware\software failures in the accessory boards.

UMAC System Status Bits

There are three online PMAC commands for reporting the status of the UMAC System at any time:

? Report status words for motor in hex ASCII form
?? Report coordinate system status in hex ASCII form
??? Report global status words in hex ASCII

The easiest way to read this information is through the Pewin32 Pro software. Screens for motor, coordinate systems and global status are available under the View menu. Alternatively, the most commonly used status bits can be monitored through the set of suggested M-Variables definitions.

Direct Access to Hardware Features

In PMAC, a motor is a software concept. A PMAC motor is a set of registers and variables in the PMAC memory space that is controlled by the servo algorithms inside the PMAC firmware and that uses an actual hardware circuitry. This hardware circuitry is referred to as a motion channel. A set of four hardware channels, in turn is referred to as a Servo IC. These are application-specific ICs (ASICs) designed by Delta Tau and manufactured in gate array technology to create a full-feature set in a cost-effective manner. The Servo ICs contain all of the digital logic to provide the interface between the CPU and the motion (servo or stepper) channels.

One or more channel can be associated with a single motor by means of motor-specific I-Variables. These variables define, for example, where the amplifier command will be output or which encoder input will be used for feedback. If the motor activation control variable Ixx00 is set to zero, however, the hardware channels associated with that motor xx can be directly controlled through M-Variables. This is useful for directly controlling output features like DAC or stepper outputs and amplifier enable outputs, thus allowing testing the functionality of a particular feature by direct access to the channel registers.

Example: The functionality of DAC #1 of an Acc-24E2A can be tested with the following procedure:

```
M102->Y:$078202,8,16,S ;Address of DAC #1 of Servo IC #2
I100 = 0               ;Deactivate motor #1 to allow direct access to the
channel
I7216 = 3              ;Sets output mode of channel 1 of servo IC #2 to DACs
M102 = 16383           ;DAC register is scaled as 3276.7 = 1V
<measure 5V between pins 1 and 12 of the amplifier connector of the Acc-24E2A
board>
M102 = -16383          ;Set DAC output to -5V
<measure -5V between pins 1 and 12 of the amplifier connector of the Acc-24E2A
board>
I100 = 1               ;Activate motor after the procedure is completed
```

Note:

Make sure the amplifier and any other external device using the hardware outputs is not powered or disconnected during this test. Deactivating the motor automatically disables any safety feature that applies to the hardware channels.

Motor Parameters

1. If there is no movement at all, check the following:
   a. Check the output configuration of the motor. For an analog amplifier, set I7mn6=3 (I7216=3 for the first motor of the first axes board). For a stepper driver, set Ixx02 = Ixx02+2 (I102 for motor #1).
   b. When using an analog amplifier, check the power supply lines +15V, -15V, and GND. If the UMAC internal power supply is used (default), jumpers E85, E87, and E88 in the Acc-24E2A board must be installed. The voltage can be checked in the connector at the back panel of the UMAC rack.
   c. Check the functionality of the hardware end-of-travel limits, or disable this feature by setting bit 17 of the Ixx24 variable to 1 (I124 = $20001 to disable the overtravel limits of motor #1).
   d. Make sure that the proportional gain (Ixx30) is greater than zero.
   e. Make sure that output can be measured at the DAC pin when an **O** command has been given.
   f. If the following error limit is being tripped, increase the fatal following error limit (Ixx11) by setting it to a slightly higher value and try to move again.
   g. Set the feedrate override of the addressed coordinate system to 100 by issuing a **%100** online command.
2. If there is movement, but it is sluggish, check the following:
   a. Make sure that the proportional gain (Ixx30) is not too low. Try increasing it (as long as stability is kept).
   b. Make sure that the big step limit (Ixx67) is not too low. Try increasing it to 8,000,000 – near the maximum – to eliminate any effect.

c. Make sure that the output limit (Ixx69) is not too low. Try increasing it to 32,767 (the maximum) to make sure PMAC can output adequate voltage.

d. Use an integrator. Try increasing integral gain (Ixx33) to 10,000 or more and the integration limit (Ixx63) to 8,000,000.

3. If there is a runaway condition, check the following:
   a. Make sure that there is feedback. Check that position changes can be read in both directions.
   b. Make sure that the feedback polarity matches the output polarity. Recheck the polarity match as explained above.
4. If there is brief movement, then it stops. Check the following:
   a. If the following error limit is being tripped, increase the fatal following error limit (Ixx11) by setting it to a slightly higher value and try to move again.

Motion Programs

If the program does not run at all, there are several possibilities:

1. Try to list the program. In terminal mode, type **LIST PROG 1** (or whichever program) and see if it is there. If not, try to download it to the card again.
2. Make sure that the program buffer is closed. Type **A** to check if the program is running; type **CLOSE** to close any open buffer; type **B1** (or the program #) to point to the top of the program; and type **R** to try to run it again.
3. Make sure that each motor in the coordinate system can be jogged in both directions. If not, review that motor's setup.
4. Check if any motors have been assigned to a coordinate system that is not really set up yet. Every motor in the coordinate system must have its limits conducting current, even if there is no real motor attached.

Try the following steps for any other motion program problem:

1. Type **&1%100** in the terminal window.
2. Check that only one of the motors to be used in the motion program can be jogged appropriately.
3. Type the following commands in a text editor to be downloaded to PMAC:

```
close                ; Close any buffer opened
delete gather        ; Erase unwanted gathered data
undefine all         ; Erase coordinate definitions in all coordinate systems
#1->2000X            ; Replace #1 for the motor to be used and 2000 by the
                     ; appropriate scale factor for the number of counts
                     ; per user units
OPEN PROG 1 CLEAR    ; Prepare buffer to be written
LINEAR               ; Linear interpolation
INC                  ; Incremental mode
TA500                ; Acceleration time is 500 msec
TS0                  ; No S-curve acceleration component
TM2000               ; Total move time is 500 + 2000=2500 msec
X1                   ; One unit of distance, 2000 encoder counts
CLOSE                ; Close written buffer, program one
```

4. To run it, press **CTRL+A** and then type **B1R** in the terminal window.
5. Repeat steps 2 through 4 for all the motors to be run in the actual motion program.

A good method to test motion programs is to run them at lower than one hundred percent override rate. Any value of n from 1 to 99 in the **%n** online command will run the motion programs slower, increasing the chances for success in execution. For example, in the terminal window type: **&1 %75 B1R**. If a program runs successfully at lower feedrate override values, there could be two main reasons why it fails at 100%: either there is insufficient calculation time for the programmed moves or the acceleration and\or velocity parameters involved are unsuitable for the machine.

PLC Programs

PLCs and PLCCs are the most common sources for communication or watchdog timer failures.

- Any **SEND**, **COMMAND**, or **DISPLAY** action statement should be done on an edge-triggered condition only because the PLC can cycle faster than these operations can process their information, and the communications channels can get overwhelmed if these statements get executed on consecutive scans through the PLC.

```
IF (M7000=1)                ; Input is ON
   IF (P11=0)               ; Input was not ON last time
         COMMAND"#1J+"      ; Jog motor
         P11=1              ; Set latch
   ENDIF
ELSE
   P11=0                    ; Reset latch
ENDIF
```

- PLC0 or PLCC0 are meant to be used for only a few tasks (usually a single task) that must be done at a higher frequency than the other PLC tasks. The PLC 0 will execute every real-time interrupt as long as the tasks from the previous RTI have been completed. Potentially, PLC 0 is the most dangerous task on PMAC as far as disturbing the scheduling of tasks is concerned. If it is too long, it will starve the background tasks for time. The first thing to notice is that communications and background PLC tasks will become sluggish. In the worst case, the watchdog timer will trip, shutting down the card because the housekeeping task in background did not have the time to keep it updated.
- Because all PLC programs in PMAC's memory are enabled at power-on/reset, save I5 as 0 in PMAC's memory when developing PLC programs. This allows PMAC to be reset and no PLCs running (an enabled PLC only runs if I5 is set properly) and recover more easily from a PLC programming error.
- As an example, type these commands in the terminal window. After that, open a watch window and monitor for P1 to be counting up:

```
OPEN PLC1 CLEAR             ; Prepare buffer to be written
P1=P1+1                     ; P1 continuously incrementing
CLOSE                       ; Close written buffer, PLC1
I5=2
```

Press <**CTRL+D**> and type **ENA PLC1**.

APPENDIX A — UMAC ERROR CODE SUMMARY

I6, Error Reporting Mode

I6 controls how UMAC reports errors in command lines. When I6 is set to 0 or 2, UMAC reports any error only with a **<BELL>** character. When I6 is 0, the **<BELL>** character is given for invalid commands issued both from the host and from UMAC programs (using CMD"{command}"). When I6 is 2, the **<BELL>** character is given only for invalid commands from the host; there is no response to invalid commands issued from UMAC programs. (In no mode is there a response to valid commands issued from UMAC programs.)

When I6 is set to 1 or 3, an error number message can be reported along with the **<BELL>** character. The message comes in the form of **ERRnnn<CR>**, where nnn represents the three-digit error number. If I3 is set to 1 or 3, there is a **<LF>** character in front of the message.

When I6 is set to 1, the form of the error message is **<BELL>{error message}**. This setting is the best for interfacing with host-computer driver routines. When I6 is set to 3, the form of the error message is **<BELL><CR>{error message}**. This setting is appropriate for use with the PMAC Executive Program, Pewin, in terminal mode.

Currently, the following error messages can be reported:

Error	Problem	Solution
ERR001	Command not allowed during program execution	(Halt program execution before issuing command)
ERR002	Password error	(Enter the proper password)
ERR003	Data error or unrecognized command	(Correct syntax of command)
ERR004	Illegal character: bad value (>127 ASCII) or serial parity/framing error	(Correct the character and or check for noise on the serial cable)
ERR005	Command not allowed unless buffer is open	(Open a buffer first)
ERR006	No room in buffer for command	(Allow more room for buffer – **DELETE** or **CLEAR** other buffers)
ERR007	Buffer already in use	(Close currently open buffer first)
ERR008	MACRO auxiliary communications error	(Check MACRO ring hardware and software setup)
ERR009	Program structural error (e.g. **ENDIF** without **IF**)	(Correct structure of program)
ERR010	Both overtravel limits set for a motor in the C. S.	(Correct or disable limits)
ERR011	Previous move not completed	(**Abort** it or allow it to complete)
ERR012	A motor in the coordinate system is open-loop	(Close the loop on the motor)
ERR013	A motor in the coordinate system is not activated	(Set Ix00 to 1 or remove motor from Coordinate System.)
ERR014	No motors in the coordinate system	(Define at least one motor in Coordinate System.)
ERR015	Not pointing to valid program buffer	(Use **B** command first or clear out scrambled buffers)
ERR016	Running improperly structured program (e.g. missing **ENDWHILE**)	(Correct structure of program)
ERR017	Trying to resume after **H** or **Q** with motors out of stopped position	(Use **J=** to return motor[s] to stopped position)
ERR018	Attempt to perform phase reference during move, or move during phase reference	(Finish move before phase reference, or finish phase reference before move)
ERR019	Illegal position-change command while moves stored in CCBUFFER	(Pass through section of Program requiring storage of moves in CCBUFFER, or abort)

APPENDIX B — SELECTED UMAC I-VARIABLES SUMMARY

General Global Setup		Range	Units	Default
I0	Serial Card Number	$0 to $F (0 to 15)	None	$0
I1	Serial Port Mode	0 to 3	None	0
I3	I/O Handshake Control	0 to 3	None	1
I4	Communications Integrity Mode	0 to 3	None	1
I5	PLC Program Control	0 to 3	None	1
I6	Error Reporting Mode	0 to 3	None	1
I7	Phase Cycle Extension	0 to 15	Phase Clock Cycles	0
I8	Real-Time Interrupt Period	0 to 255	Servo Clock Cycles	2
I9	Full/Abbreviated Listing Control	0 to 3	None	2
I10	Servo Interrupt Time	"0 to 8,388,607"	"1 / 8,388,608 msec"	"3,713,707 (442 msec)"
I11	Programmed Move Calculation Time	"0 to 8,388,607"	msec	0
I13	Foreground In-Position Check Enable	0 to 1	None	0
I14	Temporary Buffer Save Enable	0 to 1	None	0
I15	Degree/Radian Control for User Trig Functions	0 to 1	None	0 (degrees)
I19	Clock Source I-Variable Number	7207 to 7957	I- Variable number	Configuration-dependent
I39	UBUS Accessory ID Variable Display Control	0 to 5	None	0
I40	Watchdog Timer Reset Value	"0 to 65,535"	Servo cycles	0 (sets 4095)
I41	I-Variable Lockout Control	$0 – $F (0 – 15)	None	0
I42	Spline/PVT Time Control Mode	0 to 1	None	0
I43	Auxiliary Serial Port Parser Disable	0 to 1	None	0
I46	P and Q- Variable Storage Location	0 to 3	None	0
I51	Compensation Table Enable	0 to 1	None	0 (disabled)
I52	CPU Frequency Control	0 to 15	Multiplication factor	7 (80 MHz)
I53	Auxiliary Serial Port Baud Rate Control	0 to 15	None	0 (disabled)
I54	Serial Port Baud Rate Control	0 to 15	None	12 (38400 baud)
I59	Motor/C.S. Group Select	0 to 3	None	0
I60	Filtered Velocity Sample Time	0 to 15	Servo Cycles - 1	15
I61	Filtered Velocity Shift	0 to 255	Bits	8
I62	Internal Message Carriage Return Control	0 to 1	None	1
I63	Control-X Echo Enable	0 to 1	None	1
I64	Internal Response Tag Enable	0 to 1	None	0
I68	Coordinate System Activation Control	0 to 15	None	15

Motor Definition I-Variables (xx: motor # from 1 to 32)		**Range**	**Units**	**Default**
Ixx00	Motor xx Activation Control	0 to 1	None	I100 = 1, I200 .. I3200 = 0
Ixx01	Motor xx Commutation Enable	0 to 3	None	0
Ixx02	Motor xx Command Output Address	$000000 to $FFFFFF	Turbo PMAC Addresses	See software reference
Ixx03	Motor xx Position Loop Feedback Address	$000000 to $FFFFFF	Turbo PMAC Addresses	See software reference
Ixx04	Motor xx Velocity Loop Feedback Address	$000000 to $FFFFFF	Turbo PMAC Addresses	See software reference
Ixx05	Motor xx Master Position Address	$000000 to $FFFFFF	Turbo PMAC 'X' Addresses	$0035C0 (end of table)
Ixx06	Motor xx Position Following Enable and Mode	0 to 3	None	0
Ixx07	Motor xx Master (Handwheel) Scale Factor	-8,388,608 to 8,388,607	None	96
Ixx08	Motor xx Position Scale Factor	0 to 8,388,607	None	96
Ixx09	Motor xx Velocity-Loop Scale Factor	0 to 8,388,607	None	96
Ixx10	Motor xx Power-On Servo Position Address	$000000 to $FFFFFF	Turbo PMAC Addresses	$0

Motor Safety I-Variables xx: motor # from 1 to 32)		**Range**	**Units**	**Default**
Ixx11	Motor xx Fatal Following Error Limit	0 to 8,388,607	1/16 count	32,000 (2000 counts)
Ixx12	Motor xx Warning Following Error Limit	0 to 8,388,607	1/16 count	16,000 (1000 counts)
Ixx13	Motor xx Positive Software Position Limit	-235 to +235	Counts	0 (disabled)
Ixx14	Motor xx Negative Software Position Limit	-235 to +235	Counts	0 (disabled)
Ixx15	Motor xx Abort/Limit Deceleration Rate	Positive Floating-Point	Counts / msec2	0.25
Ixx16	Motor xx Maximum Program Velocity	Positive Floating-Point	Counts / msec	32.0
Ixx17	Motor xx Maximum Program Acceleration	Positive Floating-Point	Counts / msec2	0.5
Ixx19	Motor xx Maximum Jog/Home Acceleration	Positive Floating-Point	Counts / msec2	0.15625

Motor Motion I-Variables (xx: motor # from 1 to 32)		**Range**	**Units**	**Default**
Ixx20	Motor xx Jog/Home Acceleration Time	0 to 8,388,607	msec	0 (so Ixx21 controls)
Ixx21	Motor xx Jog/Home S-Curve Time	0 to 8,388,607	msec	50
Ixx22	Motor xx Jog Speed	Positive Floating Point	Counts / msec	32.0
Ixx23	Motor xx Home Speed and Direction	Floating Point	Counts / msec	32.0
Ixx24	Motor xx Flag Mode Control	$000000 to $FFFFFF	None	$000001
Ixx25	Motor xx Flag Address	$000000 to $FFFFFF	Turbo PMAC Addresses	See software reference
Ixx26	Motor xx Home Offset	-8,388,608 to 8,388,607"=	1/16 count	0
Ixx27	Motor xx Position Rollover Range	-235 to +235	Counts	0
Ixx28	Motor xx In-Position Band	0 to 8,388,607	1/16 count	160 (10 counts)
Ixx29	Motor xx Output/First Phase Offset	-32,768 to 32,767	16-bit equivalent	0

Motor xx PID Servo Setup (xx: motor # from 1 to 32)		**Range**	**Units**	**Default**
Ixx30	Motor xx PID Proportional Gain	-8,388,608 to 8,388,607	See software reference	2000
Ixx31	Motor xx PID Derivative Gain	-8,388,608 to 8,388,607	See software reference	1280
Ixx32	Motor xx PID Velocity Feedforward Gain	-8,388,608 to 8,388,607	See software reference	1280
Ixx33	Motor xx PID Integral Gain	0 to 8,388,607	See software reference	1280
Ixx34	Motor xx PID Integration Mode	0 to 1	None	1
Ixx35	Motor xx PID Acceleration Feedforward Gain	-8,388,608 to 8,388,607	See software reference	0
Ixx40	Motor xx Net Desired Position Filter Gain	0.0 to 0.999999	None	0.0
Ixx41	Motor xx Desired Position Limit Band	0 to 8,388,607	Counts	0

Motor Servo Setup (xx: motor # from 1 to 32)		**Range**	**Units**	**Default**
Ixx57	Motor xx Continuous Current Limit	-32,768 to 32,767	16-bit equivalent	0
Ixx58	Motor xx Integrated Current Limit	0 to 8,388,607	See software reference	0
Ixx59	Motor xx User-Written Servo/Phase Enable	0 to 3	None	0
Ixx60	Motor xx Servo Cycle Period Extension Period	0 to 255	Servo Interrupt Periods	0
Ixx63	Motor xx Integration Limit	-8,388,608 to 8,388,607	1/16 count * servo cycle	4,194,304
Ixx64	Motor xx Deadband Gain Factor	-32,768 to 32,767	None	0 (no gain adjustment)
Ixx65	Motor xx Deadband Size	-32,768 to 32,767	1/16 count	0
Ixx67	Motor xx Position Error Limit	0 to 8,388,607	1/16 count	4,194,304 (262,144 counts)
Ixx68	Motor xx Friction Feedforward	0 to 32,767	16-bit DAC bits	0
Ixx69	Motor xx Output Command Limit	0 to 32,767	16-bit DAC bits	20,480 (6.25V or equivalent)

Further Motor I-Variables (xx: motor # from 1 to 32)		**Range**	**Units**	**Default**
Ixx85	Motor xx Backlash Take-up Rate	0 to 8,388,607	1/16 count / background cycle	0
Ixx86	Motor xx Backlash Size	0 to 8,388,607	1/16 count	0
Ixx87	Motor xx Backlash Hysteresis	0 to 8,388,607	1/16 count	64 (= 4 counts)
Ixx88	Motor xx In-Position Number of Scans	0 to 255	Background computation cycles (minus one)	0
Ixx90	Motor xx Rapid Mode Speed Select	0 to 1	None	1
Ixx91	Motor xx Power-On Phase Position Format	$000000 to $FFFFFF	None	0
Ixx92	Motor xx Jog Move Calculation Time	1 to 8,388,607	msec	10
Ixx95	Motor xx Power-On Servo Position Format	$000000 to $FFFFFF	None	$000000
Ixx96	Motor xx Command Output Mode Control	0 to 1	None	0
Ixx97	Motor xx Position Capture & Trigger Mode	0 to 3	None	0

System Configuration Reporting		**Range**	**Units**	**Default**
I4900	Servo ICs Present	$000000 to $FFFFFF	None (individual bits)	--
I4901	Servo IC Type	$000000 to $FFFFFF	None (individual bits)	--
I4904	Dual-Ported RAM ICs Present	$000000 to $FF8000	None (individual bits)	--
I4908	End of Open Memory	$006000 to $040000	None (individual bits)	--
I4909	Turbo CPU ID Configuration	$000000000 to $FFFFFFFFF	None (individual bits)	--
I4910 to I4925	Servo IC Card Identification	$000000000 to $FFFFFFFFF	None (individual bits)	--
I4942 to I4949	DPRAM IC Card Identification	$000000000 to $FFFFFFFFF	None (individual bits)	--
I4950 to I4965	I/O IC Card Identification	$000000000 to $FFFFFFFFF	None (individual bits)	--
I5060	A/D Processing Ring Size	0 to 16	Number of A/D Pairs	0
I5061 to I5076	A/D Ring Slot Pointers	$000000 to $7FFFFF	Turbo PMAC Addresses	$0 (specifies $078800)
I5080	A/D Ring Convert Enable	0 to 1	None	1
I5081 to I5096	A/D Ring Convert Codes	$000000 to $00F00F	None	$000000

Coordinate System I-Variables (sx: CS # + 50)		**Range**	**Units**	**Default**
Isx11	Coordinate System 'x' User Countdown Timer 1	-8,388,608 to 8,388,607	Servo cycles	0
Isx12	Coordinate System 'x' User Countdown Timer 2	-8,388,608 to 8,388,607	Servo cycles	0
Isx13	Coordinate System 'x' Segmentation Time	0 to 255	msec	0
Isx20	Coordinate System 'x' Lookahead Length	0 to 65,535	Isx13 segmentation periods	0
Isx21	Coordinate System 'x' Lookahead State Control	0 to 15	None	0
Isx50	Coordinate System 'x' Kinematic Calculations Enable	0 to 1	None	0
Isx53	Coordinate System 'x' Step Mode Control	0 to 1	None	0
Isx86	Coordinate System 'x' Alternate Feedrate	Positive floating point	See software reference	1000.0
Isx87	Coordinate System 'x' Default Program Acceleration Time	0 to 8,388,607	msec	0 (so Isx88 controls)
Isx88	Coordinate System 'x' Default Program S-Curve Time	0 to 8,388,607	msec	50
Isx89	Coordinate System 'x' Default Program Feedrate/Move Time	Positive floating point	See software reference	1000.0
Isx90	Coordinate System 'x' Feedrate Time Units	Positive floating point	msec	1000.0
Isx91	Coordinate System 'x' Default Working Program Number	0 to 32,767	Motion Program Numbers	0
Isx92	Coordinate System 'x' Move Blend Disable	0 to 1	None	0
Isx93	Coordinate System 'x' Time Base Control Address	$000000 to $FFFFFF	Turbo PMAC X-Addresses	See software reference
Isx94	Coordinate System 'x' Time Base Slew Rate	0 to 8,388,607	2-23msec / servo cycle	1644
Isx95	Coordinate System 'x' Feed Hold Slew Rate	0 to 8,388,607	2-23msec / servo cycle	1644
Isx96	Coordinate System 'x' Circle Error Limit	Positive floating-point	User length units	0 (function disabled)
Isx97	Coordinate System 'x' Minimum Arc Length	Non-negative floating-point	Semi-circles (180°)	0 (sets 2-20)
Isx98	Coordinate System 'x' Maximum Feedrate	Non-negative floating-point	See software reference	1000.0
Isx99	Coordinate System 'x' Cutter-Comp Outside Corner Break Point	-1.0 to 0.9999	cosine	0.998 (cos 1o)

Multi-Channel Servo IC (m: IC # from 2 to 9)		**Range**	**Units**	**Default**
I7m00	Servo IC m MaxPhase/PWM Frequency Control	0 to 32,767	See software reference	6527
I7m01	Servo IC m Phase Clock Frequency Control	0 to 15	See software reference	0
I7m02	Servo IC m Servo Clock Frequency Control	0 to 15	See software reference	3
I7m03	Servo IC m Hardware Clock Control	0 to 4095	See software reference	2258
I7m04	Servo IC m PWM Deadtime / PFM Pulse Width Control	0 to 255	See software reference	15

Channel-Specific Servo IC (m: IC # 2-9, n: ch # 1-4)		**Range**	**Units**	**Default**
I7mn0	Servo IC m Channel n Encoder/Timer Decode Control	0 to 15	None	7
I7mn1	Servo IC m Channel n Position Compare Channel Select	0 to 1	None	0
I7mn2	Servo IC m Channel n Capture Control	0 to 15	None	1
I7mn3	Servo IC m Channel n Capture Flag Select Control	0 to 3	None	0
I7mn4	Servo IC m Channel n Encoder Gated Index Select	0 to 1	None	0
I7mn6	Servo IC m Channel n Output Mode Select	0 to 3	None	0
I7mn7	Servo IC m Channel n Output Invert Control	0 to 3	None	0
I7mn8	Servo IC m Channel n PFM Direction Signal Invert Control	0 to 1	None	0

Conversion Table I-Variables		**Range**	**Units**	**Default**
I8000 to I8191	Conversion Table Setup Lines	$000000 - $FFFFFF	Turbo PMAC Addresses	See software reference

APPENDIX C — SELECTED UMAC ONLINE COMMANDS

Command	Description
<CONTROL-A>	Abort all programs and moves.
<CONTROL-B>	Report status word for eight motors.
<CONTROL-C>	Report all coordinate system status words.
<CONTROL-D>	Disable all PLC programs.
<CONTROL-F>	Report following errors for eight motors.
<CONTROL-G>	Report global status word.
<CONTROL-H>	Erase last character.
<CONTROL-I>	Repeat last command line.
<CONTROL-K>	Kill all motors.
<CONTROL-M>	Enter command line.
<CONTROL-N>	Report command line checksum.
<CONTROL-O>	Feed hold on all coordinate systems.
<CONTROL-P>	Report positions for eight motors.
<CONTROL-Q>	Quit all executing motion programs.
<CONTROL-R>	Begin execution of motion programs in all coordinate systems.
<CONTROL-S>	Step working motion programs in all coordinate systems.
<CONTROL-T>	Cancel MACRO pass-through mode.
<CONTROL-V>	Report velocity for eight motors.
<CONTROL-X>	Cancel in-process communications.
!{axis}{constant}[{axis}{constant}…]	Alter destination of RAPID move.
@	Report currently addressed card on serial daisychain.
@{card}	Address a card on the serial daisychain.
#	Report port's currently addressed motor.
#{constant}	Select port's addressed motor.
#{constant}->	Report the specified motor's coordinate system axis definition.
#{constant}->0	Clear axis definition for specified motor.
#{constant}->{axis definition}	Assign an axis definition for the specified motor.
#{constant}->I	Assign inverse-kinematic definition for specified motor.
##	Report port's motor group.
##{constant}	Select port's motor group.
$	Establish phase reference for motor.
$$	Establish phase reference for motors in coordinate system.
$$$	Full card reset.
$$$***	Global card reset and re-initialization.
$$*	Read motor absolute positions.
$*	Read motor absolute position.
%	Report the addressed coordinate system's feedrate override value.
%{constant}	Set the addressed coordinate system's feedrate override value.
&	Report port's currently addressed coordinate system.
&{constant}	Select port's addressed coordinate system.
\	Quick Stop in Lookahead/Feed Hold.
<	Back up through Lookahead Buffer.
>	Resume Forward Execution in Lookahead Buffer.
/	Halt Motion at End of Block.
?	Report motor status.
??	Report the status words of the addressed coordinate system.
???	Report global status words.
A	Abort all programs and moves in the currently addressed coordinate system.

Command	Description
ABR[{constant}]	Abort currently running motion program and start another.
ABS	Select absolute position mode for axes in addressed coordinate system.
{axis}={constant}	Re-define the specified axis position.
B{constant}	Point the addressed coordinate system to a motion program.
CHECKSUM	Report the firmware checksum value.
CID	Report card ID or part number.
CLEAR	Erase currently opened buffer.
CLEAR ALL	Erase all fixed motion, kinematic, and uncompiled PLC programs.
CLEAR ALL PLCS	Erase all uncompiled PLC programs.
CLOSE	Close the currently opened buffer.
CLOSE ALL	Close the currently opened buffer on any port.
{constant}	Assign value to variable P0, or to table entry.
CPU	Report the Turbo PMAC CPU type.
DATE	Report the firmware release date.
DEFINE BLCOMP	Define backlash compensation table.
DEFINE CCBUF	Define extended cutter-compensation buffer.
DEFINE COMP (one-dimensional)	Define Leadscrew Compensation Table.
DEFINE COMP (two-dimensional)	Define two-dimensional leadscrew compensation table.
DEFINE GATHER	Create a data gathering buffer.
DEFINE LOOKAHEAD	Create a lookahead buffer.
DEFINE ROTARY	Define a rotary motion program buffer.
DEFINE TBUF	Create a buffer for axis transformation matrices.
DEFINE TCOMP	Define torque compensation table.
DEFINE UBUFFER	Create a buffer for user variable use.
DELETE ALL	Erase all defined permanent and temporary buffers.
DELETE ALL TEMPS	Erase all defined temporary buffers.
DELETE BLCOMP	Erase backlash compensation table.
DELETE CCUBUF	Erase extended cutter-compensation buffer.
DELETE COMP	Erase leadscrew compensation table.
DELETE LOOKAHEAD	Erase the lookahead buffer.
DELETE GATHER	Erase the data gather buffer.
DELETE PLCC	Erase specified compiled PLC program.
DELETE ROTARY	Delete rotary motion program buffer of addressed coordinate system.
DELETE TBUF	Delete buffer for axis transformation matrices.
DELETE TCOMP	Erase torque compensation table.
DISABLE PLC	Disable specified PLC programs.
DISABLE PLCC	Disable compiled PLC programs.
EAVERSION	Report firmware version information.
ENABLE PLC	Enable specified PLC programs.
ENABLE PLCC	Enable specified compiled PLC programs.
ENDGATHER	Stop data gathering.
F	Report motor following error.
FRAX	Specify the coordinate system's feedrate axes.
GATHER	Begin data gathering.
H	Perform a feed hold.
HOME	Start Homing Search Move.
HOMEZ	Do a Zero-Move Homing.

Command	Description
I{constant}	Report the current I-Variable values.
I{data}={expression}	Assign a value to an I-Variable.
I{constant}=*	Assign factory default value to an I-Variable.
I{constant}=@I{constant}	Set I-Variable to address of another I-Variable.
IDC	Force active clock equal to ID-module clock.
IDNUMBER	Report electronic identification number.
INC	Specify Incremental Move Mode.
J!	Adjust motor commanded position to nearest integer count.
J+	Jog Positive.
J-	Jog Negative.
J/	Jog Stop.
J:{constant}	Jog Relative to Commanded Position.
J:*	Jog to specified variable distance from present commanded position.
J=	Jog to Prejog Position.
J={constant}	Jog to specified position.
J=*	Jog to specified variable position.
J=={constant}	Jog to specified motor position and make that position the pre-jog position.
J^{constant}	Jog Relative to Actual Position.
J^*	Jog to specified variable distance from present actual position.
{jog command}^{constant}	Jog until trigger.
K	Kill motor output.
LEARN	Learn present commanded position.
LIST	List the contents of the currently opened buffer.
LIST BLCOMP	List contents of addressed motor's backlash compensation table.
LIST BLCOMP DEF	List definition of addressed motor's backlash compensation table.
LIST COMP	List contents of addressed motor's compensation table.
LIST COMP DEF	List definition of addressed motor's compensation table.
LIST FORWARD	Report contents of forward-kinematic program buffer.
LIST GATHER	Report contents of the data gathering buffer.
LIST INVERSE	Report contents of inverse-kinematic program buffer.
LIST LDS	List Linking Addresses of Ladder Functions.
LIST LINK	List Linking Addresses of Internal Turbo PMAC Routines.
LIST PC	List Program at Program Counter.
LIST PE	List Program at Program Execution.
LIST PLC	List the contents of the specified PLC program.
LIST PROGRAM	List the contents of the specified motion program.
LIST ROTARY	List contents of addressed coordinate system's rotary program buffer.
LIST TCOMP	List contents of addressed motor's torque compensation table.
LIST TCOMP DEF	List definition of addressed motor's torque compensation table.
LOCK{constant},P{constant}	Check/set process locking bit.
M{constant}	Report the current M- Variable values.
M{data}={expression}	Assign value to M- Variable s.
M{constant}->	Report current M- Variable definitions.
M{constant}->*	Self-Referenced M-Variable Definition.
M{constant}->D:{address}	Long Fixed-Point M-Variable Definition.
M{constant}->DP:{address}	Dual-Ported RAM Fixed-Point M-Variable Definition.
M{constant}->F:{address}	Dual-Ported RAM Floating-Point M-Variable Definition.
M{constant}->L:{address}	Long Word Floating-Point M-Variable Definition.
M{constant}->TWB:{address}	Binary Thumbwheel-Multiplexer Definition.
M{constant}->TWD:{address}	BCD Thumbwheel-Multiplexer M-Variable Definition.

Command	Description
M{constant}->TWR:{address}	Resolver Thumbwheel-Multiplexer M-Variable Definition.
M{constant}->TWS:{address}	Serial Thumbwheel-Multiplexer M-Variable Definition.
M{constant}->X/Y:{address}	Short Word M-Variable Definition.
MFLUSH	Clear pending synchronous M-variable assignments.
MOVETIME	Report time left in presently executing move.
NOFRAX	Remove all axes from list of vector feedrate axes.
NORMAL	Report circle-plane unit normal vector.
O{constant}	Open loop output.
OPEN BINARY ROTARY	Open all existing rotary buffers for binary DPRAM entry.
OPEN FORWARD	Open a forward-kinematic program buffer for entry.
OPEN INVERSE	Open an inverse-kinematic program buffer for entry.
OPEN PLC	Open a PLC program buffer for entry.
OPEN PROGRAM	Open a fixed motion program buffer for entry.
OPEN ROTARY	Open all existing rotary motion program buffers for text entry.
P	Report motor position.
P{constant}	Report the current P-Variable values.
P{data}={expression}	Assign a value to a P-Variable.
PASSWORD={string}	Enter/set program password.
PAUSE PLC	Pause specified PLC programs.
PC	Report program counter.
PE	Report program execution pointer.
PMATCH	Re-match axis positions to motor positions.
PR	Report rotary program remaining.
Q	Quit program at end of move.
Q{constant}	Report Q-Variable value.
Q{data}={expression}	Q-Variable value assignment
R	Run motion program
R[H]{address}	Report the contents of specified memory addresses.
RESUME PLC	Resume execution of specified PLC programs.
S	Execute one move (step) of motion program.
SAVE	Copy setup parameters to non-volatile memory.
SETPHASE	Set commutation phase position value.
SID	Report serial electronic identification number.
SIZE	Report the amount of unused buffer memory in Turbo PMAC.
STN	Report MACRO station order number.
STN={constant}	Set MACRO station order number.
TIME	Report present time.
TIME={time}	Set the present time.
TODAY	Report present date.
TODAY={date}	Set the present date.
TYPE	Report type of Turbo PMAC.
UNDEFINE	Erase coordinate system definition.
UNDEFINE ALL	Erase coordinate definitions in all coordinate systems.
UNLOCK{constant}	Clear process locking bit.
UPDATE	Copy present date and time to non-volatile storage.
V	Report motor velocity.
VERSION	Report PROM firmware version number.
VID	Report vendor identification number.
W{address}	Write values to specified addresses.
Z	Coordinate-system specific.

APPENDIX D — SELECTED UMAC MOTION PROGRAM COMMANDS

Command	Description
{axis}{data}[{axis}{data}...]	Position-only move specification
{axis}{data}:{data} [{axis}{data}:{data}…]	Position and velocity move specification
{axis}{data}[{axis}{data}…] {vector}{data} [{vector}{data}…]	Circular arc move specification
A{data}	A-Axis move
ABS	Absolute move mode
B{data}	B-Axis move
C{data}	C-Axis move
CALL	Jump to subprogram with return
CIRCLE1	Set blended clockwise circular move mode
CIRCLE2	Set blended counterclockwise circular move mode
"COMMANDx""{command}"""	Command issuance from internal program
COMMANDx^{letter}	Control-character command issuance from internal program
DELAY{data}	Delay for specified time
"DISABLE PLC {constant}[,{constant}...]"	Disable PLC programs
"DISABLE PLCC {constant}[,{constant}...]"	Disable compiled PLC programs
"DISPLAY [{constant}] ""{message}"""	Display text to display port
DISPLAY ... {variable}	Formatted display of variable value
DWELL	Dwell for specified time
ELSE	Start false condition branch
ENABLE PLC	Enable PLC buffers
ENABLE PLCC	Enable compiled PLC programs
ENDIF	Mark end of conditional block
ENDWHILE	Mark end of conditional loop
F{data}	Set Move Feedrate (Velocity)
FRAX	Specify feedrate axes
GOSUB	Unconditional jump with return
GOTO	Unconditional jump without return
HOME	Programmed homing
HOMEZ	Programmed zero-move homing
I{data}	I-Vector specification for circular moves or normal vectors
I{data}={expression}	Set I-Variable value
IF ({condition})	Conditional branch
INC	Incremental move mode
J{data}	J-Vector specification for circular moves
K{data}	K-Vector specification for circular moves
LINEAR	Blended linear interpolation move mode
M{data}={expression}	Set M-Variable value
M{data}=={expression}	Synchronous M-Variable value assignment
N{constant}	Program line label

Command	Description
OR({condition})	Conditional **OR**
P{data}={expression}	Set P-Variable value
PSET	Redefine current axis positions (position SET)
Q{data}={expression}	Set Q-Variable value
R{data}	Set circle radius
RAPID	Set rapid traverse mode
READ	Read arguments for subroutine
RETURN	Return from subroutine jump/end main program
SENDx	Cause Turbo PMAC to send message
SENDx^{letter}	Cause Turbo PMAC to send control character
STOP	Stop program execution
TA{data}	Set acceleration time
TM{data}	Set move time
TS{data}	Set S-Curve acceleration time
U{data}	U-Axis move
V{data}	V-Axis move
W{data}	W-Axis move
WAIT	Suspend program execution
WHILE({condition})	Conditional looping
X{data}	X-Axis move
Y{data}	Y-Axis move
Z{data}	Z-Axis move

APPENDIX E — SELECTED UMAC PLC PROGRAM COMMANDS

Command	Description
ADDRESS	Motor/Coordinate System Modal Addressing
ADDRESS#P{constant}	Select program's addressed motor
ADDRESS&P{constant}	Select program's addressed coordinate system
AND ({condition})	Conditional **AND**
"COMMANDx""{command}"""	Command issuance from internal program
COMMANDx^{letter}	Control-Character command issuance
"DISABLE PLC {constant}[,{constant}...]"	Disable PLC programs
"DISABLE PLCC {constant}[,{constant}...]"	Disable compiled PLC programs
"DISPLAY [{constant}] ""{message}"""	Display text to display port
DISPLAY ... {variable}	Formatted display of variable value
ELSE	Start false condition branch
ENABLE PLC	Enable PLC buffers
ENABLE PLCC	Enable compiled PLC programs
ENDIF	Mark end of conditional block
ENDWHILE	Mark end of conditional loop
I{data}={expression}	Set I-Variable value
IF ({condition})	Conditional branch
M{data}={expression}	Set M-Variable value
OR({condition})	Conditional **OR**
P{data}={expression}	Set P-Variable value
PAUSE PLC	Pause execution of PLC programs
Q{data}={expression}	Set Q-Variable value
RESUME PLC	Resume execution of PLC programss
SENDx	Cause Turbo PMAC to send message
SENDx^{letter}	Cause Turbo PMAC to send control character
WHILE({condition})	Conditional looping

APPENDIX F — MOTOR SUGGESTED M-VARIABLE DEFINITIONS

Hardware Channel Registers	Channel #1	Channel #2	Channel #3	Channel #4
ENC 24-bit counter position	"M101->X:$078201,0,24,S"	"M201->X:$078209,0,24,S"	"M301->X:$078211,0,24,S"	"M401->X:$078219,0,24,S"
OUTA command value DAC or PWM	"M102->Y:$078202,8,16,S"	"M202->Y:$07820A,8,16,S"	"M302->Y:$078212,8,16,S"	"M402->Y:$07821A,8,16,S"
OUTC command value PFM or PWM	"M107->Y:$078204,8,16,S"	"M207->Y:$07820C,8,16,S"	"M307->Y:$078214,8,16,S"	"M407->Y:$07821C,8,16,S"
AENA output status	"M114->X:$078205,14"	"M214->X:$07820D,14"	"M314->X:$078215,14"	"M414->X:$07821D,14"
USER flag input status	"M115->X:$078200,19"	"M215->X:$078208,19"	"M315->X:$078210,19"	"M415->X:$078218,19"
ENC count error flag	"M118->X:$078200,8"	"M218->X:$078208,8"	"M318->X:$078210,8"	"M418->X:$078218,8"
CHC input status	"M119->X:$078200,14"	"M219->X:$078208,14"	"M319->X:$078210,14"	"M419->X:$078218,14"
HMFL flag input status	"M120->X:$078200,16"	"M220->X:$078208,16"	"M320->X:$078210,16"	"M420->X:$078218,16"
PLIM flag input status	"M121->X:$078200,17"	"M221->X:$078208,17"	"M321->X:$078210,17"	"M421->X:$078218,17"
MLIM flag input status	"M122->X:$078200,18"	"M222->X:$078208,18"	"M322->X:$078210,18"	"M422->X:$078218,18"
FAULT flag input status	"M123->X:$078200,15"	"M223->X:$078208,15"	"M323->X:$078210,15"	"M423->X:$078218,15"

Hardware Channel Registers	Channel #5	Channel #6	Channel #7	Channel #8
ENC 24-bit counter position	"M501->X:$078301,0,24,S"	"M601->X:$078309,0,24,S"	"M701->X:$078311,0,24,S"	"M801->X:$078319,0,24,S"
OUTA command value DAC or PWM	"M502->Y:$078302,8,16,S"	"M602->Y:$07830A,8,16,S"	"M702->Y:$078312,8,16,S"	"M802->Y:$07831A,8,16,S"
OUTC command value PFM or PWM	"M507->Y:$078304,8,16,S"	"M607->Y:$07830C,8,16,S"	"M707->Y:$078314,8,16,S"	"M807->Y:$07831C,8,16,S"
AENA output status	"M514->X:$078305,14"	"M614->X:$07830D,14"	"M714->X:$078315,14"	"M814->X:$07831D,14"
USER flag input status	"M515->X:$078300,19"	"M615->X:$078308,19"	"M715->X:$078310,19"	"M815->X:$078318,19"
ENC count error flag	"M518->X:$078300,8"	"M618->X:$078308,8"	"M718->X:$078310,8"	"M818->X:$078318,8"
CHC input status	"M519->X:$078300,14"	"M619->X:$078308,14"	"M719->X:$078310,14"	"M819->X:$078318,14"
HMFL flag input status	"M520->X:$078300,16"	"M620->X:$078308,16"	"M720->X:$078310,16"	"M820->X:$078318,16"
PLIM flag input status	"M521->X:$078300,17"	"M621->X:$078308,17"	"M721->X:$078310,17"	"M821->X:$078318,17"
MLIM flag input status	"M522->X:$078300,18"	"M622->X:$078308,18"	"M722->X:$078310,18"	"M822->X:$078318,18"
FAULT flag input status	"M523->X:$078300,15"	"M623->X:$078308,15"	"M723->X:$078310,15"	"M823->X:$078318,15"

Motor Status Bits	**Motor #1**	**Motor #2**	**Motor #3**	**Motor #4**
Stopped-on-position-limit bit	"M130->Y:$0000C0,11,1"	"M230->Y:$000140,11,1"	"M330->Y:$0001C0,11,1"	"M430->Y:$000240,11,1"
Positive-end-limit-set bit	"M131->X:$0000B0,21,1"	"M231->X:$000130,21,1"	"M331->X:$0001B0,21,1"	"M431->X:$000230,21,1"
Negative-end-limit-set bit	"M132->X:$0000B0,22,1"	"M232->X:$000130,22,1"	"M332->X:$0001B0,22,1"	"M432->X:$000230,22,1"
Desired-velocity-zero bit	"M133->X:$0000B0,13,1"	"M233->X:$000130,13,1"	"M333->X:$0001B0,13,1"	"M433->X:$000230,13,1"
Dwell-in-progress bit	"M135->X:$0000B0,15,1"	"M235->X:$000130,15,1"	"M335->X:$0001B0,15,1"	"M435->X:$000230,15,1"
Running-program bit	"M137->X:$0000B0,17,1"	"M237->X:$000130,17,1"	"M337->X:$0001B0,17,1"	"M437->X:$000230,17,1"
Open-loop-mode bit	"M138->X:$0000B0,18,1"	"M238->X:$000130,18,1"	"M338->X:$0001B0,18,1"	"M438->X:$000230,18,1"
Amplifier-enabled status bit	"M139->X:$0000B0,19,1"	"M239->X:$000130,19,1"	"M339->X:$0001B0,19,1"	"M439->X:$000230,19,1"
In-position bit	"M140->Y:$0000C0,0,1"	"M240->Y:$000140,0,1"	"M340->Y:$0001C0,0,1"	"M440->Y:$000240,0,1"
Warning-following error bit	"M141->Y:$0000C0,1,1"	"M241->Y:$000140,1,1"	"M341->Y:$0001C0,1,1"	"M441->Y:$000240,1,1"
Fatal-following-error bit	"M142->Y:$0000C0,2,1"	"M242->Y:$000140,2,1"	"M342->Y:$0001C0,2,1"	"M442->Y:$000240,2,1"
Amplifier-fault-error bit	"M143->Y:$0000C0,3,1"	"M243->Y:$000140,3,1"	"M343->Y:$0001C0,3,1"	"M443->Y:$000240,3,1"
Home-complete bit	"M145->Y:$0000C0,10,1"	"M245->Y:$000140,10,1"	"M345->Y:$0001C0,10,1"	"M445->Y:$000240,10,1"

Motor Status Bits	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
Stopped-on-position-limit bit	"M530->Y:$0002C0,11,1"	"M630->Y:$000340,11,1"	"M730->Y:$0003C0,11,1"	"M830->Y:$000440,11,1"
Positive-end-limit-set bit	"M531->X:$0002B0,21,1"	"M631->X:$000330,21,1"	"M731->X:$0003B0,21,1"	"M831->X:$000430
Negative-end-limit-set bit	"M532->X:$0002B0,22,1"	"M632->X:$000330,22,1"	"M732->X:$0003B0,22,1"	"M832->X:$000430,22,1",21,1"
Desired-velocity-zero bit	"M533->X:$0002B0,13,1"	"M633->X:$000330,13,1"	"M733->X:$0003B0,13,1"	"M833->X:$000430,13,1"
Dwell-in-progress bit	"M535->X:$0002B0,15,1"	"M635->X:$000330,15,1"	"M735->X:$0003B0,15,1"	"M835->X:$000430,15,1"
Running-program bit	"M537->X:$0002B0,17,1"	"M637->X:$000330,17,1"	"M737->X:$0003B0,17,1"	"M837->X:$000430,17,1"
Open-loop-mode bit	"M538->X:$0002B0,18,1"	"M638->X:$000330,18,1"	"M738->X:$0003B0,18,1"	"M838->X:$000430,18,1"
Amplifier-enabled status bit	"M539->X:$0002B0,19,1"	"M639->X:$000330,19,1"	"M739->X:$0003B0,19,1"	"M839->X:$000430,19,1"
In-position bit	"M540->Y:$0002C0,0,1"	"M640->Y:$000340,0,1"	"M740->Y:$0003C0,0,1"	"M840->Y:$000440,0,1"
Warning-following error bit	"M541->Y:$0002C0,1,1"	"M641->Y:$000340,1,1"	"M741->Y:$0003C0,1,1"	"M841->Y:$000440,1,1"
Fatal-following-error bit	"M542->Y:$0002C0,2,1"	"M642->Y:$000340,2,1"	"M742->Y:$0003C0,2,1"	"M842->Y:$000440,2,1"
Amplifier-fault-error bit	"M543->Y:$0002C0,3,1"	"M643->Y:$000340,3,1"	"M743->Y:$0003C0,3,1"	"M843->Y:$000440,3,1"
Home-complete bit	"M545->Y:$0002C0,10,1"	"M645->Y:$000340,10,1"	"M745->Y:$0003C0,10,1"	"M845->Y:$000440,10,1"

Motor Move Registers	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
Commanded position (1/[Ixx08*32] cts)	M161->D:$000088	M261->D:$000108	M361->D:$000188	M461->D:$000208
Actual position (1/[Ixx08*32] cts)	M162->D:$00008B	M262->D:$00010B	M362->D:$00018B	M462->D:$00020B
Target (end) position (1/[Ixx08*32] cts)	M163->D:$0000C7	M263->D:$000147	M363->D:$0001C7	M463->D:$000247
Position bias (1/[Ixx08*32] cts)	M164->D:$0000CC	M264->D:$00014C	M364->D:$0001CC	M464->D:$00024C
Actual velocity (1/[Ixx09*32] cts/cyc)	"M166->X:$00009D,0,24,S"	"M266->X:$00011D,0,24,S"	"M366->X:$00019D,0,24,S"	"M466->X:$00021D,0,24,S"
Present master pos (1/[Ixx07*32] cts)	M167->D:$00008D	M267->D:$00010D	M367->D:$00018D	M467->D:$00020D
Filter Output (16-bit DAC bits)	"M168->X:$0000BF,8,16,S"	"M268->X:$00013F,8,16,S"	"M368->X:$0001BF,8,16,S"	"M468->X:$00023F,8,16,S"
Compensation correction (1/[Ixx08*32] cts)	M169->D:$000090	M269->D:$000110	M369->D:$000190	M469->D:$000210
Variable jog position/distance (cts)	M172->L:$0000D7	M272->L:$000157	M372->L:$0001D7	M472->L:$000257
Encoder home capture position (cts)	"M173->Y:$0000CE,0,24,S"	"M273->Y:$00014E,0,24,S"	"M373->Y:$0001CE,0,24,S"	"M473->Y:$00024E,0,24,S"
Averaged actual velocity (1/[Ixx09*32] cts/cyc)	M174->D:$0000EF	M274->D:$00016F	M374->D:$0001EF	M474->D:$00026F
Motor following error (1/[Ix08*32] cts)	M180->D:$000091	"M275->X:$000139,8,16,S"	"M375->X:$0001B9,8,16,S"	"M475->X:$000239,8,16,S"

Motor Move Registers	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
Commanded position (1/[Ixx08*32] cts)	M561->D:$000288	M661->D:$000308	M761->D:$000388	M861->D:$000408
Actual position (1/[Ixx08*32] cts)	M562->D:$00028B	M662->D:$00030B	M762->D:$00038B	M862->D:$00040B
Target (end) position (1/[Ixx08*32] cts)	M563->D:$0002C7	M663->D:$000347	M763->D:$0003C7	M863->D:$000447
Position bias (1/[Ixx08*32] cts)	M564->D:$0002CC	M664->D:$00034C	M764->D:$0003CC	M864->D:$00044C
Actual velocity (1/[Ixx09*32] cts/cyc)	"M566->X:$00029D,0,24,S"	"M666->X:$00031D,0,24,S"	"M766->X:$00039D,0,24,S"	"M866->X:$00041D,0,24,S"
Present master pos (1/[Ixx07*32] cts)	M567->D:$00028D	M667->D:$00030D	M767->D:$00038D	M867->D:$00040D
Filter Output (16-bit DAC bits)	"M568->X:$0002BF,8,16,S"	"M668->X:$00033F,8,16,S"	"M768->X:$0003BF,8,16,S"	"M868->X:$00043F,8,16,S"
Compensation correction (1/[Ixx08*32] cts)	M569->D:$000290	M669->D:$000310	M769->D:$000390	M869->D:$000410
Variable jog position/distance (cts)	M572->L:$0002D7	M672->L:$000357	M772->L:$0003D7	M872->L:$000457
Encoder home capture position (cts)	"M573->Y:$0002CE,0,24,S"	"M673->Y:$00034E,0,24,S"	"M773->Y:$0003CE,0,24,S"	"M873->Y:$00044E,0,24,S"
Averaged actual velocity (1/[Ixx09*32] cts/cyc)	M574->D:$0002EF	M674->D:$00036F	M774->D:$0003EF	M874->D:$00046F
Motor following error (1/[Ix08*32] cts)	"M575->X:$0002B9,8,16,S"	"M675->X:$000339,8,16,S"	"M775->X:$0003B9,8,16,S"	"M875->X:$000439,8,16,S"

C. S. End-of-Calculated Move Positions	Coordinate System 1	Coordinate System 2	Coordinate System 3	Coordinate System 4
A-axis target position (engineering units)	M5141->L:$002041	M5241->L:$002141	M5341->L:$002241	M5441->L:$002341
B-axis target position (engineering units)	M5142->L:$002042	M5242->L:$002142	M5342->L:$002242	M5442->L:$002342
C-axis target position (engineering units)	M5143->L:$002043	M5243->L:$002143	M5343->L:$002243	M5443->L:$002343
U-axis target position (engineering units)	M5144->L:$002044	M5244->L:$002144	M5344->L:$002244	M5444->L:$002344
V-axis target position (engineering units)	M5145->L:$002045	M5245->L:$002145	M5345->L:$002245	M5445->L:$002345
W-axis target position (engineering units)	M5146->L:$002046	M5246->L:$002146	M5346->L:$002246	M5446->L:$002346
X-axis target position (engineering units)	M5147->L:$002047	M5247->L:$002147	M5347->L:$002247	M5447->L:$002347
Y-axis target position (engineering units)	M5148->L:$002048	M5248->L:$002148	M5348->L:$002248	M5448->L:$002348
Z-axis target position (engineering units)	M5149->L:$002049	M5249->L:$002149	M5349->L:$002249	M5449->L:$002349

C. S. End-of-Calculated Move Positions	Coordinate System 5	Coordinate System 6	Coordinate System 7	Coordinate System 8
A-axis target position (engineering units)	M5541->L:$002441	M5641->L:$002541	M5741->L:$002641	M5841->L:$002741
B-axis target position (engineering units)	M5542->L:$002442	M5642->L:$002542	M5742->L:$002642	M5842->L:$002742
C-axis target position (engineering units)	M5543->L:$002443	M5643->L:$002543	M5743->L:$002643	M5843->L:$002743
U-axis target position (engineering units)	M5544->L:$002444	M5644->L:$002544	M5744->L:$002644	M5844->L:$002744
V-axis target position (engineering units)	M5545->L:$002445	M5645->L:$002545	M5745->L:$002645	M5845->L:$002745
W-axis target position (engineering units)	M5546->L:$002446	M5646->L:$002546	M5746->L:$002646	M5846->L:$002746
X-axis target position (engineering units)	M5547->L:$002447	M5647->L:$002547	M5747->L:$002647	M5847->L:$002747
Y-axis target position (engineering units)	M5548->L:$002448	M5648->L:$002548	M5748->L:$002648	M5848->L:$002748
Z-axis target position (engineering units)	M5549->L:$002449	M5649->L:$002549	M5749->L:$002649	M5849->L:$002749

Coordinate System Status Bits	Coordinate System 1	Coordinate System 2	Coordinate System 3	Coordinate System 4
Program-running bit	"M5180->X:$002040,0,1"	"M5280->X:$002140,0,1"	"M5380->X:$002240,0,1"	"M5480->X:$002340,0,1"
Circle-radius-error bit	"M5181->Y:$00203F,21,1"	"M5281->Y:$00213F,21,1"	"M5381->Y:$00223F,21,1"	"M5481->Y:$00233F,21,1"
Run-time-error bit	"M5182->Y:$00203F,22,1"	"M5282->Y:$00213F,22,1"	"M5382->Y:$00223F,22,1"	"M5482->Y:$00233F,22,1"
Continuous motion request	"M5184->X:$002040,0,4"	"M5284->X:$002140,0,4"	"M5384->X:$002240,0,4"	"M5484->X:$002340,0,4"
In-position bit (AND of motors)	"M5187->Y:$00203F,17,1"	"M5287->Y:$00213F,17,1"	"M5387->Y:$00223F,17,1"	"M5487->Y:$00233F,17,1"
Warning-following-error bit (OR)	"M5188->Y:$00203F,18,1"	"M5288->Y:$00213F,18,1"	"M5388->Y:$00223F,18,1"	"M5488->Y:$00233F,18,1"
Fatal-following-error bit (OR)	"M5189->Y:$00203F,19,1"	"M5289->Y:$00213F,19,1"	"M5389->Y:$00223F,19,1"	"M5489->Y:$00233F,19,1"
Amp-fault-error bit (OR of motors)	"M5190->Y:$00203F,20,1"	"M5290->Y:$00213F,20,1"	"M5390->Y:$00223F,20,1"	"M5490->Y:$00233F,20,1"

Coordinate System Status Bits	Coordinate System 5	Coordinate System 6	Coordinate System 7	Coordinate System 8
Program-running bit	"M5580->X:$002440,0,1"	"M5680->X:$002540,0,1"	"M5780->X:$002640,0,1"	"M5880->X:$002740,0,1"
Circle-radius-error bit	"M5581->Y:$00243F,21,1"	"M5681->Y:$00253F,21,1"	"M5781->Y:$00263F,21,1"	"M5881->Y:$00273F,21,1"
Run-time-error bit	"M5582->Y:$00243F,22,1"	"M5682->Y:$00253F,22,1"	"M5782->Y:$00263F,22,1"	"M5882->Y:$00273F,22,1"
Continuous motion request	"M5584->X:$002440,0,4"	"M5684->X:$002540,0,4"	"M5784->X:$002640,0,4"	"M5884->X:$002740,0,4"
In-position bit (AND of motors)	"M5587->Y:$00243F,17,1"	"M5687->Y:$00253F,17,1"	"M5787->Y:$00263F,17,1"	"M5887->Y:$00273F,17,1"
Warning-following-error bit (OR)	"M5588->Y:$00243F,18,1"	"M5688->Y:$00253F,18,1"	"M5788->Y:$00263F,18,1"	"M5888->Y:$00273F,18,1"
Fatal-following-error bit (OR)	"M5589-	"M5689-	"M5789-	"M5889->Y:$00273F,19,1"

Motor Axis Definition Registers	**Motor #1**	**Motor #2**	**Motor #3**	**Motor #4**
X/U/A/B/C-Axis scale factor (cts/unit)	M191->L:$0000CF	M291->L:$00014F	M391->L:$0001CF	M491->L:$00024F
Y/V-Axis scale factor (cts/unit)	M192->L:$0000D0	M292->L:$000150	M392->L:$0001D0	M492->L:$000250
Z/W-Axis scale factor (cts/unit)	M193->L:$0000D1	M293->L:$000151	M393->L:$0001D1	M493->L:$000251
Axis offset (cts)	M194->L:$0000D2	M294->L:$000152	M394->L:$0001D2	M494->L:$000252
Motor Axis Definition Registers	**Motor #5**	**Motor #6**	**Motor #7**	**Motor #8**
X/U/A/B/C-Axis scale factor (cts/unit)	M591->L:$0002CF	M691->L:$00034F	M791->L:$0003CF	M891->L:$0D62
Y/V-Axis scale factor (cts/unit)	M592->L:$0002D0	M692->L:$000350	M792->L:$0003D0	M892->L:$0D63
Z/W-Axis scale factor (cts/unit)	M593->L:$0002D1	M693->L:$000351	M793->L:$0003D1	M893->L:$0D64
Axis offset (cts)	M594->L:$0002D2	M694->L:$000352	M794->L:$0003D2	M894->L:$0D65
Coordinate System Variables	**Coordinate System 1**	**Coordinate System 2**	**Coordinate System 3**	**Coordinate System 4**
Isx11 timer (for synchronous assignment)	M5111->X:$002015	M5211->X:$002115	M5311->X:$002215	M5411->X:$002315
Isx12 timer (for synchronous assignment)	M5111->Y:$002015	M5212->Y:$002115	M5312->Y:$002215	M5412->Y:$002315
Host commanded time base (I10 units)	"M5197->X:$002000,0,24,S"	"M5297->X:$002100,0,24,S"	"M5397->X:$002200,0,24,S"	"M5497->X:$002300,0,24,S"
Present time base (I10 units)	"M5198->X:$002002,0,24,S"	"M5298->X:$002102,0,24,S"	"M5398->X:$002202,0,24,S"	"M5498->X:$002302,0,24,S"
Coordinate System Variables	**Coordinate System 5**	**Coordinate System 6**	**Coordinate System 7**	**Coordinate System 8**
Isx11 timer (for synchronous assignment)	M5511->X:$002415	M5611->X:$002515	M5711->X:$002615	M5811->X:$002715
Isx12 timer (for synchronous assignment)	M5512->Y:$002415	M5612->Y:$002515	M5712->Y:$002615	M5812->Y:$002715
Host commanded time base (I10 units)	"M5597->X:$002400,0,24,S"	"M5697->X:$002500,0,24,S"	"M5797->X:$002600,0,24,S"	"M5897->X:$002700,0,24,S"
Present time base (I10 units)	"M5598->X:$002402,0,24,S"	"M5698->X:$002502,0,24,S"	"M5798->X:$002602,0,24,S"	"M5898->X:$002702,0,24,S"

APPENDIX G — FIRST DIGITAL I/O ACCESSORY M-VARIABLES

Name	Definition
MI/O0	M7000->Y:$078C00,0,1
MI/O1	M7001->Y:$078C00,1,1
MI/O2	M7002->Y:$078C00,2,1
MI/O3	M7003->Y:$078C00,3,1
MI/O4	M7004->Y:$078C00,4,1
MI/O5	M7005->Y:$078C00,5,1
MI/O6	M7006->Y:$078C00,6,1
MI/O7	M7007->Y:$078C00,7,1
MI/O8	M7008->Y:$078C01,0,1
MI/O9	M7009->Y:$078C01,1,1
MI/O10	M7010->Y:$078C01,2,1
MI/O11	M7011->Y:$078C01,3,1
MI/O12	M7012->Y:$078C01,4,1
MI/O13	M7013->Y:$078C01,5,1
MI/O14	M7014->Y:$078C01,6,1
MI/O15	M7015->Y:$078C01,7,1
MI/O16	M7016->Y:$078C02,0,1
MI/O17	M7017->Y:$078C02,1,1
MI/O18	M7018->Y:$078C02,2,1
MI/O19	M7019->Y:$078C02,3,1
MI/O20	M7020->Y:$078C02,4,1
MI/O21	M7021->Y:$078C02,5,1
MI/O22	M7022->Y:$078C02,6,1
MI/O23	M7023->Y:$078C02,7,1
MI/O24	M7024->Y:$078C03,0,1
MI/O25	M7025->Y:$078C03,1,1
MI/O26	M7026->Y:$078C03,2,1
MI/O27	M7027->Y:$078C03,3,1
MI/O28	M7028->Y:$078C03,4,1
MI/O29	M7029->Y:$078C03,5,1
MI/O30	M7030->Y:$078C03,6,1
MI/O31	M7031->Y:$078C03,7,1
MI/O32	M7032->Y:$078C04,0,1
MI/O33	M7033->Y:$078C04,1,1
MI/O34	M7034->Y:$078C04,2,1
MI/O35	M7035->Y:$078C04,3,1
MI/O36	M7036->Y:$078C04,4,1
MI/O37	M7037->Y:$078C04,5,1
MI/O38	M7038->Y:$078C04,6,1
MI/O39	M7039->Y:$078C04,7,1
MI/O40	M7040->Y:$078C05,0,1
MI/O41	M7041->Y:$078C05,1,1
MI/O42	M7042->Y:$078C05,2,1
MI/O43	M7043->Y:$078C05,3,1
MI/O44	M7044->Y:$078C05,4,1
MI/O45	M7045->Y:$078C05,5,1
MI/O46	M7046->Y:$078C05,6,1
MI/O47	M7047->Y:$078C05,7,1

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP