KMotion User Manual

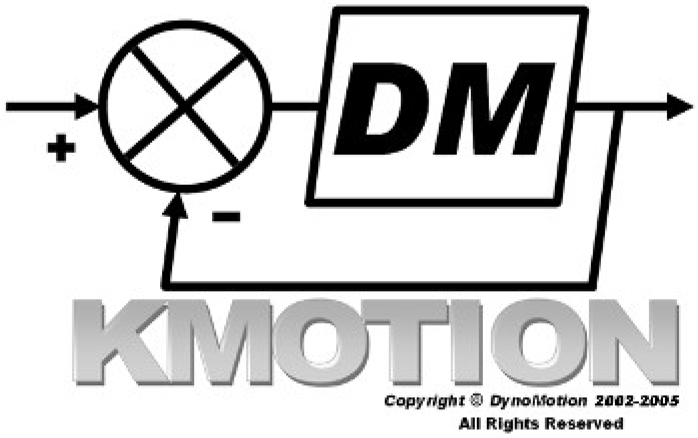
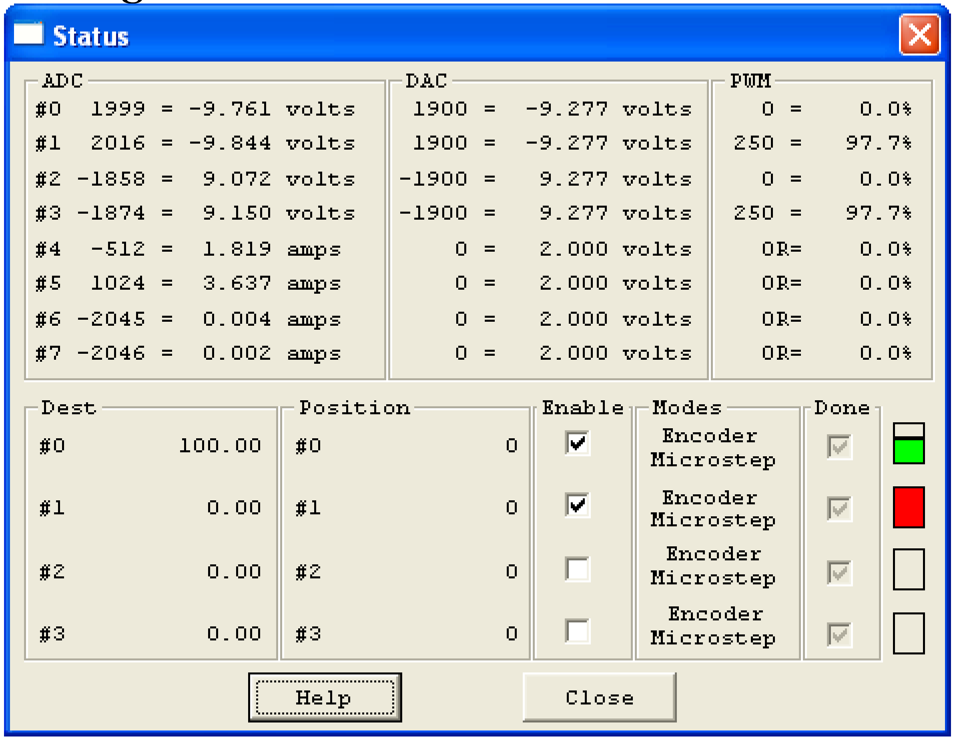


Table of Contents - KMotion

Analog I/O Status Screen



```
ADC                              DAC                        PWM
#0  1999 = -9.761 volts           1900 =  -9.277 volts       0 =   0.0%
#1  2016 = -9.844 volts           1900 =  -9.277 volts     250 =  97.7%
#2 -1858 =  9.072 volts          -1900 =   9.277 volts       0 =   0.0%
#3 -1874 =  9.150 volts          -1900 =   9.277 volts     250 =  97.7%
#4  -512 =  1.819 amps               0 =   2.000 volts      OR=   0.0%
#5  1024 =  3.637 amps               0 =   2.000 volts      OR=   0.0%
#6 -2045 =  0.004 amps               0 =   2.000 volts      OR=   0.0%
#7 -2046 =  0.002 amps               0 =   2.000 volts      OR=   0.0%

Dest               Position            Enable  Modes       Done
#0       100.00    #0              0    ✓      Encoder      ✓    🟩
                                               Microstep
#1         0.00    #1              0    ✓      Encoder      ✓    🟥
                                               Microstep
#2         0.00    #2              0    ☐      Encoder      ✓    ☐
                                               Microstep
#3         0.00    #3              0    ☐      Encoder      ✓    ☐
                                               Microstep
```

(Click on above image to jump to relative topic)

The ***Analog I/O Status Screen*** displays various analog measurements and commands including:

- Measured +/10 V ADC inputs
- Measured current flow per axis
- Commanded +/10 V DAC outputs
- Commanded 0-4V DAC outputs
- Commanded PWM power amp settings
- Commanded Destinations of each axis
- Measured Position of each axis
- Status and Mode of each axis

KMotion has a number of 12-bit ADC and DAC channels that may be used either as servo channel inputs and outputs or as general purpose data acquisition and command signals.

ADC's

KMotion's ADC channels may also be read by a host computer via the Console Script Command: ADCn, or by a on-board C program via the ADC(n) macro. These commands return ADC counts in the range of -2048 to 2047 counts.

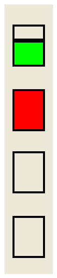
The first four of ***KMotion's*** eight on-board ADC channels are general purpose +/- 10V inputs. Note that in order for these to operate properly, the on-board 5V to +/- 15V power generation must be enabled. The input buffers are inverting such that nominally +10V corresponds to -2048 ADC counts, 0 V corresponds to 0 ADC counts and -10V corresponds to +2047 ADC counts. See the input circuit here. Both the ADCs and DACs are designed to have slightly larger (nominally 2%) than +/- 10V range to account for component tolerance errors. In this way the ranges may always be scaled and offset to provide exact +/-10 V ranges.

Certain of ***KMotion's*** servo input modes allow ADC inputs as position information instead of digital encoder inputs. Such as *ADC Input mode* and *Resolver Input* modes. The *Resolver* servo input mode requires two ADC channels configured as sine and cosine signals. There are insufficient ADC channels for more than two axis to be configured as *Resolver* input mode. When configuring an axis for a mode utilizing an ADC input the corresponding InputChannel(s) (0 and 1) for the axis must be set appropriately. Additionally, the corresponding Input Gains and Offsets for the axis may be used to correct for scale and offsets in either the ADC channels or the sensor itself.

Of ***KMotion's*** *eight* on-board ADC channels, *four* of the channels are dedicated to measuring current flow through each axis of the on-board power amplifiers. ADC Channels 4-7 normally corresponding to axis 0-3 (if default *output channel* mapping is used). Since each axis consists of two full-bridge power amplifiers, the measured current per axis is the sum of *both* full bridges. This provides a convenient means to determine abnormal conditions such as stalled motors or open motor coils. This feature is designed to make an approximate measurement of axis current, not a precise measurement of axis current.

The range of current measurement is fixed at nominally 0 - 4.85 Amps. Since each of ***KMotion's*** full bridge power amplifiers are rated for 3 Amps continuous (per full bridge) and are normally utilized to drive a 4 phase motor (such as a stepper motor), a worst case current per axis occurs when *both* coils are conducting 0.707 of 3 Amps or 4.24 amps. The 4.85 Amp range provides some margin beyond this. An zero current corresponds to an ADC reading in counts of -2048 and 4.85Amps corresponds to an reading of +2047 counts.

Small bar charts on the screen also provide an indication of current in each axis, see below. The graph displays a green bar from 0 to 100% full which corresponds to 0 to 3A of current. Currents higher than 3A show a full red bar which may indicate an over current situation if more than 3A is flowing in any one of the two full bridge circuits.



DAC's

KMotion's DAC channels may be set by a host computer via the Console Script Command: DACn=x, or by a on-board C program via the DAC(n,x) macro. These commands set the DAC counts in the range of -2048 to 2047 counts.

The first four of ***KMotion's*** eight on-board DAC channels are general purpose +/- 10V outputs. Note that in order for these to operate properly, the on-board 5V to +/- 15V power generation must be enabled. The output buffers are inverting such that nominally +10V corresponds to -2048 ADC counts, 0 V corresponds to 0 ADC counts and -10V corresponds to +2047 ADC counts (similar to the ADC readings). See the output circuit here. Both the ADCs and DACs are designed to have slightly larger (nominally 2%) than +/- 10V range to account for component tolerance errors. In this way the ranges may always be scaled and offset to provide exact +/-10 V ranges.

The next four of ***KMotion's*** eight on-board DAC channels are general purpose 0 - 4V outputs. These outputs do *not* require the on-board 5V to +/- 15V power generation to be enabled. 0 V corresponds to -2048 ADC counts and +4V corresponds to +2047 ADC counts.

PWM's

The state of ***KMotion's*** eight PWM (Pulse Width Modulators) are displayed in the top right area of the status screen. The PWM's are connected to on-board *Full Bridge Drivers* to allow direct control of various motors or loads. See the description of KMotion's Power Amplifiers and PWM's for details. The PWM's may operate independently to drive a full bridge driver, or they may function as a pair of PWM's connected to a pair of Full Bridge drivers to drive a 3 phase load. The default and recommended PWM assignment is to have:

Axis 0 configured to utilize PWM 0 and 1

Axis 1 configured to utilize PWM 2 and 3

Axis 2 configured to utilize PWM 4 and 5

Axis 3 configured to utilize PWM 6 and 7

However the PWMs may be reassigned by changing the OutputChan0 and OutputChan1 parameters for an axis. Only consecutive even and odd PWMS (0/1, 2/3, 4/5, or 6/7) may be paired to drive a 3 phase load.

Therefore, there are three possible modes for each PWM channels:

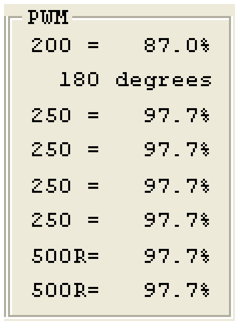
- Normal Independent
- Recirculating Independent
- 3 Phase - paired

If a PWM channel is operating in Normal mode, the PWM channel status will show the value in counts (-255 ... +255) and the percent of full scale.

If a PWM channel is operating in Recirculating mode, the PWM channel status will show the value in counts (-511 ... +511), followed by an "R", and the percent of full scale.

If a pair of PWM channels is operating in 3 Phase mode, the PWM channel status will show the value in counts (-230 ... 230) after the first PWM channel and the phase angle in degrees after second PWM channel.

The example status below shows PWM channels 0 and 1 operating in 3 phase mode, PWM channels 2-5 operating in Normal mode, and PWM channels 6 and 7 operating in Recirculating mode.


```
PWM
200 =   87.0%
  180 degrees
250 =   97.7%
250 =   97.7%
250 =   97.7%
250 =   97.7%
500R=   97.7%
500R=   97.7%
```

Dest

This section of status displays the currently commanded destination for each axis. The Destination is the theoretically desired position for the axis. If a move is in progress, this is the current point along the desired trajectory of the motion. The units are in position units for the axis.

Pos

This section of status displays the currently measured position for each axis. Depending on the input mode for the axis, this position may represent an encoder position, a resolver position, or an absolute position based on an ADC reading. An axis operating as an open loop microstepper mode may have no position sensor at all. In this case the position value is meaningless and ignored.

Enable

Displays whether an axis is enabled or not. Clicking on the Enabled checkbox will enable the axis if disabled or disable the axis if enabled.

Modes

Displays the currently configured input and output modes for each axis. The input mode determines what technique is used to determine the axis's measured position. The output mode determines how the axis's output should be driven.

Valid input modes are:

- Encoder
- ADC
- Resolver
- User Input

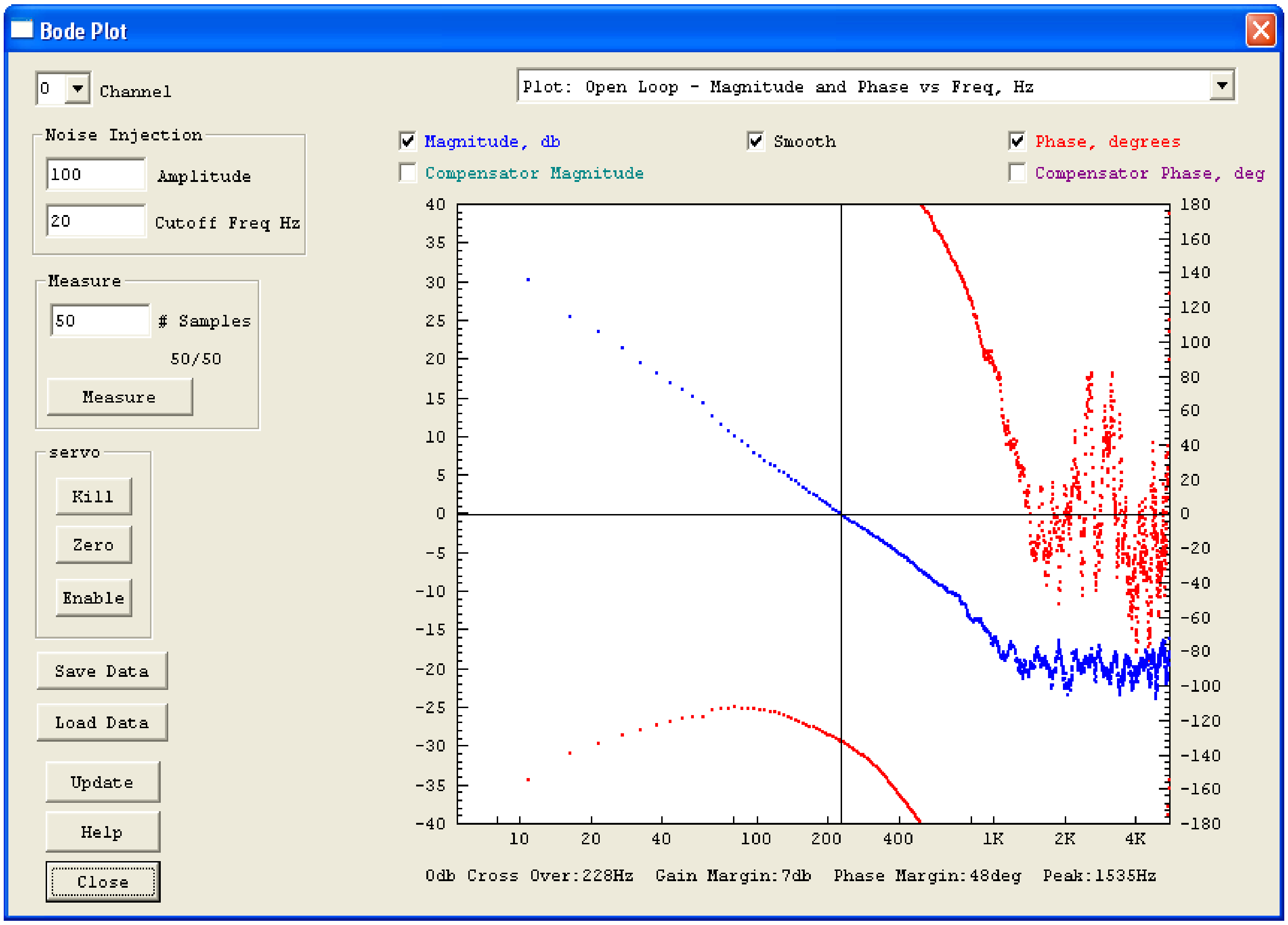
Valid output modes are:

- Microstep
- DC Servo
- 3PH Servo
- 4PH Servo
- DAC Servo

Done

Displays whether an axis operating as an independent axis (not a coordinated motion axis) has completed its last trajectory.

Bode Plot Screen



0 Channel

Plot: Open Loop - Magnitude and Phase vs Freq, Hz

Noise Injection
100 Amplitude
20 Cutoff Freq Hz

Measure
50 # Samples
50/50

☑ Magnitude, db ☑ Smooth ☑ Phase, degrees
☐ Compensator Magnitude ☐ Compensator Phase, deg

0db Cross Over:228Hz Gain Margin:7db Phase Margin:48deg Peak:1535Hz

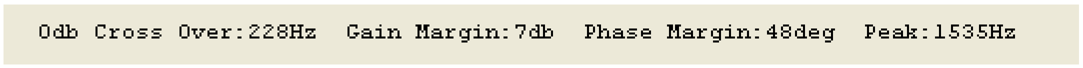
(Click on above image to jump to relative topic)

The ***Bode Plot Screen*** allows the measurement of a servo loop and graphs the open loop response. A Bode Plot is by far the most common means used to measure and understand the behavior of a control loop. ***KMotion*** contains advanced built-in features to allow rapid Bode Plot measurement and display. The current PID and IIR Filter transfer functions may also be superimposed and graphed.

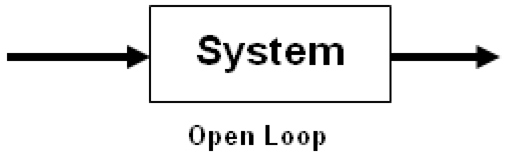
A Bode Plot is a plot of magnitude and phase of a system with respect to frequency. Any linear system that is driven by a sinusoidal input for a long time, will output an sinusoidal signal of the same frequency. The output signal may be shifted in phase and of a different magnitude than the input. A Bode plot is a graph of both the change in phase and the relative change in magnitude (expressed in decibels, db), as a function of frequency.

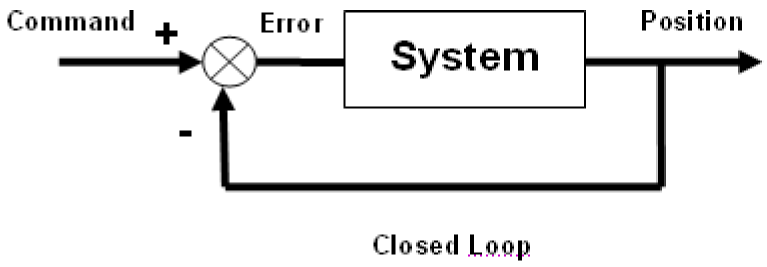
A Bode plot is a useful tool used to examine the stability of a servo feedback loop. If a system has an open loop gain of -1 (magnitude of 0 db and phase of -180 degrees), then if it is placed into a negative feedback loop, it will become unstable and oscillate. Because a system's gain and phase vary as function of frequency, if the system has a magnitude of 0db and phase of -180 degrees at *any frequency* it will be unstable and oscillate at that frequency. The way to avoid an unstable system is to avoid having simultaneous conditions of 0db and -180 degrees occur at any frequency. Where the magnitude of the system is 0db the amount that the phase is different from -180 degrees is called the *phase margin*. The larger the phase margin the more stable the system. Similarly where the phase is -180 degrees the amount that the magnitude is different from 0db is called the *gain margin*. Again the larger the gain margin, the more stable the system. As a general rule of thumb, for a system to be reasonably stable it should have a phase margin of at least 30 degrees and a gain margin of at least 3 db.

The Bode Plot Screen attempts to identify and measure the 0 db crossover frequency (the first point where the open loop magnitude becomes less than 1, often defined as the system bandwidth, 228 Hz on the example above), the gain and phase margins, and one or two sharp peaks in the magnitude data after the crossover. Some mechanical systems have sharp resonant peaks that may cause the system to go unstable if these peaks approach the 0 db line and have phase near -180 degrees. A notch filter placed at these frequencies may increase performance. The measurements are displayed under the graph as shown below.

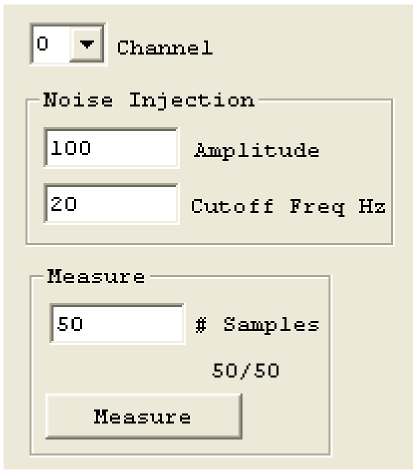


```
0db Cross Over:228Hz  Gain Margin:7db  Phase Margin:48deg  Peak:1535Hz
```





The most direct method for obtaining the open loop response, is to break the servo loop open, inject a signal into the system and measure the output. However this is usually impractical as most systems will run out of their linear range if driven in an open loop manner. ***KMotion*** operates the servo loop in its normal closed loop form, injects a command signal, measures the position response, and mathematically derives the open loop response. This does require that the servo loop function in some form as a stable closed loop servo *before* a measurement may be made. Performance is not a requirement so low gains might be used to obtain an initial stable system.



Channel: 0

Noise Injection

Amplitude: 100

Cutoff Freq Hz: 20

Measure

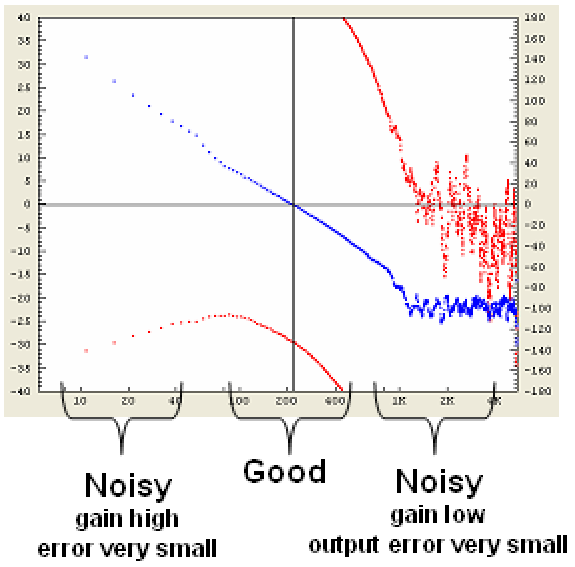
# Samples: 50

50/50

To perform a *Bode Plot* Measurement: select the *channel* to measure, select the desired *Amplitude and Cutoff Frequency* for the stimulus to be injected, select the *# of samples* to average, and depress the *Measure* Pushbutton. All current Configuration Parameters (from the Configuration Screen), Tuning Parameters (from the Step Response Screen), and Filter Parameters (from the IIR Filter Screen) will be downloaded, the selected Axis channel will be enabled, and the measurement will begin.

While the measurement is in progress the number of samples acquired will be displayed and the *Measure* Pushbutton will change to a *Stop* Pushbutton. Pushing the Stop button will stop acquiring data after the current sample is complete.

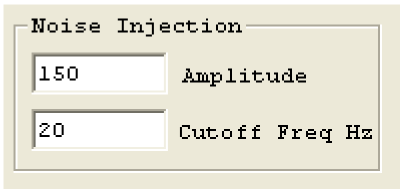
Depending on the type of plot requested (either Time Domain or Frequency Domain) either the last acquired time domain measurement will be displayed or the *average* all the frequency domain measurement so far acquired will be displayed.



Unfortunately Bode Plots often have regions that are very noisy. But *fortunately* these are almost always in regions that are not important to us. At frequencies where the open loop gain is very high (usually at low frequencies), the servo loop performs very well, and the *Position* closely matches the *Command* signal. In this case, the calculation of the *Error* signal (see above) is calculated by taking the difference between two nearly equal values. A small error in Position measurement will then result in a relatively large error in the calculated error value. Similarly, when the system has a very low gain (usually at high frequencies), the position signal is very small and often highly influenced by noise, or if an encoder is used, by encoder resolution. The regions of interest in order to determine system stability, are where the open loop gain is near 0db and the measurements are normally very accurate.

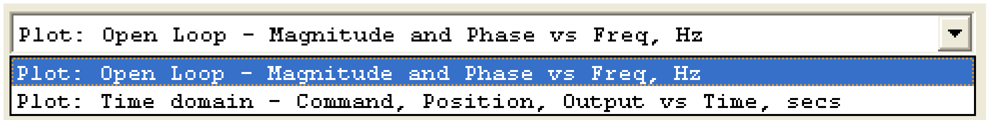
Additionally, instead of injecting sine waves at various frequencies individually, ***KMotion*** uses a technique where a random noise signal that is rich in *many* frequencies is injected. Using a method involving an FFT (Fast Fourier Transform) of the input and output, the entire frequency response may be obtained at once.

Bode Plot analysis is fundamentally based on the assumption that the system being analyzed is *linear*. Linear in this context means that any signal injected into a system that provides a response, might be broken into parts, each piece injected separately, and all the resulting responses when summed would equal the original response. If a system being measured does not meet this criteria then the result is basically useless and meaningless. Masses, Springs, Dampers, Inductors, Resistors, Capacitors, and all combinations thereof are examples of devices which produce very linear effects. Static friction, Saturation, and Encoder count quantization, are examples of non-linear effects.

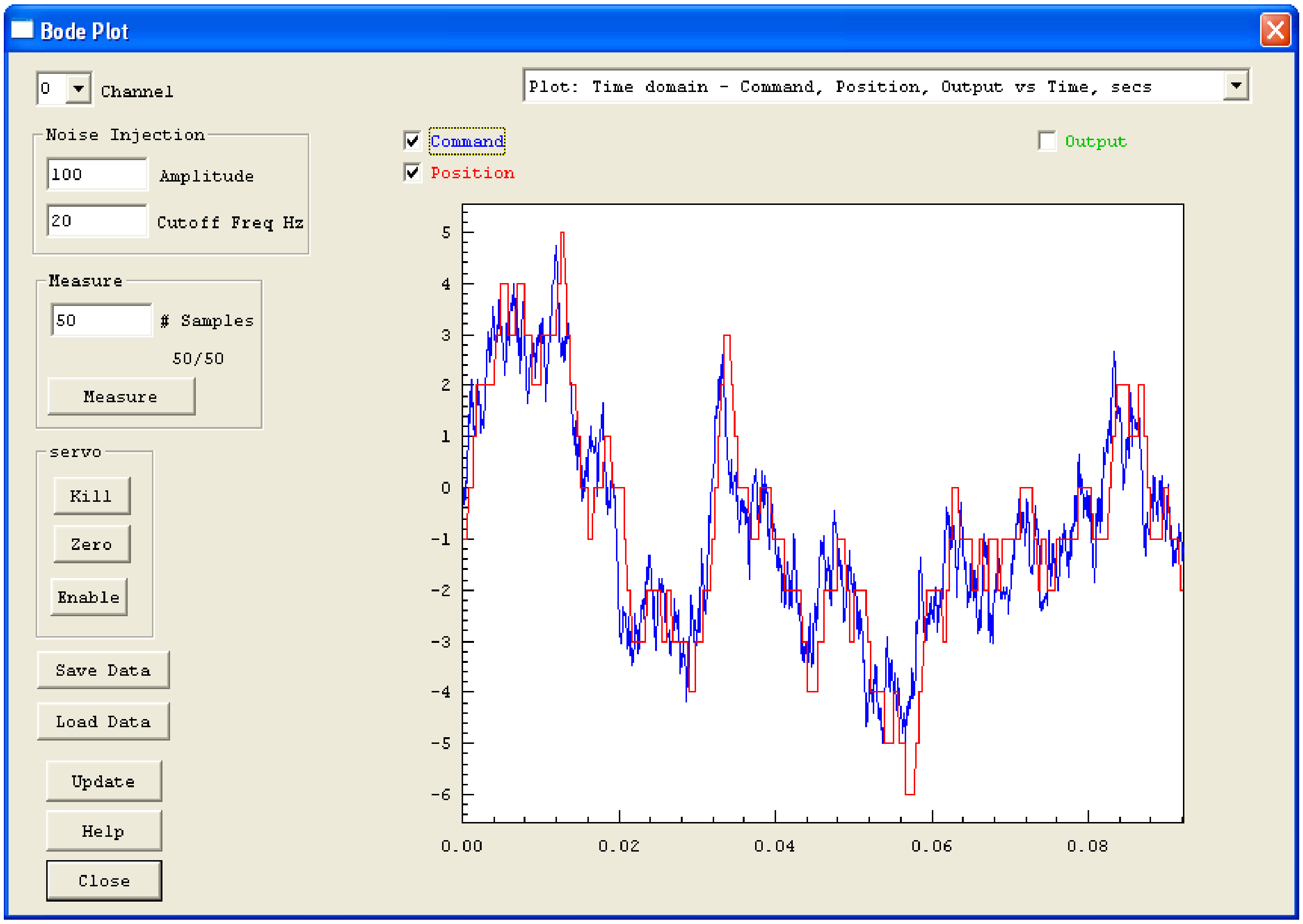

Noise Injection
150 Amplitude
20 Cutoff Freq Hz

It is therefore very important to verify that while the system is being measured that it is operating in its linear range. This usually entails that the system isn't being driven too hard (or too fast), so that the drive to the system (*Output*) is not reaching saturation. Additionally, it is important to verify that the system is being driven sufficiently hard (or slowly enough) that a measurable *Position* change is being observed. The *Noise Injection Amplitude* and *Cutoff Frequency* should be adjusted to optimize these conditions. *Noise Amplitude* has the same units as *Position* Measurement. It should be noted that setting the *Cutoff Frequency* very low, may reduce the high frequency stimulation to the system to such a point that the higher frequency measurements are invalid or very noisy.

The Bode Plot Screen allows the measurement data to be viewed in the *time domain* in order to check the signal amplitudes so that the optimal signal levels may be used in order to minimize the non-linear effects of saturation and quantization. Select the Plot Type drop down list shown below to switch between frequency domain and dime domain displays.

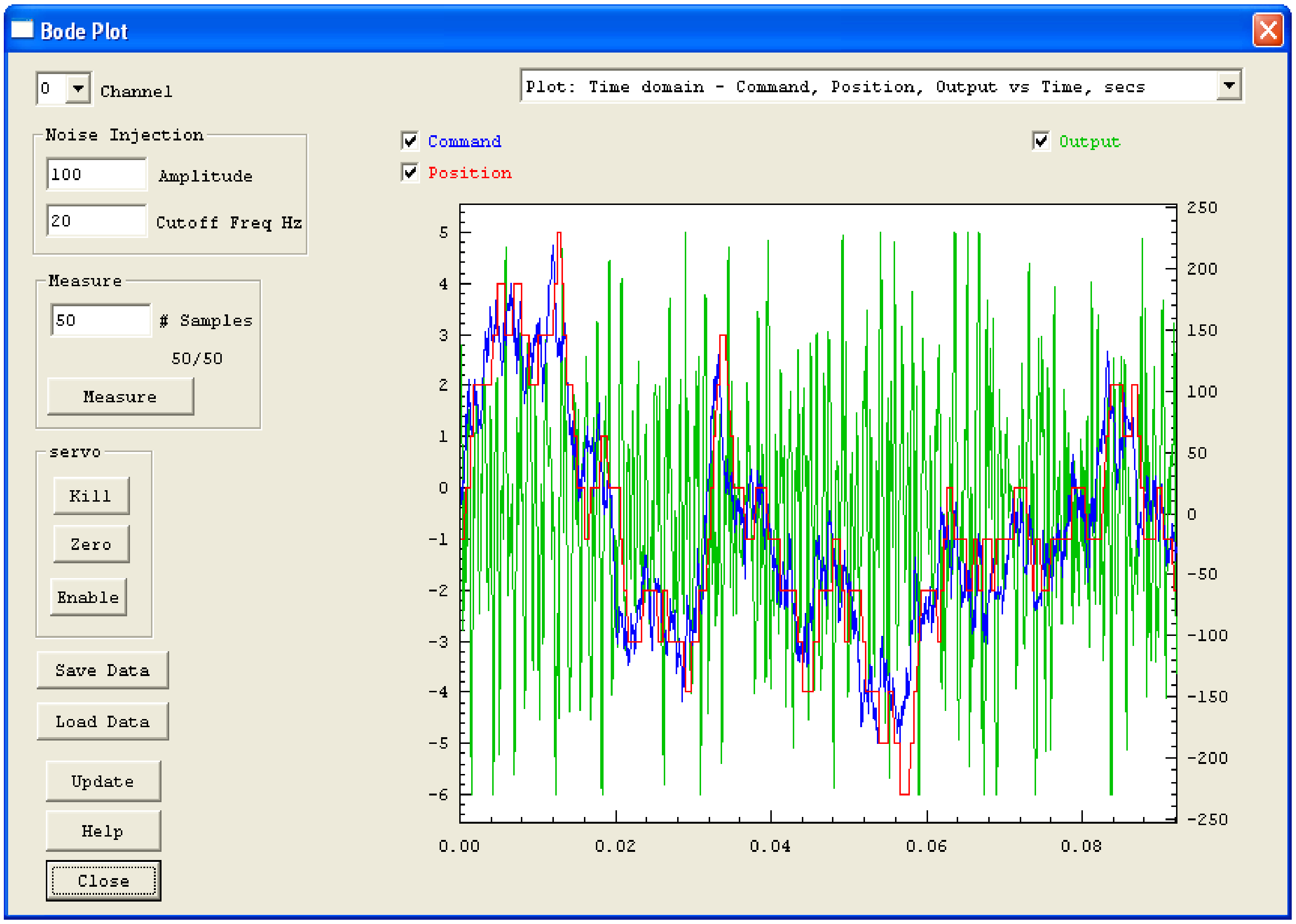

Plot: Open Loop - Magnitude and Phase vs Freq, Hz
Plot: Open Loop - Magnitude and Phase vs Freq, Hz
Plot: Time domain - Command, Position, Output vs Time, secs

A typical time domain measurement is shown below. The *blue* graph shows the random stimulus to the system. The *red* graph shows the system's response, which in this example is the output of an encoder. Note that the position is quantized to integer counts and has a range of approximately 10 counts. This is nearly the minimum number of counts to expect a reasonable Bode Plot measurement. A larger range of encoder counts could be obtained by driving the system harder by increasing the Noise Injection Amplitude, provided there is additional output drive available.

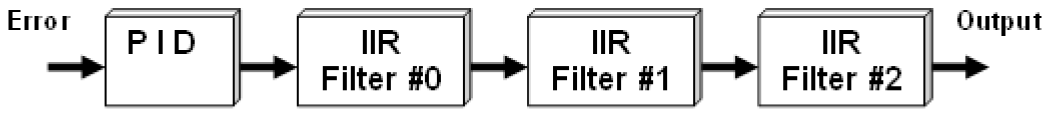


The graphs below include the output drive signal shown in *green*. Note that the output drive is on the verge of saturating at the maximum allowed output value (example is from a 3 phase brushless motor, maximum allowed PWM counts of 230). This shows that we are driving the system as hard as possible without saturating in order to obtain the most accurate measurement.

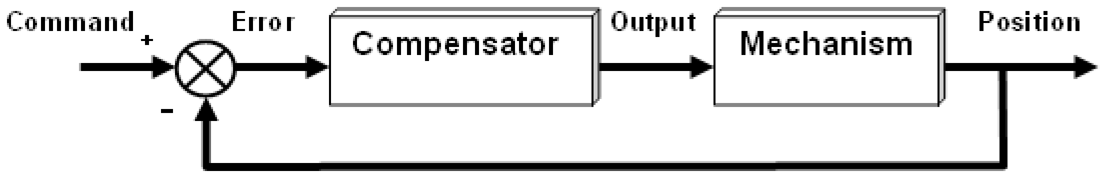
Another means of improving the measurement accuracy (as well as servo performance in general) is obviously to increase the encoder resolution if economically or otherwise feasible.


Bode Plot
0
Channel
Noise Injection
100
Amplitude
20
Cutoff Freq Hz
Measure
50
# Samples
50/50
Measure
servo
Kill
Zero
Enable
Save Data
Load Data
Update
Help
Close
Plot: Time domain - Command, Position, Output vs Time, secs
Command
Position
Output
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
250
200
150
100
50
0
-50
-100
-150
-200
-250
0.00
0.02
0.04
0.06
0.08

Compensator Response

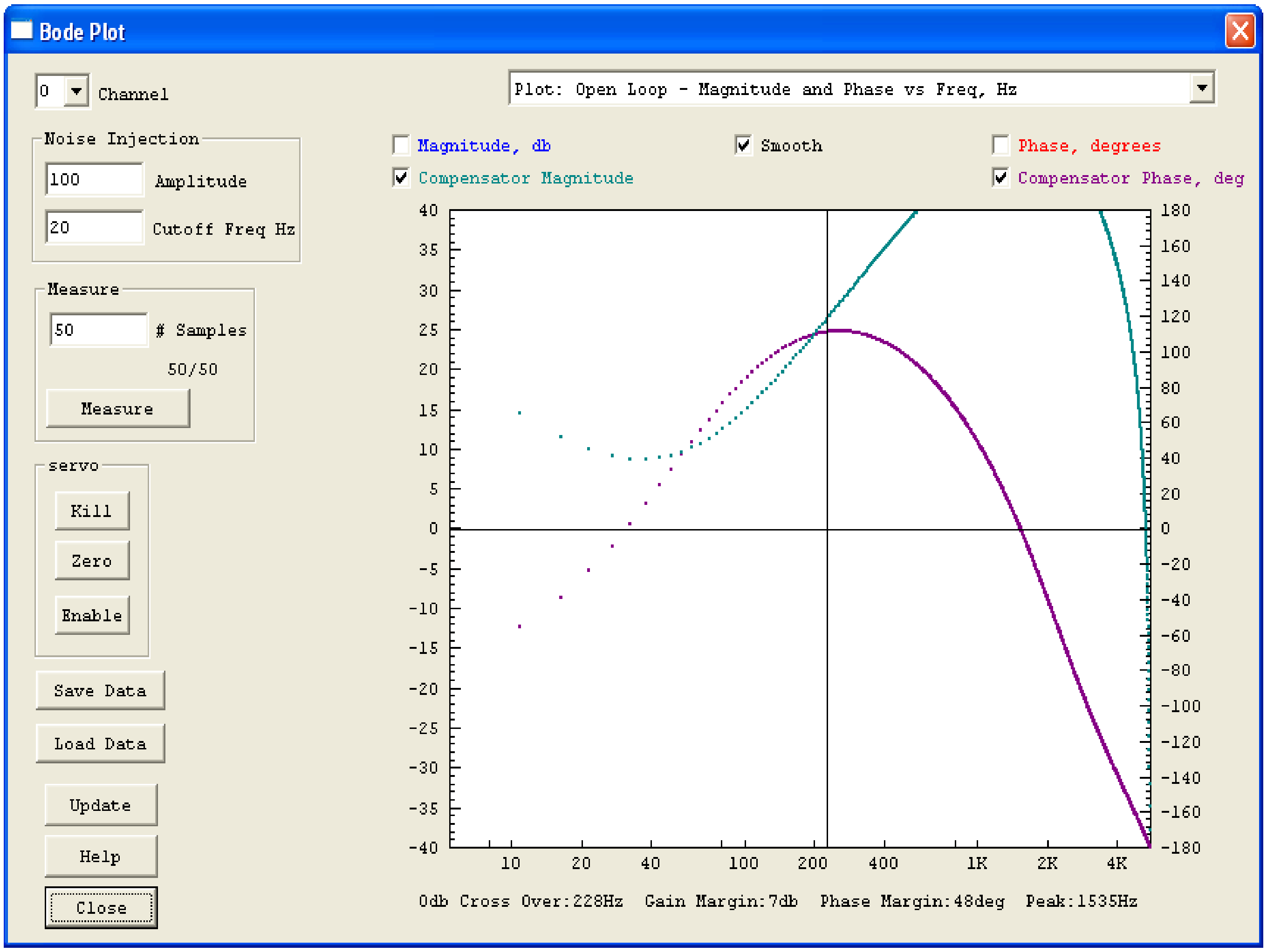


KMotion Compensator



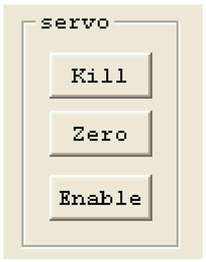
Complete Closed Loop Servo

The *Bode Plot Screen* is also capable of graphing a *Bode Plot* of the combined PID and IIR Filters. This is often referred to as the *Compensator* for the system. The Cyan graph shows the magnitude of the compensator and the violet graph shows the phase of the compensator. Notice that in this example the maximum phase has been adjusted to match the 0 db crossover frequency of the system (~ 233 Hz) to maximize the system phase margin.



0db Cross Over:228Hz Gain Margin:7db Phase Margin:48deg Peak:1535Hz

Axis Control



The Axis Control buttons are present to conveniently disable (Kill), Zero, or Enable an axis. If the axis becomes unstable (possible due to a gain setting too high), the Kill button may be used to disable the axis, the gain might then be reduced, and then the axis may be enabled. The Enable button downloads all parameters from all screens before enabling the axis in the same manner as the *Measure* button described above.

Note for brushless output modes that commutate the motor based on the current position, Zeroing the position may adversely affect the commutation.

Save/Load Data

The Save/Load Data buttons allow the captured Bode Plot to be saved to a text file and re-loaded at a later time. The text file format also allows the data to be imported into some other program for display or analysis. The file format consists of one line of header followed by one line of 9 comma separated values, one line for each frequency. The values are:

1. Frequency in Hz
2. Input Stimulus - Real Part of complex number
3. Input Stimulus - Complex Part of complex number (always zero)
4. Measured Closed Loop Output - Real Part of complex number
5. Measured Closed Loop Output - Complex Part of complex number
6. Open loop Magnitude - in decibels
7. Open loop Phase - in degrees
8. Open loop Magnitude - in decibels - "smoothed"
9. Open loop Phase - in degrees - "smoothed"

Example of data file follows:

```
Freq,InputRe,InputIm,OutputRe,OutputIm,Mag,Phase,SmoothMag,SmoothPhase
0,2.329706e+007,0,2.344316e+007,0,44.10739,-180,0,0
5.425347,1.968735e+007,0,1.98995e+007,-32055.19,39.34598,-171.5001,39.34598,-171.5001
10.85069,1.816919e+007,0,1.848909e+007,-713239.6,27.48402,-116.3662,29.58283,-139.1553
16.27604,2.024904e+007,0,2.124962e+007,-1383543,21.91849,-129.5997,25.14718,-134.5283
21.70139,1.53403e+007,0,1.645491e+007,-1651059,18.38331,-129.7526,22.70204,-127.6139
27.12674,1.225619e+007,0,1.301412e+007,-1336369,18.60411,-125.4229,20.60267,-123.6751
32.55208,7014539,0,7393482,-958714.3,17.18516,-118.9553,18.9778,-119.2762
```

•

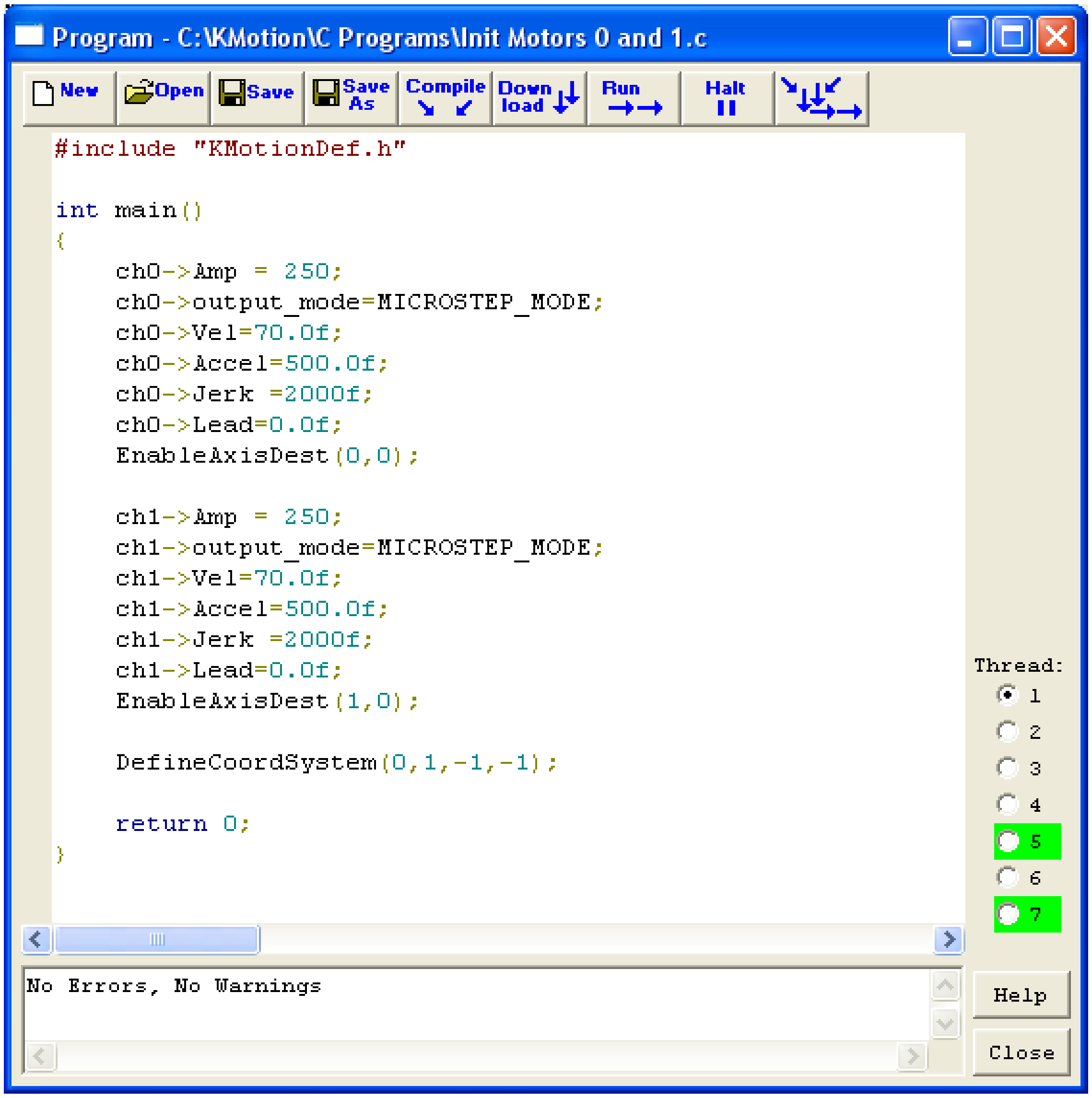
•

•

Update Pushbutton

The Update button may be used to update the displayed *compensator* graph if any parameters (on other screens) have been modified and not downloaded or otherwise acted upon. If Measure, Download, Step, Move, Save, Close, or Enable is used on this or the any other screen then this command is unnecessary, however if none of these commands is performed, then this button may be used to update the graph.

C Program Screen



```
#include "KMotionDef.h"

int main()
{
	ch0->Amp = 250;
	ch0->output_mode=MICROSTEP_MODE;
	ch0->Vel=70.0f;
	ch0->Accel=500.0f;
	ch0->Jerk =2000f;
	ch0->Lead=0.0f;
	EnableAxisDest(0,0);

	ch1->Amp = 250;
	ch1->output_mode=MICROSTEP_MODE;
	ch1->Vel=70.0f;
	ch1->Accel=500.0f;
	ch1->Jerk =2000f;
	ch1->Lead=0.0f;
	EnableAxisDest(1,0);

	DefineCoordSystem(0,1,-1,-1);

	return 0;
}
```

Thread: [✓ 1] 2 3 4 5 6 7

```
No Errors, No Warnings
```

The ***C Program Screen*** allows the user to edit C language programs, compile, link, download, and run them within the ***KMotion*** board. C programs executing within the ***KMotion*** board have direct access to all the Motion, I/O, and other miscellaneous functions incorporated into ***KMotion*** System.
One of the most powerful features of the ***KMotion*** system is the ability for a user to compile and download native DSP programs and have them run in real time. Native DSP code runs faster than interpreted code. The TMS320C67x DSP that powers the ***KMotion*** system has hardware support for both 32 bit *and* 64 bit floating point math. Multiple threads (programs) may execute simultaneously (up to 7). The integrated C compiler allows with a single pushbutton to save, compile, link, download, and execute all within a fraction of a second. After programs have been developed and

tested they may be flashed into memory and run stand alone with no host connection.

Other features of the ***C Program Screen*** include a rich text editor with syntax highlighting, keyword drop down lists, function tips, unlimited undo/redo, and Find/Replace with regular expressions.

See list below for available constants and functions.

For a more details on the functions, see the KMotionDef.h header file. This file is normally included into a user program so that all accessible base functions and data structures are defined.

See PC-DSP.h for common definitions between the PC host and ***KMotion*** DSP.

Constants:

FALSE 0
TRUE 1

PI 3.14159265358979323846264
PI_F 3.1415926535f
TWO_PI (2.0 * PI)
TWO_PI_F (2.0f * PI_F)
PI_2F (PI_F * 0.5f)

TRAJECTORY_OFF 0
TRAJECTORY_INDEPENDENT 1
TRAJECTORY_LINEAR 2
TRAJECTORY_CIRCULAR 3
TRAJECTORY_SPECIAL 4

Axis Input Modes

ENCODER_MODE 1

ADC_MODE 2

RESOLVER 3

USER_INPUT_MODE 4

Axis Output Modes

MICROSTEP_MODE 1

DC_SERVO_MODE 2

BRUSHLESS_3PH_MODE 3
BRUSHLESS_4PH_MODE 4
DAC_SERVO_MODE 4

Data Gather/Plot Functions:
void SetupGatherAllOnAxis(int c, int n_Samples);
void TriggerGather();
int CheckDoneGather();

Analog I/O Functions:
ADC(ch);

DAC(ch, value);

Power Amp Functions:
void WritePWMR(int ch, int v);
void WritePWM(int ch, int v);

void Write3PH(int ch, float v, double angle_in_cycles);
void Write4PH(int ch, float v, double angle_in_cycles);

Timer Functions:

double Time_sec();
void WaitUntil(double time_sec);
void Delay_sec(double sec);
double WaitNextTimeSlice(void);

Axis Move Functions:
void DisableAxis(int ch);
void EnableAxisDest(int ch, double Dest);
void EnableAxis(int ch);
void Zero(int ch);
void Move(int ch, double x);
void MoveRel(int ch, double dx);
int CheckDone(int ch);
void MoveXYZ(double x, double y, double z);
int CheckDoneXYZ();
void DefineCoordSystem(int axisx, int axisy, int axisz, int axis a);
void StopMotion(CHAN *ch);

Digitial I/O Functions:
void SetBitDirection(int bit, int dir);
int GetBitDirection(int bit);
void SetBit(int bit);
void ClearBit(int bit);
void SetStateBit(int bit, int state);
int ReadBit(int bit);

Print to Console Window Functions:
int Print(char *s);
int PrintFloat(char *Format, float v);
int PrintInt(char *Format, int v);

Thread Functions:
void StartThread(int thread);
void PauseThread(int thread);
void ThreadDone();
int ResumeThread(int thread);

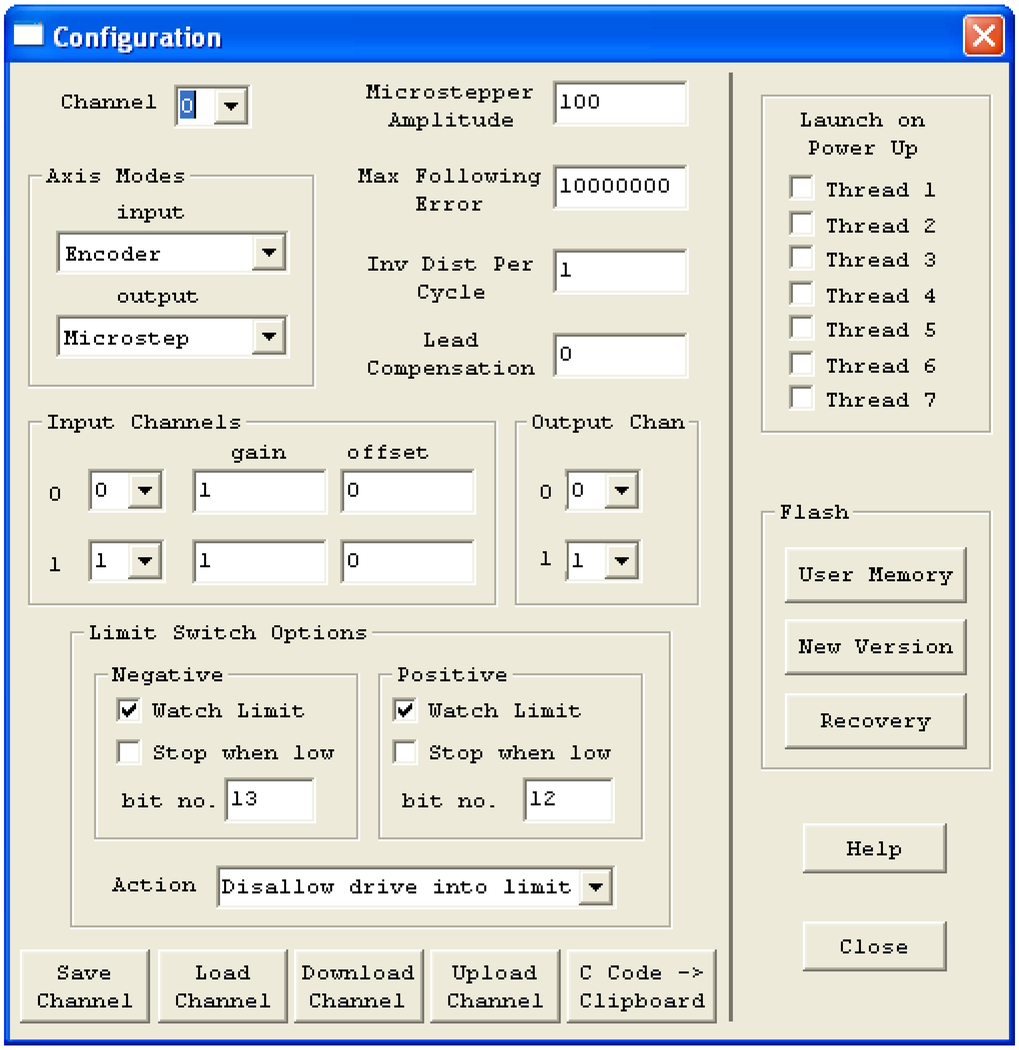
Math Functions:

double sqrt(double x);
double exp(double x);
double log(double x);
double log10(double x);
double pow(double x, double y);
double sin(double x);
double cos(double x);
double tan(double x);
double asin(double x);
double acos(double x);

double atan(double x);
double atan2(double y, double x);

float sqrtf (float x);
float expf (float x);
float logf (float x);
float log10f(float x);
float powf (float x, float y);
float sinf (float x);
float cosf (float x);
float tanf (float x);
float asinf (float x);
float acosf (float x);
float atanf (float x);
float atan2f(float y, float x);

Configuration and FLASH Screen


Configuration

Channel: 0

Microstepper Amplitude: 100

Max Following Error: 10000000

Inv Dist Per Cycle: 1

Lead Compensation: 0

Axis Modes
- input: Encoder
- output: Microstep

Input Channels

		gain	offset
0	0	1	0
1	1	1	0

Output Chan
- 0: 0
- 1: 1

Limit Switch Options
- Negative: [✓] Watch Limit; [ ] Stop when low; bit no. 13
- Positive: [✓] Watch Limit; [ ] Stop when low; bit no. 12
- Action: Disallow drive into limit

Launch on Power Up: [ ] Thread 1, [ ] Thread 2, [ ] Thread 3, [ ] Thread 4, [ ] Thread 5, [ ] Thread 6, [ ] Thread 7

(Click on above image to jump to relative topic)

The ***Configuration and FLASH Screen*** displays and allows changes to ***KMotion's*** configuration and allows the configuration, new firmware, or user programs to be FLASH'ed to non volatile memory.

Channel: 0

Each axis channel is configured independently. To view or make changes to a configuration first select the desired axis channel using the channel drop down. Note that changing an axis on any screen switches the active channel on all other screens simultaneously.

The parameters for each axis's configuration are grouped into three classes: Definitions, Tuning, and Filters. Each class of parameters are displayed on three corresponding screens:

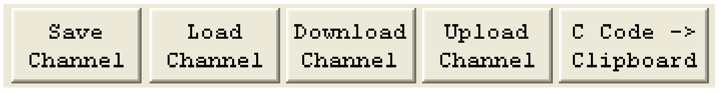
- Configuration Screen
- Step Response Screen
- IIR Filter Screen

The *Configuration Screen* contains *definition* parameters that should be set once and remain set unless a physical change to the hardware is made. For example, a Stepper motor might be replaced with a Brushless Motor and Encoder.

The *Step Response Screen* contains parameters that are *tuning* related and are located where the tuning response is most often adjusted and checked. For example, PID (proportional, integral, derivative) parameters are located there.

The *IIR Filter Screen* contains parameters related to servo *filters*.

Utilities

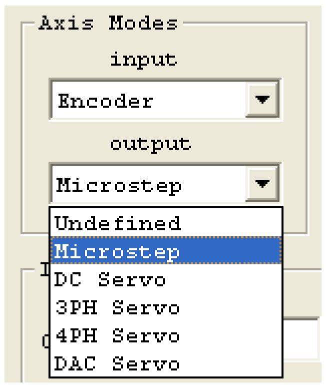


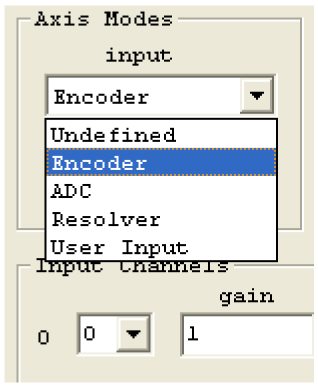
The buttons along the bottom of the Configuration Screen allow a set of axis parameters to be:

- Saved or Loaded from a disk file (*.mot)
- Uploaded or Downloaded to a ***KMotion***
- Converted to equivalent C Code for use in a ***KMotion** C Program*

Note that these buttons operate on all parameters (for one axis) from all screens as a unit.

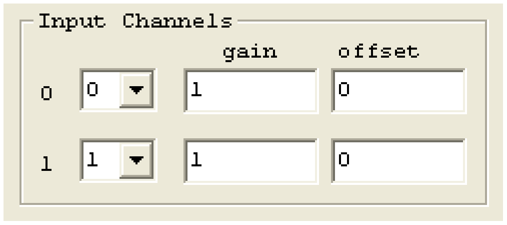
Axis Modes





Use the respective dropdown to set either the axis input or output mode. The input mode defines the type of position measurement (if required) for the axis. Closed loop control always requires some type of position measurement.. For open loop stepper motor control, position measurement is optional. The output mode determines how the output command should be achieved. Either by driving the on board PWMs and Full Bridge Drivers to control a specific type of motor, or by driving a DAC signal that will drive an external power amplifier.

Input Channels



The Input Channels section specifies which channels for the selected input device will be used. This may be the channel of an Encoder input or an ADC input depending on the selected input mode. Resolvers requires two ADC input channels (for sine and cosine), for all other modes the second channel number is not used.

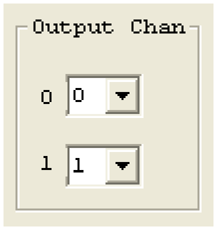
The gain and offset parameters are applied to the respective input device. The gain is applied before the offset, i.e. $x' = ax+b,$ where a is the gain and b is the offset..

Incremental encoders only utilize the gain parameter which may be used to scale or reverse (using a negative gain) the measurement.

A *Resolver* is a device that generates analog sine and cosine signals as it's shaft angle changes. Either one or multiple sine and cosine waves may be produced per revolution of a resolver. An encoder that generates analog sine and cosine signals may also be connected to a ***KMotion*** as though it was a resolver. Resolver inputs may utilize both gains and offsets and be adjusted such that the sine and cosine ADC measurements are symmetrical about zero and have the same amplitude. Gain and offset errors may be introduced by either the ADC input circuitry and/or the Resolver itself. If one were to plot the sine vs. cosine signals as a resolver's position changes, the result should be circle. ***KMotion*** computes the arctangent of the point on the circle (also keeping track of the number of rotations) to obtain the current position. An offset or elliptical "circle" will result in a distorted position measurement throughout the cycle. Therefore note that adjusting the gains and offsets will result in changing the linearity of the position measurement, not the scale of the position measurement itself. The scale of a resolver input will always be 2π radians per cycle.

An ADC input uses a single absolute ADC channel input to obtain the position measurement. Gain0 and Offset0 may be used to modify the ADC counts from -2048 .. +2047 to and desired range.

Output Channels



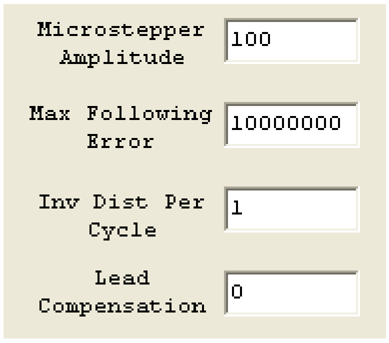
The Output Channels section specifies which channels for the selected output device will be used. This may be the channel of a PWM connected to an on-board power amplifier, or a DAC that is used to drive an external power amplifier.

Stepper mode and 4 phase brushless mode require two channels of PWM to be specified.

DC Servo motor (Brush motor type) only require one PWM channel.

3 Phase brushless motors require a consecutive pair of PWM channels. In 3 Phase output mode, only the Output Channel 0 value is used and must be set to an even PWM number.

Microstepper Amplitude, Max Following Error, Inv Dist Per Cycle, Lead Compensation



Microstepper Amplitude 100

Max Following Error 10000000

Inv Dist Per Cycle 1

Lead Compensation 0

Microstepper Amplitude is only applicable to configurations with output mode of Microstepper. This parameter sets the amplitude (of the sine wave) in PWM counts (0 .. 255) that will be output to the sine and cosine PWM channels while moving slowly or at rest. Note that at higher speeds ***KMotion*** has the ability to increase the amplitude to compensate for motor inductance effects and *may* actually be higher. See *Lead Compensation* in this same section.

Max Following Error is applicable to all closed loop servo output modes (DC Servo, 3 Phase Brushless, 4 Phase brushless, and DAC Servo). Whenever the commanded destination and the measured position differ by greater than this value, the axis will be disabled (if this axis is a member of the defined coordinate system, then any coordinated motion will also stop). To disable following errors set this parameter to a large value.

Inv Dist Per Cycle applies to Stepper, 3 Phase, and 4 Phase motors. For a stepper motor, the *distance per cycle* defines the distance that the commanded destination should change by for a motor coil to be driven through a complete sinusoidal cycle. Parameter should be entered as the inverse (reciprocal) of the distance per cycle. Stepper motors are most often characterized by shaft angle change per "Full Step". A motor coil is driven through a complete cycle every four - "Full Steps". See the following examples:

Example #1 : A mechanism moves 0.001" for each full step of a step motor It is desired for commanded distance to be in inches.

Result: One Cycle = 4 full steps = 0.004", Thus InvDistPerCycle = 1.0/0.004 = 250.0 (cycles/inch). Commanding a move of 1.00 will generate 250 sine waves, or the equivalent of 1000 full steps, or one inch of movement..

Example #2 : InvDistPerCycle is left at the default value of 1.0

Result: Move units are in cycles. Commanding a move of 50 will generate 50 sine waves, or the equivalent of 200 full steps, or one revolution of a 200 Step or 1.8 degree motor.

For 3 Phase or 4 Phase motors, *Inv Dist Per Cycle* represents the inverse of the distance for one complete commutation cycle. See the example below.

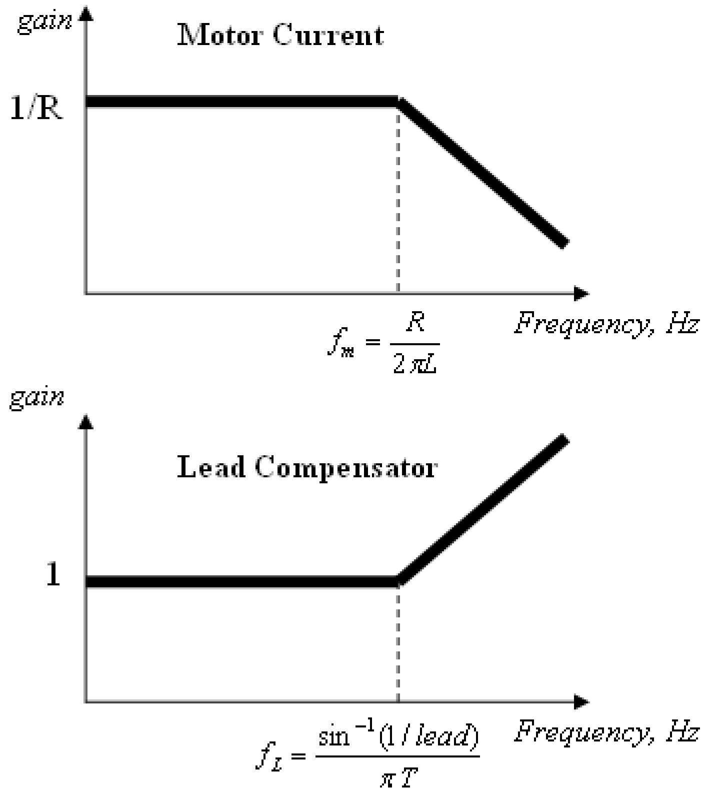
Example #1 : A 3 phase motor/encoder has a 4096 count per revolution encoder which is used for position feedback and for motor commutation. InputGain0 is set to 1.0 so position measurement remains as encoder counts. The motor design is such that the commutation goes through 3 complete cycles each motor revolution.

Result: One Cycle = 4096 counts/3.0 Thus InvDistPerCycle = 3.0/4096 = 0.000732421875.

Note that it is important to use a high degree of precision to avoid commutation errors after moving to very large positions (or at constant velocity for a long period of time). ***KMotion*** maintains *Inv Dist Per Cycle* (as well as position) as a double precision (64 bit) floating point number for this reason (*more than 70 years* at 1 MHz would be required to have 1 count of error)

Lead Compensation may be used to compensate for motor inductance. When a voltage is applied to a coil at a low frequencies, the current flow is dictated by the coil's resistance and is constant. As the frequency increases at some point, where $f_m = \frac{R}{2\pi L}$, the inductance, *L*, begins to dominate and the current drops (see plot below). ***KMotion's*** Lead Compensator has the opposite effect, it has a constant gain of 1 and at some point increases with frequency. The Lead Compensation parameter sets (indirectly) the frequency where this occurs. If the frequency is set to match the frequency of the motor, the effects will cancel, and the motor current (and torque) will remain constant to a much higher frequency.

This assumes that the nominal drive voltage is lower than the available supply voltage. For example, a 5V stepper motor might be driven with a 15V supply to allow head room for the applied voltage to be increased at high frequencies (speeds).



The simple formula that implements the Lead Compensation is:

$v' = v + \Delta v\, L$

where *v* is the voltage before the compensation, *v'* is the voltage after the compensation, *Δv* is the change in output voltage from the last servo sample, and *L* is the Lead Compensation value.

The following formula will compute the "knee" frequency for a particular *lead* and servo sample rate (normally *T*=90 us).

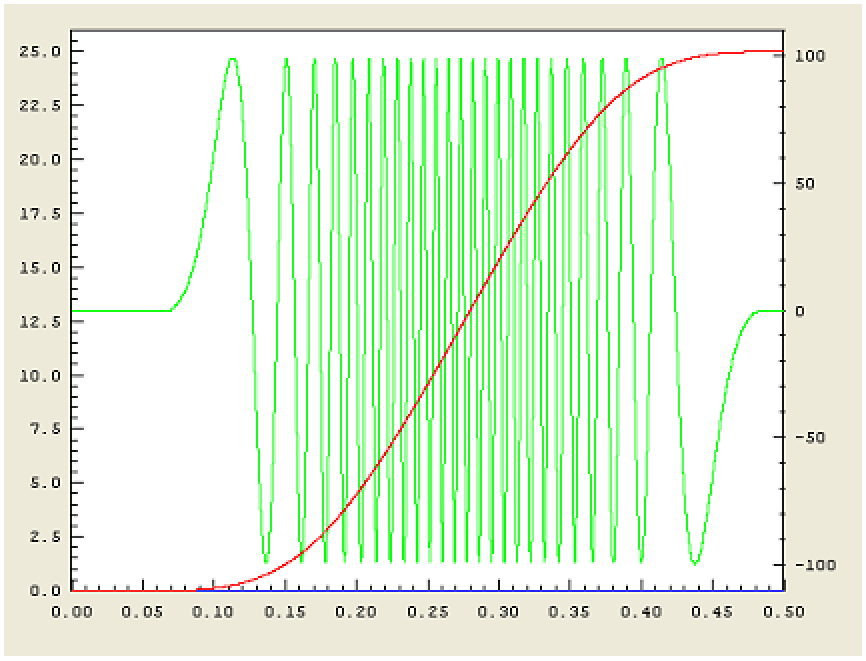
$$f_L = \frac{\sin^{-1}(1/lead)}{\pi T}$$

or the inverse of this formula will provide the lead value to position the knee at a particular frequency.

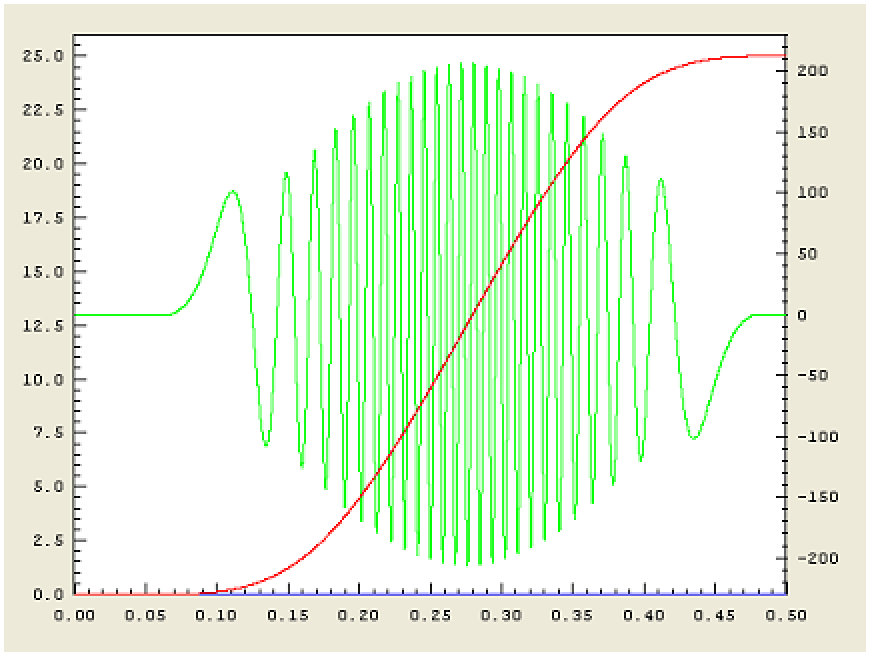
$$lead = \frac{1}{2\sin(\pi T\, f_L)}$$

The Following table generated from the above formula may also be used. For most motors the Lead Compensation values will be within the range of 5 - 20.

Freq, Hz	Lead
50	35.37
60	29.47
70	25.26
80	22.11
90	19.65
100	17.69
120	14.74
140	12.63
160	11.06
180	9.83
200	8.85
220	8.04
240	7.37
260	6.81
280	6.32
300	5.90
350	5.06
400	4.43
450	3.94
500	3.55
550	3.23
600	2.96
650	2.74
700	2.54
750	2.38
800	2.23
850	2.10
900	1.99
950	1.88
1000	1.79



This plot above displays a simple 0.5 second motion with no Lead Compensation for a Microstepper Motor. Position axis shown on the primary (left axis) for the red plot has units of cycles. PWM output shown on the secondary (right axis) for the green plot has units of PWM counts. Move parameters are: Vel=200 cycles/sec, Accel=200 cycles/sec^2, Jerk=10000 cycles/sec3· Note that regardless of velocity PWM amplitude is constant



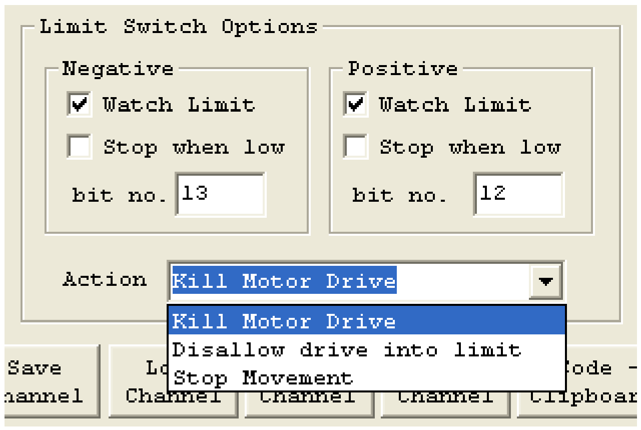
This plot displays the same 0.5 second motion with Lead Compensation = 27.0. All other parameters same as above. Note how PWM amplitude increases with velocity

If motor parameters are unknown, a trial and error approach may be used to find the best lead compensation value. The following procedure may be used:

1. Set Lead Compensation to zero
2. Increase motor speed until a drop in torque is first detected
3. Increase Lead Compensation until normal torque is restored

Setting the Lead Compensation too high should be avoided, as it may cause over current in the motor at medium speeds or voltage distortion due to saturation (clipping).

<u>Limit Switch Options</u>



Limit Switch Options

Negative
[✓] Watch Limit
[ ] Stop when low
bit no. 13

Positive
[✓] Watch Limit
[ ] Stop when low
bit no. 12

Action: Kill Motor Drive

KMotion has the ability to monitor limit switch inputs for each axis and stop motion when a physical limit switch is detected. The limit switch options allow this feature to be enabled or disabled for each limit (positive or negative), what specific bit to be monitored for each limit, what polarity of the bit indicates contact with the limit, and what action to perform when a limit is detected.

Select *Watch Limit* to enable limit switch monitoring.

Select *Stop when low* to select negative true logic for the limit (motion will be stopped when a low level is detected).

Specify a *bit no.* for which bit is to be monitored for the limit condition. See the Digital IO Screen for current I/O bit status and a recommended bit assignment for limit switches (bits 12 through 19). If in a particular application it isn't critical to determine which Limit Switch (either positive or negative, or even which axis) the number of digital I/O bits consumed by limit switches may be reduced by "wire ORing" (connecting in parallel) multiple switches together. In this case, the same bit number may be specified more than one place.

The Action drop down specifies what action should be performed when a limit is encountered.

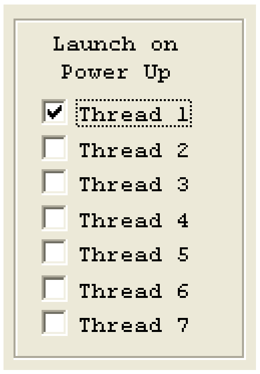
Kill Motor Drive - will completely disable the axis whenever the limit condition is present. Note that it will not be possible to re-enable the axis (and move out of the limit) while the limit condition is still present and this mode remains to be selected.

Disallow drive into limit - will disable the axis whenever the limit condition is present *and* a motion is made into the direction of the limit. This mode will allow the axis to be re-enabled while inside the limit and will allow a move away from the limit.

Stop Movement - this action will keep the axis enabled, but will freeze the commanded position as soon as the limit is detected, and stop any motion trajectory in progress for the axis.

Furthermore, in all the above cases and a limit condition is detected, if the axis belongs to the set of axis in the coordinated group, then any coordinated motion trajectory will be halted.

Launch on Power Up

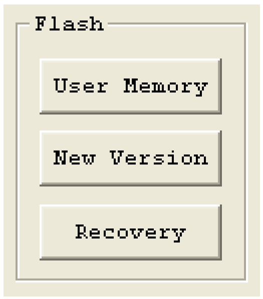


The launch on power up configuration specifies which User Programs are to be automatically launched on power up for stand alone operation of ***KMotion***. See the C Program Screen for information on how to Edit, Compile, and Download a C program into ***KMotion*** for execution into one (or more) of the 7 Thread program spaces within ***KMotion***.

To configure a program execute on power up, perform the following steps:

1. *Compile and Download* a C Program to a particular Thread Space.
2. Select *Launch on Power Up* for the same Thread.
3. *Flash* the User Memory (see following section).
4. *Disconnect* the Host USB cable
5. *Cycle Power* on the ***KMotion***

FLASH

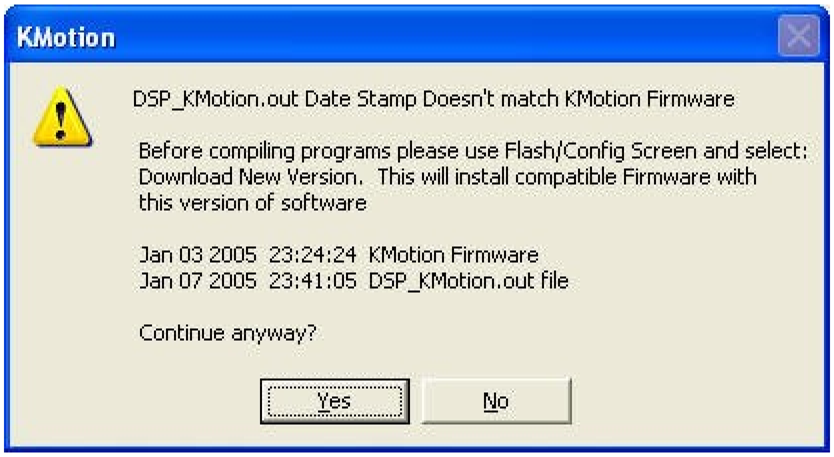


The entire user memory space may be Flashed into nonvolatile memory by depressing the *Flash - User Memory* button. This saves all of the axis configurations, all user program thread spaces, and the user persistent data section. On all subsequent power up resets, ***KMotion*** will revert to that saved configuration. (note that it is preferred to have the host, or a user program, configure the board before each use rather than relying on the exact state of a KMotion set to a particular state at some point in the past).

To upgrade the system firmware in a ***KMotion use*** the *Flash - New Version* button. The user will be prompted to select a **DSPKMotion.out** COFF file from within the ***KMotion*** Install Directory to download and Flash. Note that all user programs and data will be deleted from ***KMotion*** when loading a new version.

After the firmware has been flashed it is necessary to re-boot the ***KMotion*** in order for the new firmware to become active.

It is important that the **<Install Directory>\DSP_KMotion\DSPKMotion.out** file match the firmware that is flashed into ***KMotion***. User C programs are *Linked* using this file to make calls and to access data located within the ***KMotion*** firmware. Whenever a user program is compiled and linked using this file, the timestamp of this file is compared against the timestamp of the executing firmware (if a ***KMotion*** is currently connected). If the timestamps differ, the following message will be displayed, and it is not recommended to continue. The "Version" Console Script Command may also be used to check the firmware version.



DSP_KMotion.out Date Stamp Doesn't match KMotion Firmware

Before compiling programs please use Flash/Config Screen and select:
Download New Version. This will install compatible Firmware with
this version of software

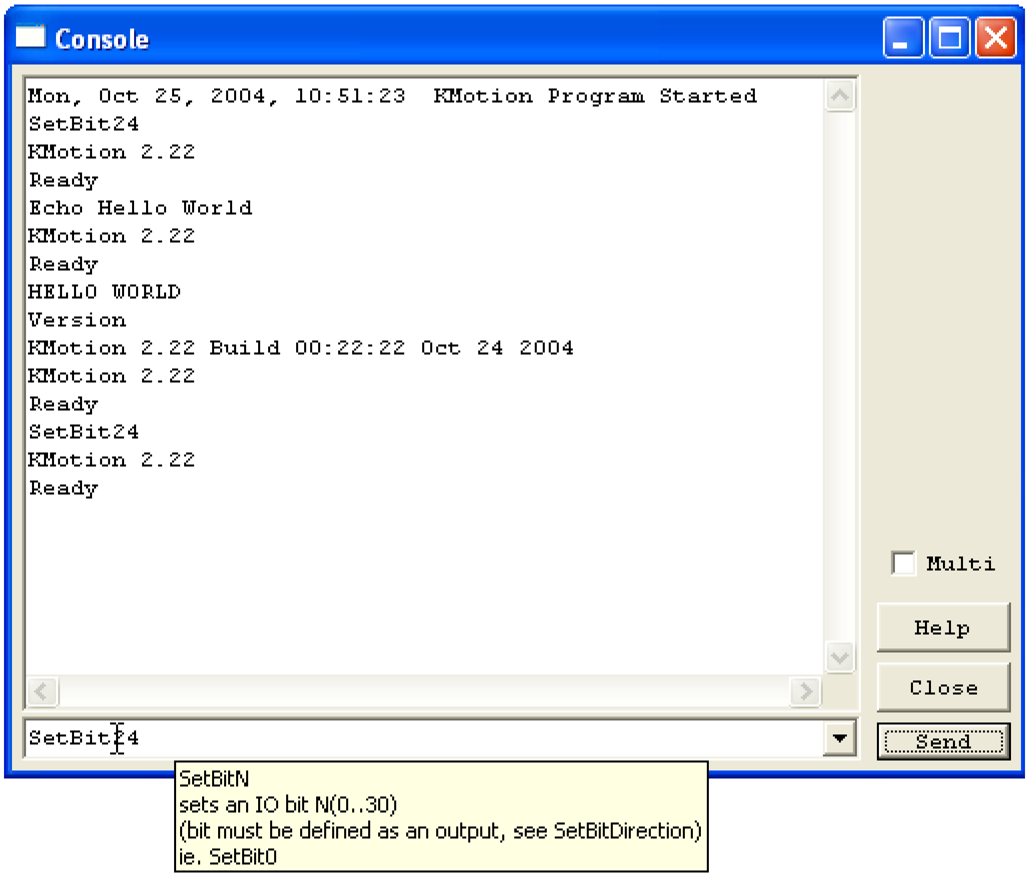
Jan 03 2005 23:24:24 KMotion Firmware
Jan 07 2005 23:41:05 DSP_KMotion.out file

Continue anyway?

In all cases while flashing firmware or user programs the process should not be interrupted or a corrupted flash image may result which renders the board un-bootable. However if this occurs the *Flash Recovery* mode may be used to recover from the situation. To perform the recovery, press the *Flash Recovery* button and follow the dialog prompts to:

1. Select the firmware file to boot
2. Turn off ***KMotion***
3. Turn on KMotion
4. After ***KMotion*** boots, Flash the New Version

Console Screen



```
Mon, Oct 25, 2004, 10:51:23  KMotion Program Started
SetBit24
KMotion 2.22
Ready
Echo Hello World
KMotion 2.22
Ready
HELLO WORLD
Version
KMotion 2.22 Build 00:22:22 Oct 24 2004
KMotion 2.22
Ready
SetBit24
KMotion 2.22
Ready
```

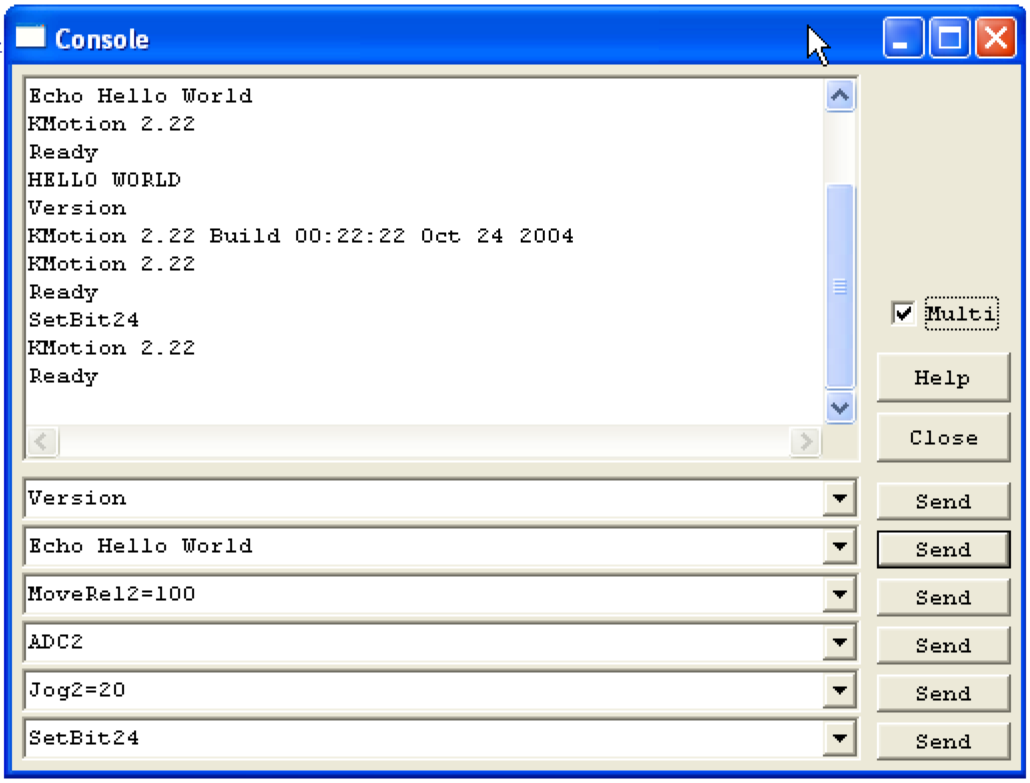
```
SetBit24
```

SetBitN
sets an IO bit N(0..30)
(bit must be defined as an output, see SetBitDirection)
ie. SetBit0

The ***Console Screen*** displays messages from the DSP and the PC. The Console window retains the last 1000 lines of text. After more than 1000 lines are displayed the earliest messages scroll off into a permanent text file (**LogFile.txt**) in the **KMotion\Data** subdirectory.

To Send a command to the DSP enter the text string in the bottom command cell and press the ***Send*** button.

Selecting the ☑ Multi Check box changes from a single command line to multiple command lines, see below. This allows several commands to be entered and then easily sent with a single push button.



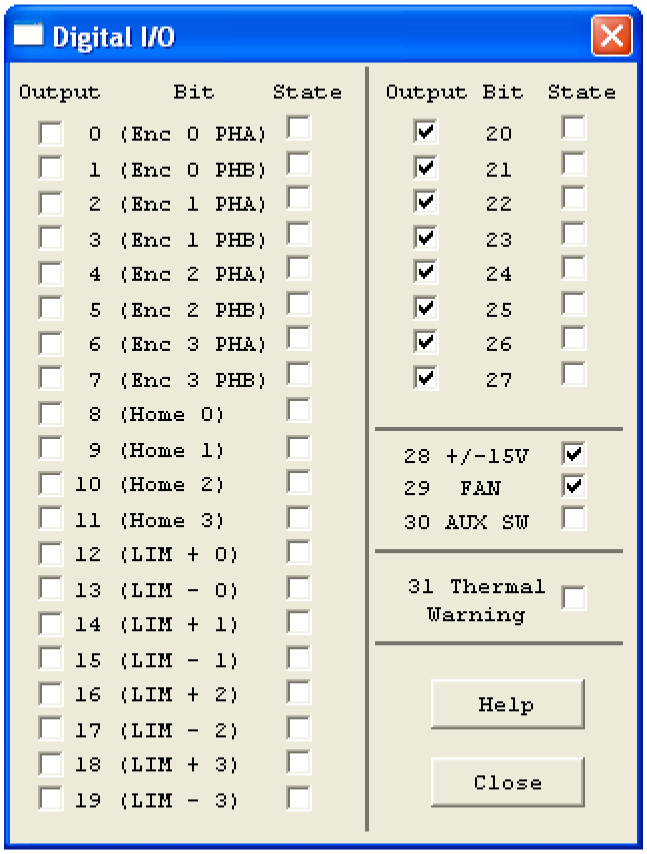
```
Echo Hello World
KMotion 2.22
Ready
HELLO WORLD
Version
KMotion 2.22 Build 00:22:22 Oct 24 2004
KMotion 2.22
Ready
SetBit24
KMotion 2.22
Ready
```

```
Version
Echo Hello World
MoveRel2=100
ADC2
Jog2=20
SetBit24
```

See the alphabetical list for available commands.

Or see commands grouped by category here.

Digital I/O Screen


Digital I/O

Output	Bit	State
☐	0 (Enc 0 PHA)	☐
☐	1 (Enc 0 PHB)	☐
☐	2 (Enc 1 PHA)	☐
☐	3 (Enc 1 PHB)	☐
☐	4 (Enc 2 PHA)	☐
☐	5 (Enc 2 PHB)	☐
☐	6 (Enc 3 PHA)	☐
☐	7 (Enc 3 PHB)	☐
☐	8 (Home 0)	☐
☐	9 (Home 1)	☐
☐	10 (Home 2)	☐
☐	11 (Home 3)	☐
☐	12 (LIM + 0)	☐
☐	13 (LIM - 0)	☐
☐	14 (LIM + 1)	☐
☐	15 (LIM - 1)	☐
☐	16 (LIM + 2)	☐
☐	17 (LIM - 2)	☐
☐	18 (LIM + 3)	☐
☐	19 (LIM - 3)	☐

Output	Bit	State
✓	20	☐
✓	21	☐
✓	22	☐
✓	23	☐
✓	24	☐
✓	25	☐
✓	26	☐
✓	27	☐

28 +/-15V	✓
29 FAN	✓
30 AUX SW	☐
31 Thermal Warning	☐

The ***Digital I/O Screen*** displays and allows changes to the current state of the ***KMotion*** digital I/O bits.

KMotion has a number of digital I/O bits that may be used as *GPIO* (General Purpose Inputs or Outputs) or as specific dedicated functions (encoder inputs). There are 28 bits that may be utilized as GPIO (bits 0 - 27). Each bit may be independently defined as either an input or an output. On Power UP ***KMotion*** defines all I/O as inputs by default. Any bit may be configured as an output by checking the corresponding box in the "Output" columns. Alternately, the bits may be configured by a C program running within the ***KMotion*** (See SetBitDirection()) or by Script commands (See SetBitDirection) sent to the ***KMotion***.

The *State* of each I/O bit may be observed in the corresponding checkbox under the "State" columns. If the bit is defined as an output, clicking on the "State" checkbox will toggle the bit. Alternately, the bits may be read, set, or cleared by a C program running within the ***KMotion*** (See ReadBit(), SetBit(), ClearBit(), or SetStateBit()) or by Script commands (See ReadBit, SetBit, ClearBit, or SetStateBit) sent to the ***KMotion***.

Additionally, *buffered* commands may change the state of Digital I/O bits. *Buffered* I/O commands are I/O commands that are inserted into the coordinated motion buffer. When it is required that I/O bits be changed at exact times within a motion sequence, *buffered* I/O commands may be inserted into the motion buffer (see SetBitBuf, ClearBitBuf, and SetStateBitBuf). In this case the I/O commands occur when they are encountered within the motion sequence. The ***KMotion*** GCode interpreter allows buffered I/O commands to be inserted within motion sequences by using a special form of GCode comment (See buffered GCode Commands). The Digital I/O 0 - 21 are routed to connector JP3, and Digital I/O 20 - 27 are routed to connector JP4 all as LVTTL signals (which are also 5V TTL compliant and tolerant). Note that Digital I/O 20 and 21 are routed to both connectors.

Digital I/O bits 0 - 7 are wired internally to the 4 quadrature encoder inputs (2 bits - Phase A and Phase B for each encoder). For each encoder that is intended to be used, the corresponding Digital I/O bits must be defined as inputs.

Digital I/O bit 28 is dedicated as the +/-15V enable. Setting this bit activates the ***KMotion*** on-board +/-15V generator. The +/- 15V is derived from the 5V supply and is able to source ~ 2 Watts of power on each supply (~ 130ma). This supply is required to be enable when using the +/- 10V ADC inputs or +/-10V DAC outputs.

Caution: Shorting + or - 15V to any Digital I/O bit will cause permanent damage.

Digital I/O bit 29 is dedicated as the *Fan Control* bit. Applications that supply high currents (> ~ 2Amps) for extended periods (> ~10 seconds) on any axis should enable the fan to cool the output stages and avoid thermal shutdown of the output stages.

Digital I/O bit 30 is dedicated as the *Aux Control Switch*. The Aux Control Switch may be used to switch some medium power (< 30V @ 2A) device like a lamp or solenoid (See example).

Digital I/O bit 31 is a dedicated input of *Thermal Warning Status*. Any output stage that is overheating (internal junction temperature > 145 degrees C) will trigger the thermal warning and cause shutdown of all output stages.

G Code Quick Reference

G Codes

G0 X3.5 Y5.0 Z1.0 A2.0 (Rapid move)

G1 X3.5 Y5.0 Z1.0 A2.0(linear move)

G2 X0.0 Y0.5 I0 J0.25 (CW Arc move)

G3 X0.0 Y0.5 I0 J0.25 (CCW Arc move)

G4 P0.25

(Dwell seconds)

G10L2Pn G10L2P1X0Y0Z0

(Set Fixture Offset #n)

G20 Inch units

G21 mm units

G28 Move to Reference Position #1

G30 Move to Reference Position #2

G40 Tool Comp Off

G41 Tool Comp On

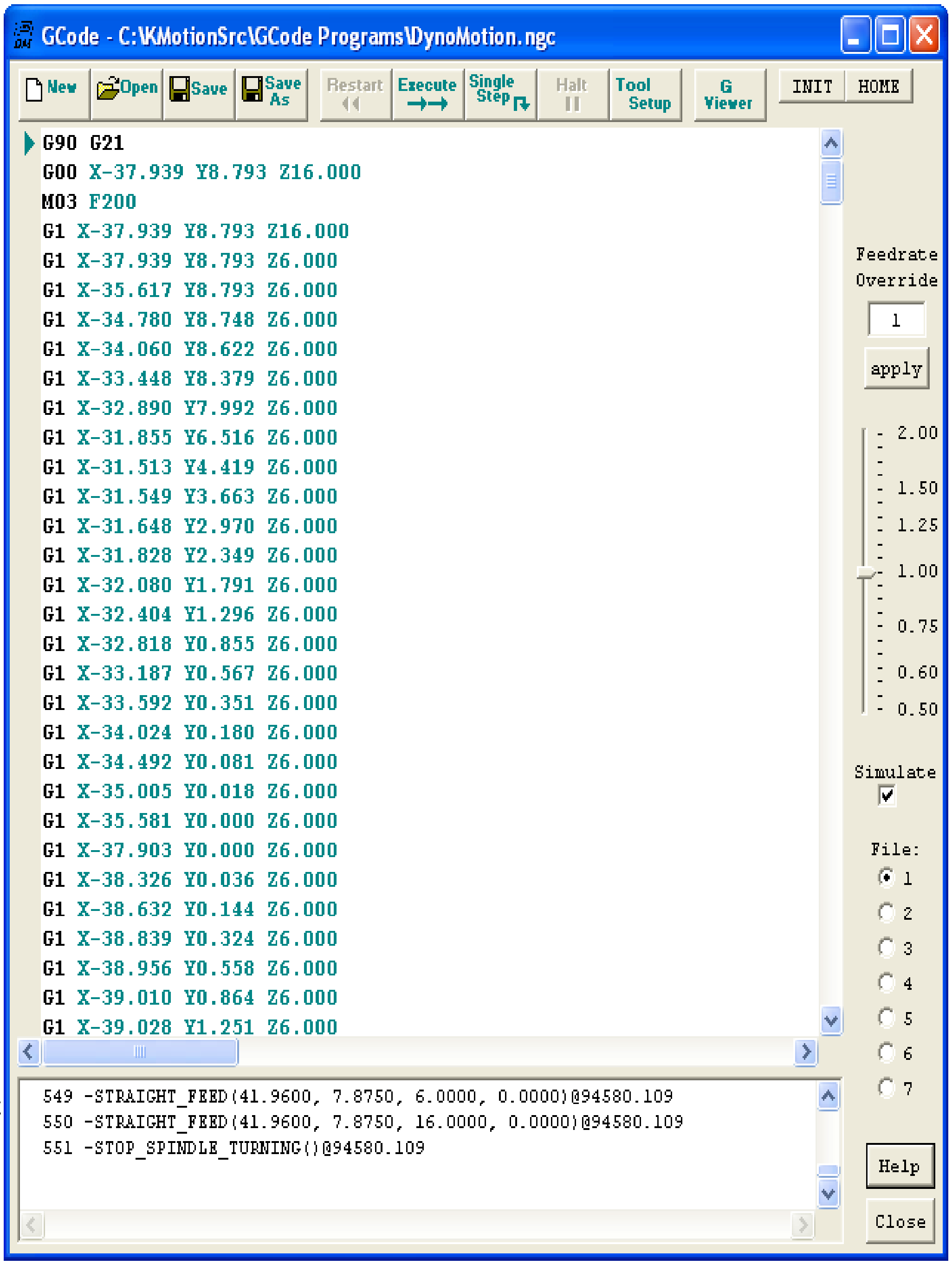
Left of Contour)

G42 Tool Comp On (Right of Contour)

G43 Hn (Tool #n length comp On)

G49 (Tool length comp off)

G Code Screen



Feedrate Override: 1

Simulate: ✓

File: [✓ 1] 2 3 4 5 6 7

```
G90 G21
G00 X-37.939 Y8.793 Z16.000
M03 F200
G1 X-37.939 Y8.793 Z16.000
G1 X-37.939 Y8.793 Z6.000
G1 X-35.617 Y8.793 Z6.000
G1 X-34.780 Y8.748 Z6.000
G1 X-34.060 Y8.622 Z6.000
G1 X-33.448 Y8.379 Z6.000
G1 X-32.890 Y7.992 Z6.000
G1 X-31.855 Y6.516 Z6.000
G1 X-31.513 Y4.419 Z6.000
G1 X-31.549 Y3.663 Z6.000
G1 X-31.648 Y2.970 Z6.000
G1 X-31.828 Y2.349 Z6.000
G1 X-32.080 Y1.791 Z6.000
G1 X-32.404 Y1.296 Z6.000
G1 X-32.818 Y0.855 Z6.000
G1 X-33.187 Y0.567 Z6.000
G1 X-33.592 Y0.351 Z6.000
G1 X-34.024 Y0.180 Z6.000
G1 X-34.492 Y0.081 Z6.000
G1 X-35.005 Y0.018 Z6.000
G1 X-35.581 Y0.000 Z6.000
G1 X-37.903 Y0.000 Z6.000
G1 X-38.326 Y0.036 Z6.000
G1 X-38.632 Y0.144 Z6.000
G1 X-38.839 Y0.324 Z6.000
G1 X-38.956 Y0.558 Z6.000
G1 X-39.010 Y0.864 Z6.000
G1 X-39.028 Y1.251 Z6.000
```

```
549 -STRAIGHT_FEED(41.9600, 7.8750, 6.0000, 0.0000)@94580.109
550 -STRAIGHT_FEED(41.9600, 7.8750, 16.0000, 0.0000)@94580.109
551 -STOP_SPINDLE_TURNING()@94580.109
```

Show screen feature descriptions

G53 Absolute Coord

G54 Fixture Offset 1

G55 Fixture Offset 2

G56 Fixture Offset 3

G57 Fixture Offset 4

G58 Fixture Offset 5

G59 Fixture Offset 6

G59.1 Fixture Offset 7

G59.2 Fixture Offset 8

G59.3 Fixture Offset 9

G90 Absolute Coordinates

G91 Relative Coordinates

G92 Set Global Offset Coordinates G92 X0Y0 Z0

M Codes:

M0 (Program Stop)

M2 (Program End)

M3 Spindle CW

M4 Spindle CCW

M5 Spindle Stop

M6 Tool Change

M7 Mist On

M8 Flood On

M9 Mist/Flood Off

Other Codes:

F (Set Feed rate in/min or mm/min)

S (Spindle Speed)

D (Tool)

See Also G Code Viewer Screen and Tool Setup Screen

The ***G Code Screen*** allows the user to edit G Code Programs and execute them.

GCode is a historical language for defining Linear/Circular/Helical Interpolated Motions often used to program numerically controlled machines (CNC Machines).

See the Quick Reference to the left for commonly used G Code commands.

KMotion's G Code interpreter was derived from the Open Source EMC G Code Interpreter. Click here for the EMC User Manual (Only the G Code portions of the manual, Chapters 10-14 pertain to ***KMotion*** G Code)

Specially coded comments embedded within a GCode program may be used to issue ***KMotion*** Console Script commands directly to ***KMotion***.

A comment in the form: (CMD,xxxxxx) will issue the command xxxxxx immediately to ***KMotion*** as soon as it is encountered by the Interpreter. Any ***KMotion*** command that does not generate a response may be used in this manner.

A comment in the form: (BUF,xxxxxx) will place the command xxxxxx into ***KMotion's*** coordinated motion buffer. Coordinated motion sequences are download to a motion buffer within ***KMotion*** before they are executed. This guarantees smooth uninterrupted motion. The BUF command form allows a command to be inserted within the buffer so they are executed at an exact time within the motion sequence. Only the following ***KMotion*** Script commands may be used in this manner.

SetBitBuf, ClearBitBuf, SetStateBitBuf.

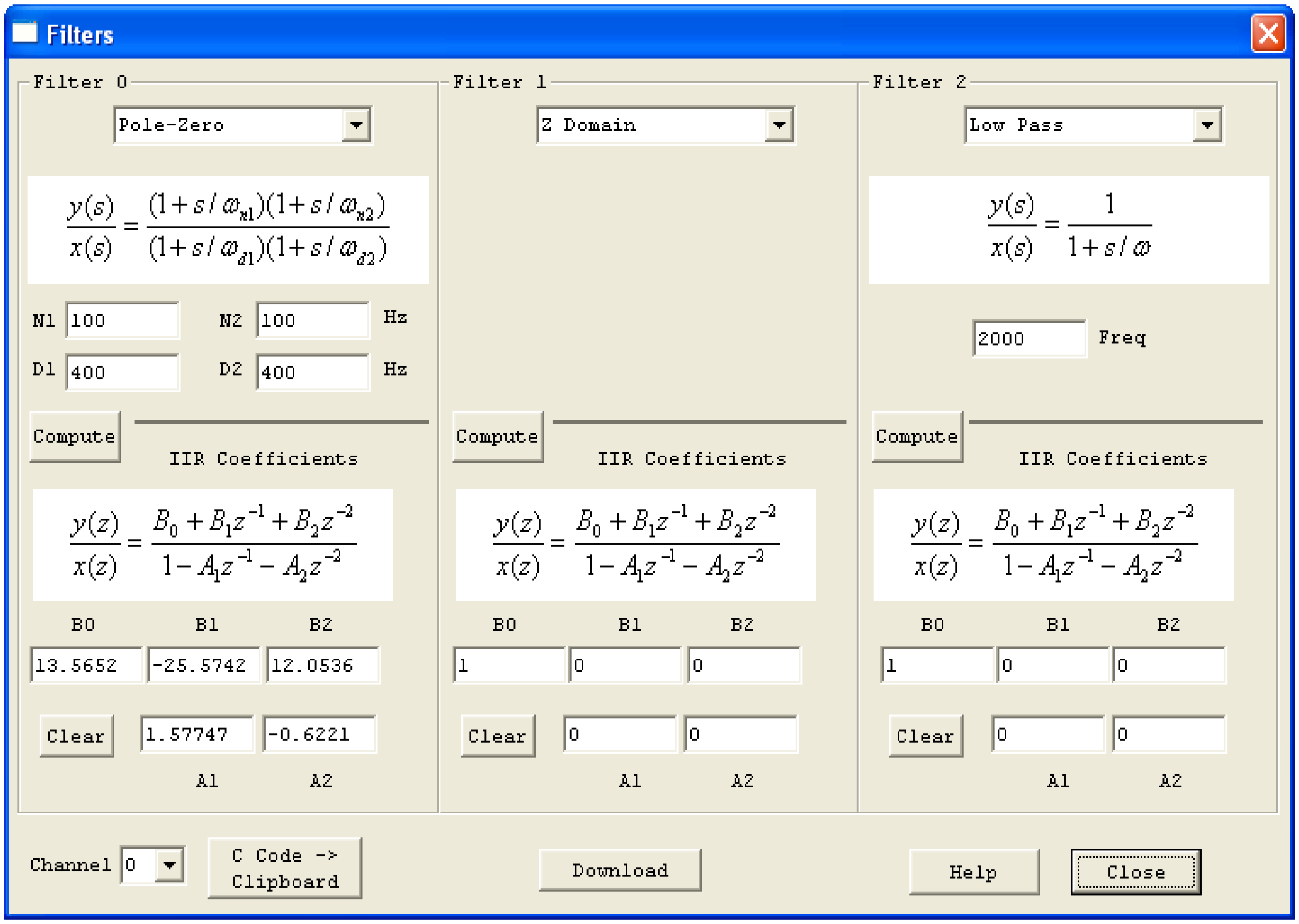
Additionally, a comment in the form: (MSG,xxxxxx) will pause GCode Execution and display a pop-up message window displaying the message xxxxxxx.

Comments:

(Simple Comment)

(MSG,OK toContinue?)
(CMD,EnableAxis0)
(BUF,SetBitBuf29)

IIR Filter Screen



Filter 0: Pole-Zero

$$\frac{y(s)}{x(s)} = \frac{(1+s/\omega_{n1})(1+s/\omega_{n2})}{(1+s/\omega_{d1})(1+s/\omega_{d2})}$$

N1 100 N2 100 Hz

D1 400 D2 400 Hz

$$\frac{y(z)}{x(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}}$$

B0	B1	B2
13.5652	-25.5742	12.0536

A1	A2
1.57747	-0.6221

Filter 1: Z Domain

B0	B1	B2
1	0	0

A1	A2
0	0

Filter 2: Low Pass

$$\frac{y(s)}{x(s)} = \frac{1}{1+s/\omega}$$

2000 Freq

B0	B1	B2
1	0	0

A1	A2
0	0

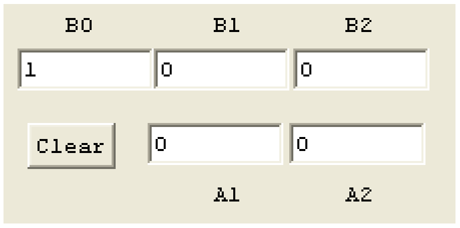
Channel 0

The ***IIR Filter Screen*** allows setting various IIR (Infinite Impulse Response) *Filters* into the control loop. ***KMotion*** allows up to three - 2nd order bi-quadratic stages per axis to be connected in cascade. See the KMotion Servo Flow Diagram for the placement of the IIR Filters.

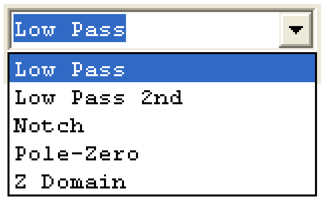
KMotion implements the filters using Z-domain coefficients shown on the bottom half of the screen. Because of the common confusion of the names and signs of the coefficients, the transfer function form is shown for reference.

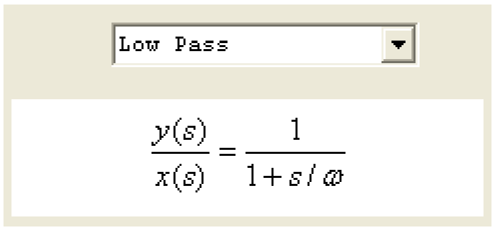
$$\frac{y(z)}{x(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}}$$

Note that setting B0=1.0 and all other coefficients, B1, B2, A1, and A2, to zero causes a transfer function of unity, effectively bypassing the filter. A Clear pushbutton is available to conveniently set this mode.

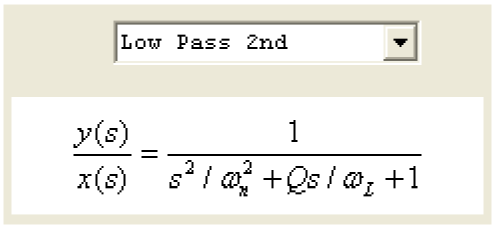


The top portion of each filter section allows various common filters to be specified in the s-domain. Supported filter types are: 1st order Low Pass, 2nd Order Low Pass, Notch, and two real poles and zeros and are selected using a drop-down list box.. Z-domain is a place holder used as a reminder that the z-domain coefficients were determined directly by some other means. The form of each of the filters in the s-domain is shown below.

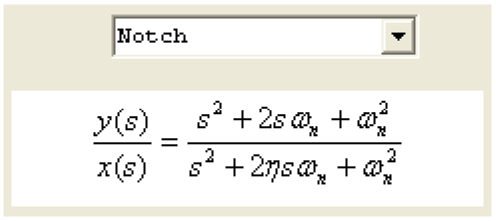




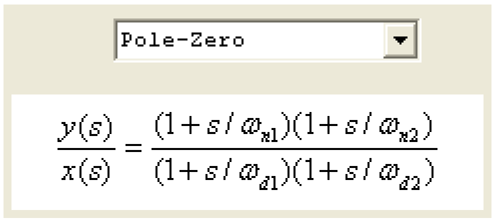
$$\frac{y(s)}{x(s)} = \frac{1}{1+s/\omega}$$



$$\frac{y(s)}{x(s)} = \frac{1}{s^2/\omega_n^2 + Qs/\omega_I + 1}$$



$$\frac{y(s)}{x(s)} = \frac{s^2 + 2s\omega_n + \omega_n^2}{s^2 + 2\eta s\omega_n + \omega_n^2}$$



$$\frac{y(s)}{x(s)} = \frac{(1+s/\omega_{n1})(1+s/\omega_{n2})}{(1+s/\omega_{d1})(1+s/\omega_{d2})}$$

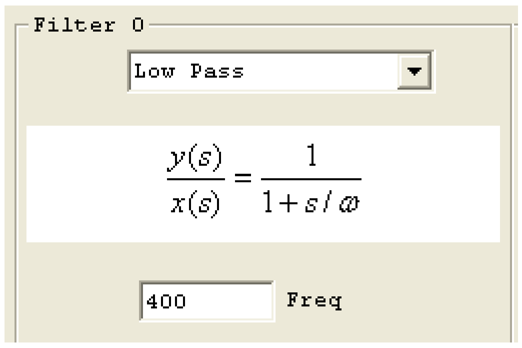
If an s-domain filter type is selected and its corresponding parameters specified, then depressing the *Compute* pushbutton will convert the s-domain transfer function to the z-domain using Tustin's approximation (with frequency pre-warping) and automatically fills in the z-domain coefficients. Note that ***KMotion*** always utilizes the current (most recently computed or entered) z-domain coefficients, regardless of any changes that might be made to the s-domain section.

Note that the Bode Plot Screen has the capability to graph the combined transfer function of all three IIR filters and the PID filter. This is referred to as the *Servo Compensation*. To view the transfer function of a *single* IIR filter, set the other filter and PID sections to unity (for PID set P=1, I=0, D=0 or for an IIR Filter B0=1, B1= B2=A1=A2=0).

Below are examples of each of the s-domain filter types (shown individually):

Low Pass (1st order)

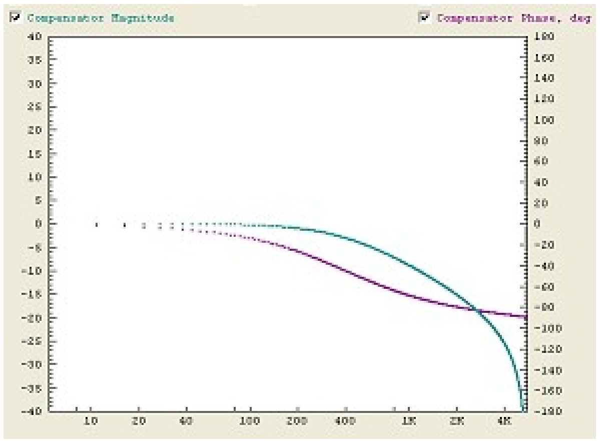
A *Low Pass* filter is commonly used in a servo system to reduce high frequency noise (or spikes) in the output. It also has the desirable effect of decreasing the gain at high frequencies. If a system's gain at high frequencies is increased sufficiently to reach 0 db it may become unstable. Unfortunately it has the effect of introducing phase lag (negative phase) which will reduce the phase margin. A 1st order Low Pass filter has 45 degrees phase and attenuation of -3db at the specified cutoff frequency. The cutoff frequency should normally be specified much higher than the servo bandwidth in order to have only a small phase lag at the system's bandwidth.


Filter 0

Low Pass

$$\frac{y(s)}{x(s)} = \frac{1}{1+s/\omega}$$

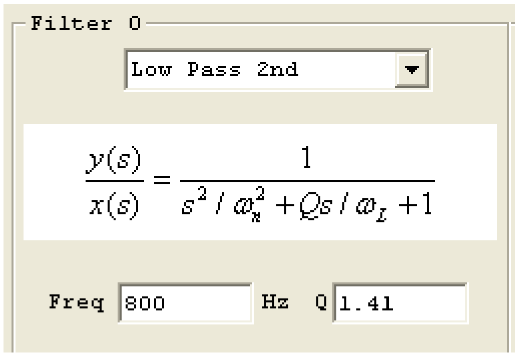
400 Freq



Low Pass (2nd order)

A *2nd order Low Pass* filter is commonly used in a similar manner as a 1st order low pass filter, except that it has higher attenuation than a 1st order low pass filter. Unfortunately it also introduces more phase lag than a 1st order low pass filter. In most cases the cutoff frequency for a 2nd order low pass filter will have to be specified at a higher frequency than a 1st order filter, in order to have similar phase lag at the system bandwidth frequency. Even so, the 2nd order low pass filter is usually preferable in that it provides slightly more high frequency attenuation and "smoothing" of the output.

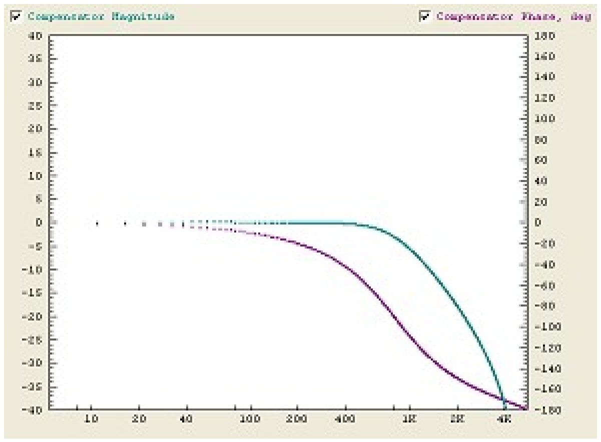
A 2nd order Low Pass filter also allows a Q parameter which changes the sharpness of the filter. A smaller value of Q results in a sharper filter at the expense of some peaking (gain increases before decreasing). A Q value of 1.41 (called a Butterworth filter), shown immediately below, is the minimal value that may be specified without peaking.


Filter 0

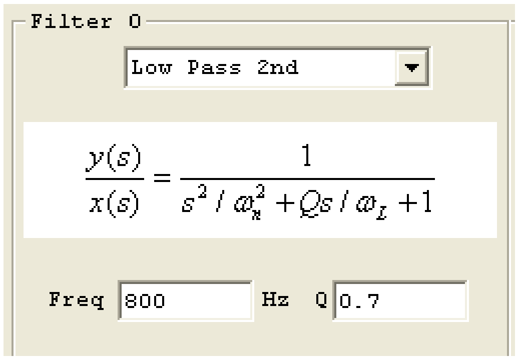
Low Pass 2nd

$$\frac{y(s)}{x(s)} = \frac{1}{s^2/\omega_n^2 + Qs/\omega_I + 1}$$

Freq 800 Hz Q 1.41



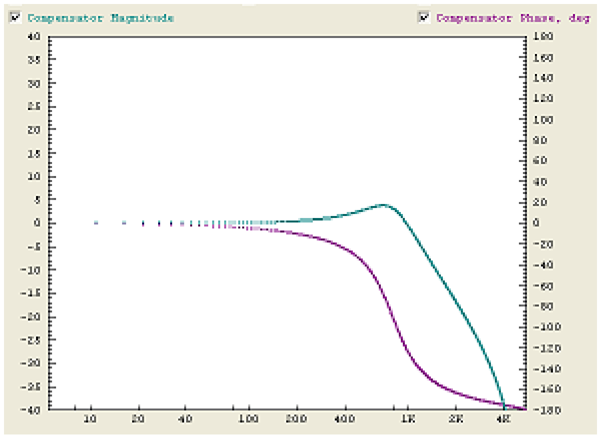
A Q value of 0.7 shows "peaking".


Filter 0

Low Pass 2nd

$$\frac{y(s)}{x(s)} = \frac{1}{s^2/\omega_n^2 + Qs/\omega_I + 1}$$

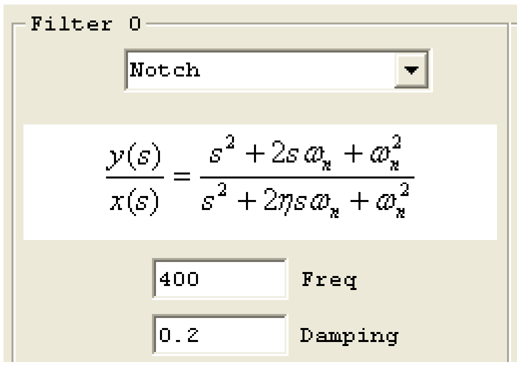
Freq 800 Hz Q 0.7



<u>Notch</u>

A *Notch* filter is commonly used in servo system when a sharp mechanical resonance would otherwise cause a system to become unstable. A Notch filter is able to attenuate a very localized range of frequencies. It has a damping factor, η, which effects sharpness or width of the notch. The disadvantage of using a notch filter is some phase lag which tends to decrease phase margin. The introduced phase lag will be less the narrower the notch is (less damping), as well as the distance the notch frequency is above the system bandwidth.

Shown below are two notch filters both at 400 Hz, one with 0.2 damping and the other with 0.4 damping.

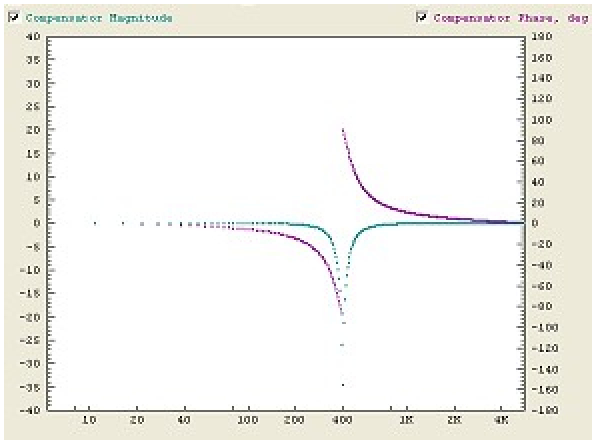


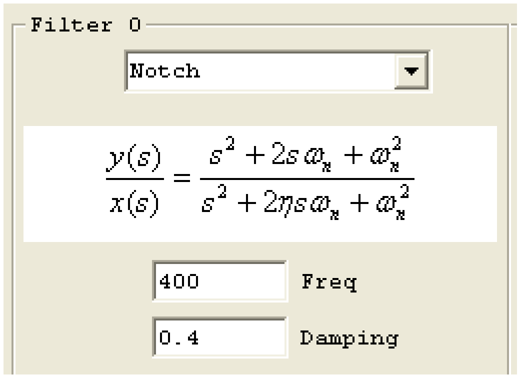
Notch

$$\frac{y(s)}{x(s)} = \frac{s^2 + 2s\omega_n + \omega_n^2}{s^2 + 2\eta s\omega_n + \omega_n^2}$$

400 Freq

0.2 Damping



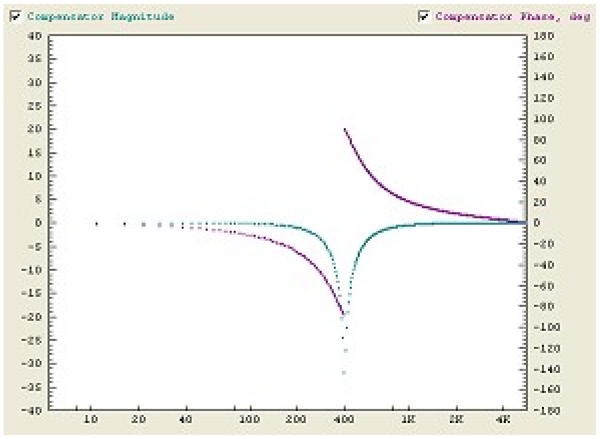


Notch

$$\frac{y(s)}{x(s)} = \frac{s^2 + 2s\omega_n + \omega_n^2}{s^2 + 2\eta s\omega_n + \omega_n^2}$$

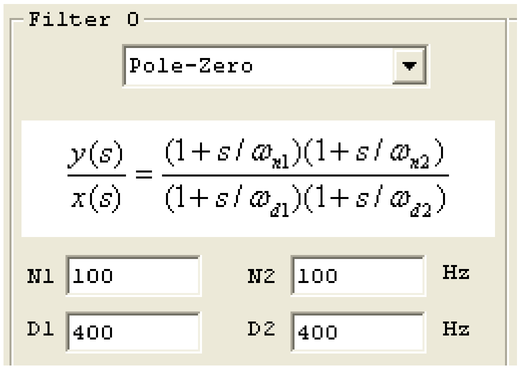
400 Freq

0.4 Damping



Pole-Zero

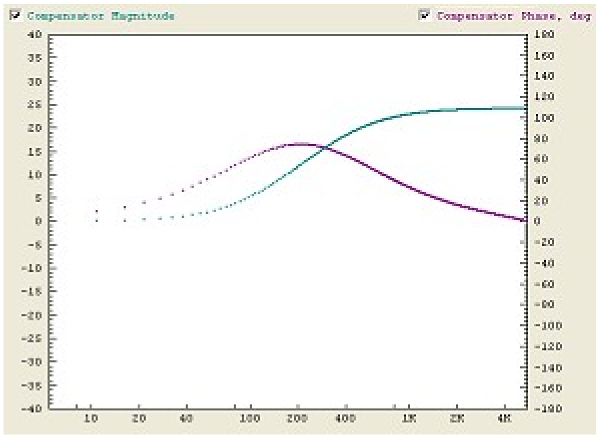
A *Pole-Zero* filter is commonly used in a lead-lag configuration shown below to shift the phase positive at the 0 db crossover frequency of the system in order to increase phase margin. The filter shown has maximum positive phase of approximately 80 degrees at 200 Hz. This is accomplished by setting the N1,N2 (zeros or numerator frequencies) 2X *lower* than 200 Hz (100 Hz), and the D1,D2 (poles or denominator frequencies) 2X *higher* than this (400 Hz). A lead-lag filter (or compensator) may often be used in place of derivative gain (in the PID stage), and has a significant advantage of lower high frequency gain. If a system's gain at high frequencies is increased sufficiently to reach 0 db it may become unstable.



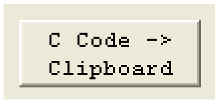
$$\frac{y(s)}{x(s)} = \frac{(1+s/\omega_{n1})(1+s/\omega_{n2})}{(1+s/\omega_{d1})(1+s/\omega_{d2})}$$

N1 100 N2 100 Hz

D1 400 D2 400 Hz



Copy C Code to Clipboard



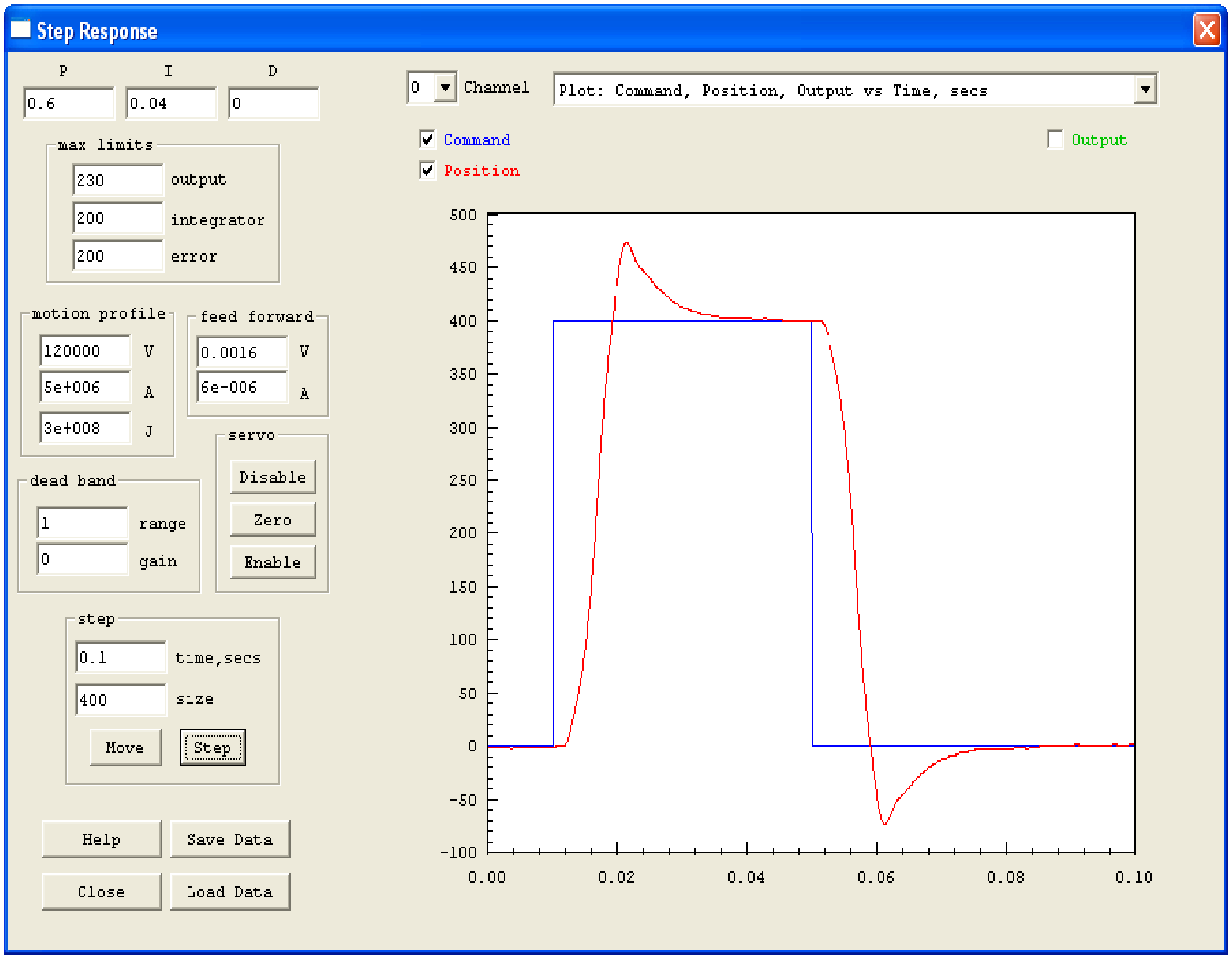
This pushbutton causes the current z-domain filter coefficients to be copied to the clipboard in a form that may be pasted directly into a ***KMotion*** C Program, see also the C Program Screen.

```
ch0->iir[0].B0=232.850006;
ch0->iir[0].B1=-450.471008;
ch0->iir[0].B2=217.869995;
ch0->iir[0].A1=1.001990;
ch0->iir[0].A2=-0.250994;

ch0->iir[1].B0=1.000000;
ch0->iir[1].B1=0.000000;
ch0->iir[1].B2=0.000000;
ch0->iir[1].A1=0.000000;
ch0->iir[1].A2=0.000000;

ch0->iir[2].B0=0.175291;
ch0->iir[2].B1=0.350583;
ch0->iir[2].B2=0.175291;
ch0->iir[2].A1=0.519908;
ch0->iir[2].A2=-0.221073;
```

Download

The *Download* push button downloads the filters for the selected axis (along with all axis configuration and tuning parameters) to the ***Kmotion***.

Step Response Screen


Step Response

P	I	D
0.6	0.04	0

max limits: 230 output; 200 integrator; 200 error

motion profile: 120000 V; 5e+006 A; 3e+008 J

feed forward: 0.0016 V; 6e-006 A

dead band: 1 range; 0 gain

step: 0.1 time,secs; 400 size

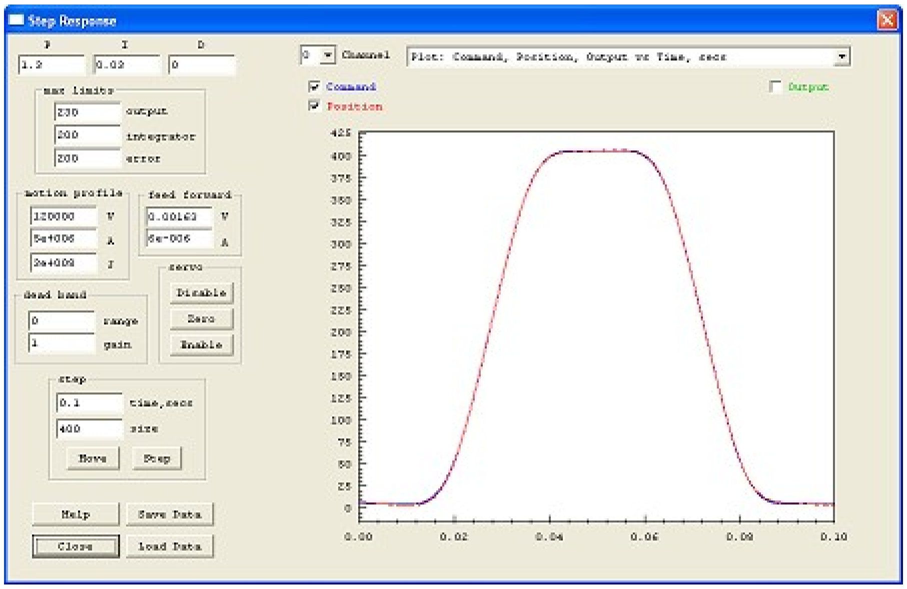
Channel 0

Plot: Command, Position, Output vs Time, secs

[✓] Command
[✓] Position
[ ] Output

(Click on above image to jump to relative topic)

The ***Step Response Screen*** allows changes to system tuning parameters and allows measurement and graphs of the system's time response for either a Move Profile or a Step Function. The graph shown above is of an applied step function of 400 counts. The graphs shown below are of a profiled move (and back) of 400 counts. The first has the Output drive hidden and the second has the Output drive displayed. Click on the graphs for a larger view. Note that the Output drive signal contains large spikes. This is the result of quantization error in the measured position. Quantization error in the measured position makes it appear to the system as if there was no motion, and then suddenly as if there was a relatively quick motion of one count in a single servo sample cycle. This is a non-linear effect. In some cases these "spikes" may exceed the output range causing saturation a still further non-linear effect. A low pass filter may be used to "smooth" the output, see the IIR Filter Screen, but has limits. Attempting too much "smoothing" by setting a lower frequency will eventually have an effect on the performance of the system, reducing the phase margin. Normally, the cutoff frequency of the low pass filter should be significantly larger than the system bandwidth.


Step Response

P	I	D
1.2	0.02	0

max limits: 230 output, 200 integrator, 200 error

motion profile: 120000 V, 5e+006 A, 2e+008 J

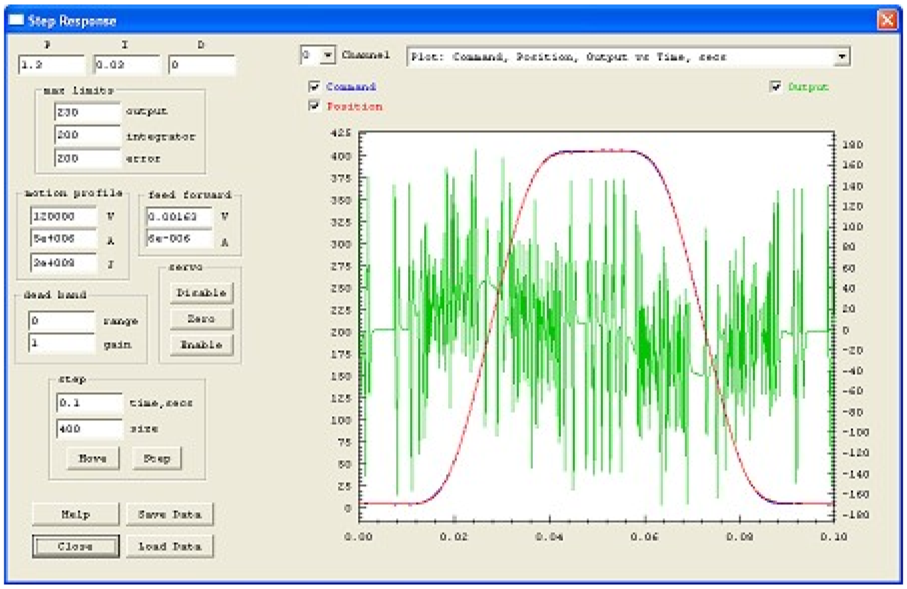
feed forward: 0.00163 V, 6e-006 A

dead band: 0 range, 1 gain

step: 0.1 time,secs, 400 size

0 Channel — Plot: Command, Position, Output vs Time, secs

[✓] Command [✓] Position [ ] Output


Step Response

P	I	D
1.2	0.02	0

max limits: 230 output, 200 integrator, 200 error

motion profile: 120000 V, 5e+006 A, 2e+008 J

feed forward: 0.00163 V, 6e-006 A

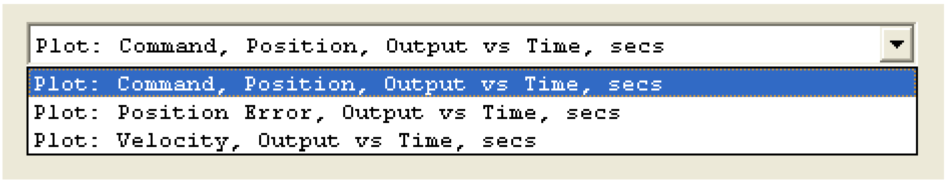
dead band: 0 range, 1 gain

step: 0.1 time,secs, 400 size

0 Channel — Plot: Command, Position, Output vs Time, secs

[✓] Command [✓] Position [✓] Output

There are three basic time domain plot types that may be selected from the drop down list, which are shown below.


```
Plot: Command, Position, Output vs Time, secs
Plot: Command, Position, Output vs Time, secs
Plot: Position Error, Output vs Time, secs
Plot: Velocity, Output vs Time, secs
```

They consist of either:

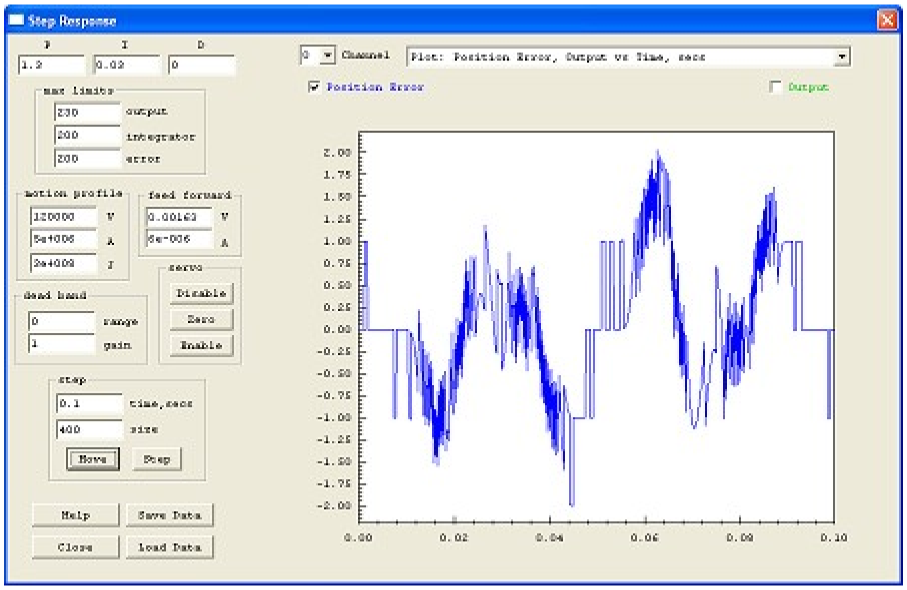
1. Commanded Position, Measured Position, and Motor Output
2. Position Error and Motor Output

3. Commanded Velocity, Measured Velocity, and Motor Output

For all three plot types the Motor Output is always displayed as a secondary *Y* axis on the right side of the graph. The other plotted values are on the primary left *Y* axis. The *X* axis is always time in seconds. After a particular plot type has been selected, each individual plot variable may be displayed or hidden by selecting the checkbox with the corresponding name (and color) of the variable.

Any portion of the graph may be zoomed by left-click dragging across the graph. Simply select the area of interest. Right clicking on the graph will bring up a context menu that allows zooming out completely or to the previous zoom level.

Below is an example of a graph of *Position Error* (for the same 400 count move shown above). Position Error is defined as Measured Position - Commanded Position. The same data as that is plotted in the Command/Position plots is used, however instead of plotting both values, the difference is plotted. Note that because the Measured Position is quantized to integer encoder counts, the quantization effect is also observed in the Position Error.


Step Response

P 1.2 I 0.02 D 0

max limits: 230 output, 200 integrator, 200 error

motion profile: 120000 V, 5e+006 A, 2e+008 J

feed forward: 0.00162 V, 6e-006 A

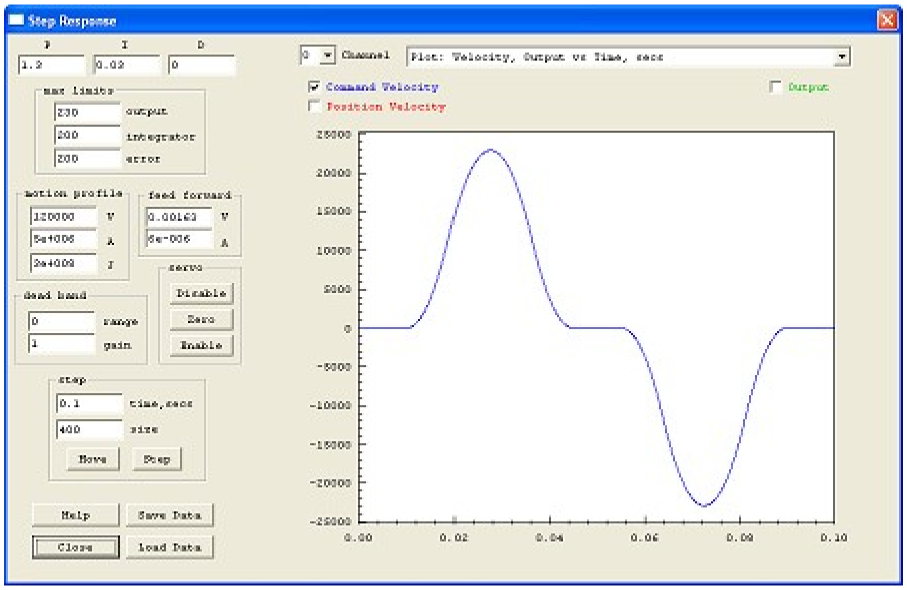
dead band: 0 range, 1 gain

step: 0.1 time,secs, 400 size

0 Channel — Plot: Position Error, Output vs Time, secs

☑ Position Error ☐ Output

The third type of plot displays the Velocity of the Commanded and/or Measured Position. Velocity units are *Position Units* per second. When a Move is commanded, a motion profile is computed which achieves the motion in the shortest time without exceeding the maximum allowed velocity, acceleration, or jerk. Because the Command is a theoretical profile computed using floating point arithmetic, it is very smooth. The blue graph immediately below shows such a plot. In a velocity graph, *slope* in the graph represents acceleration. In this case a relatively low value specified for maximum jerk causes the changes in slope to be gradual. The second plot below is the same data but with the *Measured* velocity displayed along with the Commanded velocity. Because of encoder resolution limitations, measured velocity calculated using a simple position difference per sample period tends to be highly quantized as shown. In this example even at our peak velocity at ~ 23,000 position counts per second this results in a maximum of only 3 position counts per servo sample period.



Step Response

P 1.2 I 0.02 D 0

max limits: 230 output, 200 integrator, 200 error

motion profile: 120000 V, 5e+006 A, 2e+008 J

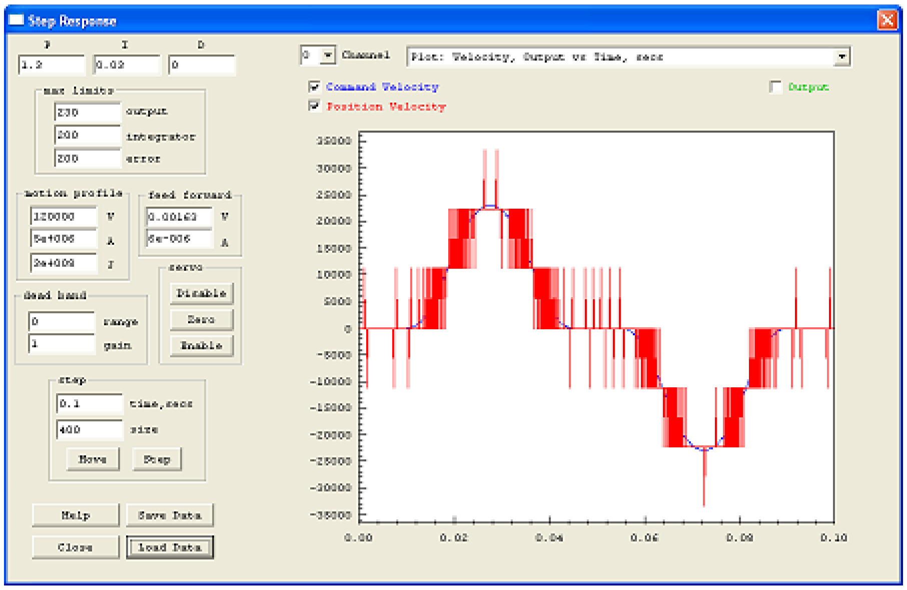
feed forward: 0.00162 V, 5e-006 A

dead band: 0 range, 1 gain

step: 0.1 time,secs, 400 size

0 Channel Plot: Velocity, Output vs Time, secs

[✓] Command Velocity
[ ] Position Velocity
[ ] Output



Step Response

P 1.2 I 0.02 D 0

max limits: 230 output, 200 integrator, 200 error

motion profile: 120000 V, 5e+006 A, 2e+008 J

feed forward: 0.00162 V, 5e-006 A

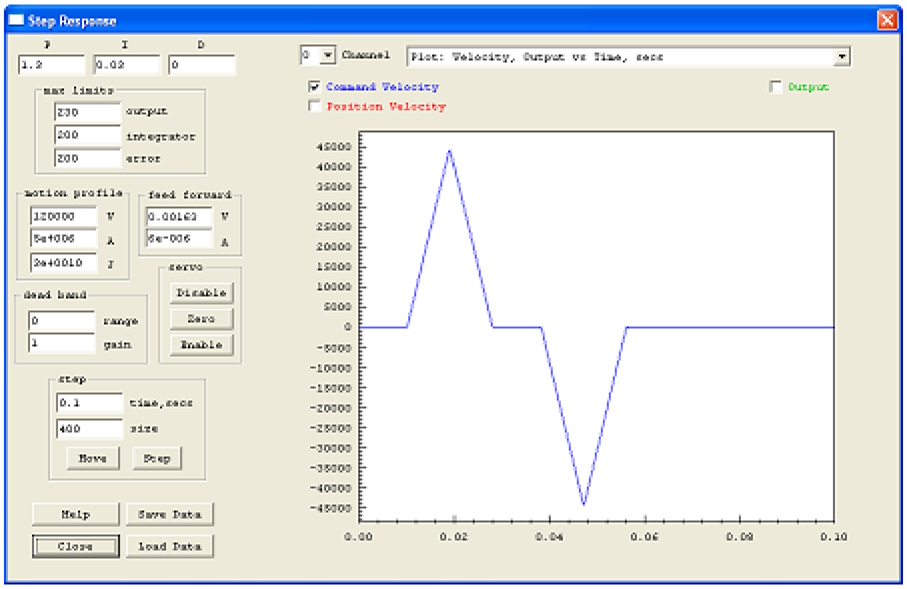
dead band: 0 range, 1 gain

step: 0.1 time,secs, 400 size

0 Channel Plot: Velocity, Output vs Time, secs

[✓] Command Velocity
[✓] Position Velocity
[ ] Output

The velocity graph below, shows the effect of setting the maximum allowed jerk to a very large value (100X higher than the graph above). Note how the slope of the velocity changes abruptly which represents a high rate of change of acceleration (jerk).


Step Response

P 1.2 I 0.02 D 0

max limits: 230 output, 200 integrator, 200 error

motion profile: 120000 V, 5e+006 A, 2e+0010 J

feed forward: 0.00163 V, 6e-006 A

dead band: 0 range, 1 gain

step: 0.1 time,secs, 400 size

0 Channel

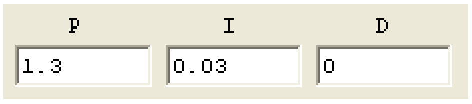
Plot: Velocity, Output vs Time, secs

[✓] Command Velocity

[ ] Position Velocity

[ ] Output

Tuning Parameters - PID


P 1.3 I 0.03 D 0

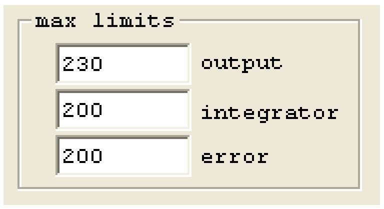
The PID (proportional, integral, and derivative) gains set the amount of feedback of the error itself (proportional), the integration of the error (integral), and the derivative of the position (derivative) that is applied to the output. Also see the KMotion Servo Flow Diagram.

The units of the proportional gain are in Output Units/Position Units. For example if the Position Units are in encoder counts, and the Output Units are in PWM counts, then a gain of 10.0 would apply an output drive of 10 PWM for an error of 1 encoder count.

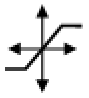
The units of the integral gain are in Output Units/Position Units per *Servo Sample Time*. ***KMotion's*** *Servo Sample Time* is fixed at 90µs. An integrator basically sums the position error every servo sample. For example, with an integral gain of 10, and an error of 1 encoder count for 5 servo samples, an output drive of 50 PWM counts would be applied Integrator gain is normally used to achieve high accuracy. This is because even a very small error will eventually integrate to a large enough value for there to be an corrective action. In fact, having any integrator gain at all guarantees a steady state error (average error) of zero. This effect also guarantees that there will always be some overshoot in response to a step function, otherwise the average error could not be equal to zero.

The units of the derivative gain are in Output Units/Position Units x *Servo Sample Time*. The derivative term is simply the change in position from one servo sample to the next. For example, with a derivative gain of 10, and a position change of 1 encoder count from the previous servo sample, an output drive of -10 PWM counts would be applied. The negative sign shows that the output is applied in a manner to oppose motion. Derivative gain has the effect of applying damping, which is a force proportional and opposite to the current velocity. Although derivative gain is often used successfully in a control system, consider using a lead/lag filter which performs in a similar manner, but doesn't have the undesirable feature of increasing gain at high frequencies.

Tuning Parameters - max limits


max limits

230	output
200	integrator
200	error

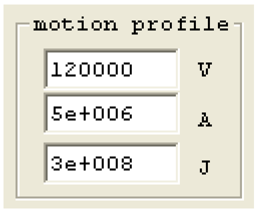
KMotion's *max limits* allow several points in the Servo Flow Diagram to be clamped, or limited to a specified range. The limits in the flow diagram are shown as a clamp symbol  . This capability is often useful in controlling how the system responds to extreme situations.

Maximum *output* limit is used to limit the maximum applied value, in counts, to the output drive. The output drive may be either one of the on-board PWM outputs or a DAC value that drives an external amplifier.

Maximum *integrator* limit is used to restrict the maximum value of the integrator. This effect is often used to avoid an effect referred to as integrator "*wind up*". Without any integrator limit, consider the case where somehow a substantial error is maintained for a significant period of time. For example turning a motor shaft by hand for several seconds. During this time the integrator would ramp up to an extremely large value. When the motor shaft was released, it would accelerate at maximum and overshoot the target by a huge amount until the integrator could ramp back down to a reasonable value. This often results in a servo slamming into a limit. The maximum integrator limit prevents this from occurring. Often the main purpose for using an integrator is to overcome static friction in order to reduce the final error to zero. This usually requires only a small fraction of total output range. In almost all cases it is of no value to allow the integrator to exceed the maximum output value.

Maximum *error* limits the maximum value allowed to pass through the servo compensator. The units are the same as position units. Typically, when a servo loop is operating normally, its following error is a small value. When some extreme even occurs, such as a sudden large step command, or possibly a large disturbance the error may become very large. In some cases there may be benefit to limiting the error to a reasonable value.

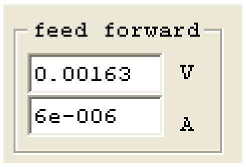
Tuning Parameters - Motion Profile


motion profile

120000	V
5e+006	A
3e+008	J

The Motion Profile parameters set the maximum allowed velocity (in position units per second), the maximum allowed acceleration (in position units per second^2), and the maximum allowed jerk (in position units per second^3). These parameters will be utilized for any independent (non coordinated motion) move command for the axis. The acceleration and jerk also apply to jog commands (move at continuous velocity) for the axis.

Tuning Parameters - Feed Forward


feed forward

0.00163	V
6e-006	A

KMotion's *Feed Forward* may often be used to dramatically reduce the following error in a system. See the Servo Flow Diagram to see precisely how it is implemented. The idea behind feed forward is to observe the velocity and acceleration of the command signal and anticipate a required output and to apply it without waiting for an error to develop.

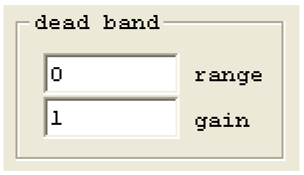
Most motion systems are constructed in manner where some sort of motor force is used to accelerate a mass. In these cases whenever an acceleration is required a force proportional to the acceleration will be required to achieve it. Acceleration feed forward may be used to reduce the amount that the feedback loop must correct. In fact, proper feed forward reduces the requirement on the feedback from the total force required to accelerate the mass, to only the variation in the force required to accelerate the mass.

Similarly most servo systems require some amount of force that is proportional to velocity simply to maintain a constant velocity. This might be due to viscous friction, or possibly motor back emf (electro motive force). In any case velocity feed forward may be used to reduce the demands of the feedback loop resulting in smaller following error.

The normal procedure to optimize feed forward is to select plot type - position error, and measure moves using the *Move Command* (Step functions should *not* be used as step functions are instantaneous changes in position that represent infinite velocity and acceleration).

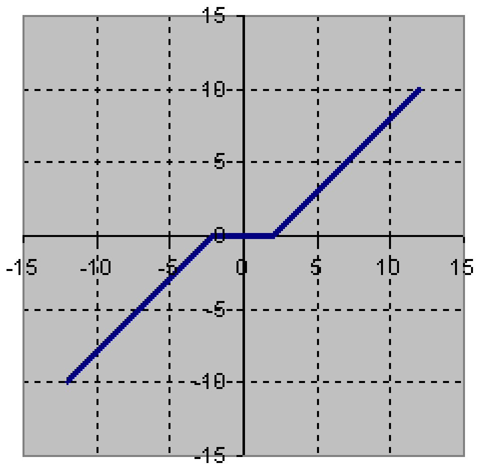
Note that in the Servo Flow Diagram the feed forward is injected *before* the final IIR Filter. This allows any feed forward waveforms to be conditioned by this filter. Feed forward pulses may be relatively sharp pulses to make rapid accelerations that may often tend to disturb a mechanical resonance in the system. Usually a system with a sharp resonance will benefit from a notch filter to improve the stability and performance of the servo loop. By placing the notch filter as the last filter in the servo loop, the feed forward waveform will also pass through this filter and the result is that the feed forward will cause less excitation of the mechanism than it would otherwise..

Tuning Parameters - Dead Band



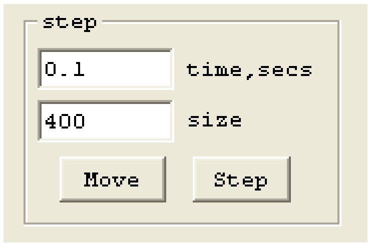
0	range
1	gain

Dead band is used to apply a different gain to the region near zero than the rest of the region. Usually either zero gain or a gain much less than 1 is used within the dead band range. See the Servo Flow Diagram for the exact location of where the dead band is inserted. Dead band is a means of introducing "slop" into a system. This usually results in less accuracy and performance, but may reduce or eliminate limit cycle oscillations while resting at the target position.



The values shown (range = 0, gain = 1) are used to defeat any dead band. The chart shows the resulting input/output for range = 2, gain = 0. The slope of the graph is always 1 outside of the specified +/- range, and the specified gain with the +/- range.

Measurement



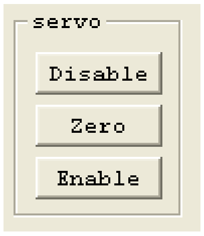
To perform a measurement and display the response, select the time duration to gather data, and the move or step size to perform, and press either the Move or Step buttons. If the axis is currently enabled, it will be disabled, all parameters from all screens will be downloaded, the axis will be enabled, the move or step will be performed while the data is gathered, the data will then be uploaded and plotted.

A *Move* will hold position for a short time, perform a motion of the specified amount from the current location, pause for a short time, and then a second motion back to the original location.

A *Step* will hold position for a short time, perform a step of the specified amount from the current location, pause for a short time, and then a second step back to the original location.

The maximum time that data may be collected is 3.5 seconds (3.5 seconds / 90µs = 38,888 data points). Note that collecting data at this rate allows zooming while still maintaining high resolution.

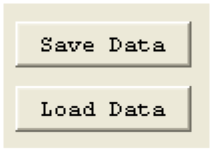
Axis Control



The Axis Control buttons are present to conveniently disable (Kill), Zero, or Enable an axis. If the axis becomes unstable (possible due to a gain setting too high), the Kill button may be used to disable the axis, the gain might then be reduced, and then the axis may be enabled. The Enable button downloads all parameters from all screens before enabling the axis in the same manner as the Measurement buttons described above.

Note for brushless output modes that commutate the motor based on the current position, Zeroing the position may adversely affect the commutation.

Save/Load Data



The Save/Load Data buttons allow the captured Step Plot to be saved to a text file and re-loaded at a later time. The text file format also allows the data to be imported into some other program for display or analysis. The file format consists of one line of header followed by one line of 5 comma separated values, one line for each sample. The values are:

1. Sample Number
2. Time, Seconds
3. Command
4. Position
5. Output

Example of data file follows:

```
Sample,Time,Command,Position,Output
0,0,5,5,-0.3301919
1,9e-005,5,5,-0.3300979
2,0.00018,5,5,-0.3300258
3,0.00027,5,5,-0.3299877
4,0.00036,5,5,-0.3299999
5,0.00045,5,5,-0.3300253
6,0.00054,5,5,-0.3300359
7,0.00063,5,5,-0.3300304
8,0.00072,5,5,-0.3300199
9,0.00081,5,5,-0.3300156
10,0.0009,5,5,-0.3300157
```

62

-
-
-

Four Axis, DSP/FPGA-based Motion Controller

DynoMotion's KMotion card combines a DSP, FPGA, Output stage, USB, and a PC-based development environment to create a versatile and programmable single-board motion solution. Designed for four-axis control, the KMotion provides advanced control for torque, speed and position applications for any mix of stepper, DC brushless, and DC brush motors. The integrated output stages allows ultra-smooth stepper operation and high-performance DC operation while reducing space, cabling, interface headaches, and board count. KMotion uses flash memory to store and run multiple-thread compiled C code on a 600 MFLOP processor with native 64-bit floating point support for stand-alone operation. A PC connected with a USB cable can be used for control and monitoring.

The included PC-based integrated development environment combines configuration, status, programming, and advanced diagnostic and tuning tools such as Bode plots and signal filtering. GCode support allows coordinated moves between axes. Libraries for controlling the KMotion card via Visual C++ and Visual Basic are included, as well as a free C compiler. Thread-safe operation allows the IDE to be used in conjunction with a user application for control and debugging.

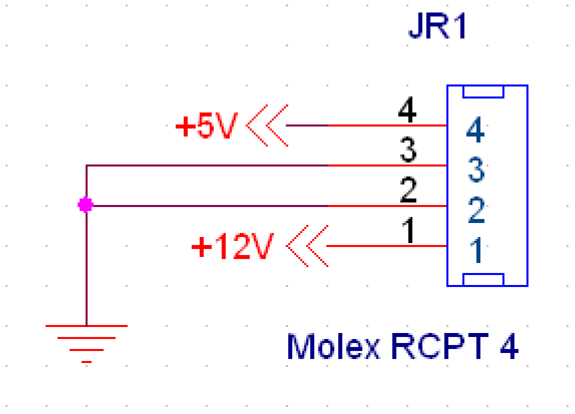
The KMotion packs a lot of IO into its 5.5 x 6.5 x 2.0 in package. Eight Full Bridges are controlled by 30 KHz, 10 bit PWMs. Four 165 watt onboard current-limited power amps provide 3A steady, 6A Peak current at 55V. The KMotion utilizes 28 Bi-directional I/O bits, shared between predefined limit switches and user-defined I/O. In addition, there are four channels of analog input (+/- 10V) and 8 channels of analog output (4@+/-10V, 4@ 0-4V).

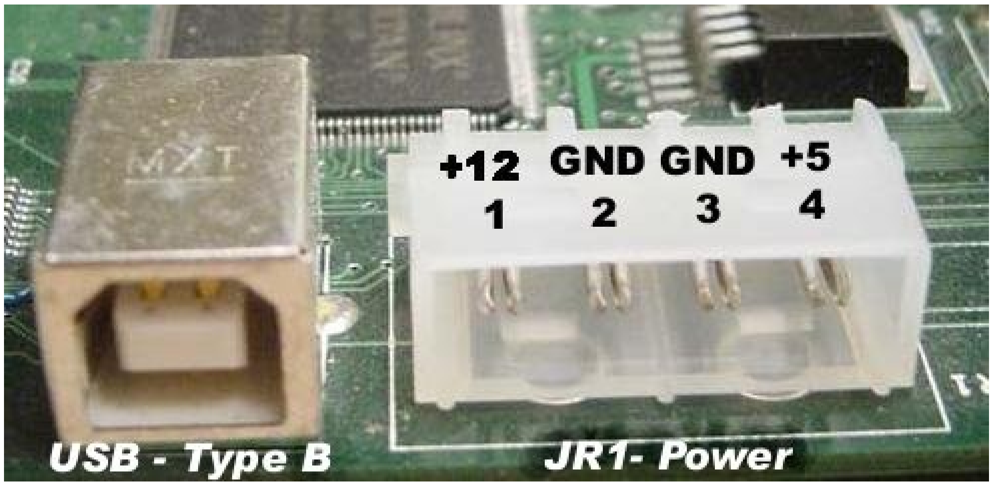
www.DynoMotion.com, Calabasas, CA. [sales@dynomotion.com]

JR1 - 5V Power (regulated +/- 5%)

Typical current = 0.7Amps with no user I/O connected. More current may be required dependent on the amount of Digital I/O and Analog +/- 15V consumed by the user. Up to 2 watts of +/-15 volts is generated on board from this 5V supply (most of which is available for external use). 5V @ 2.5A should be more than sufficient under all conditions. The 12V input is not used internally by the board, but is routed to pins on the 37pin DB Motor connector and the 16 pin Aux Connector for the convenience of the user. 5V power is also routed to the 37pin DB Motor connector, 5V power may be applied at whichever connector is more convenient. This connector is only rated for 6.5Amps per connection, if more total motor current is required, power should be externally routed to the 37pin DB Supply 0-3 inputs directly.

This connector is a standard PC-disk drive power connector which makes it easy to drive the board and small to medium size motors (< 12V) with an inexpensive PC power supply and very few external connections. Simply plug the PC power supply here, jumper the +12V signals on the DB37 Pin connector to the desired motor supply inputs, and connect your motors.





JDP1 - Motor/Motor Supply (12-55V) Connector

The ***KMotion*** motion control board is basically a 4 axis Motor controller that consists of 8 full bridge drivers (see figure 1). A full bridge driver is able to apply a positive or negative voltage to a load using only a single positive supply. The load is connected across the OUTA and OUTB terminals. The 8 full bridge drivers are grouped into 4 pairs, where a pair of full bridge drivers are associated with a motor axis. This is because some types of motors (stepper motors or 3-phase motors require more than a single full bridge to drive them).

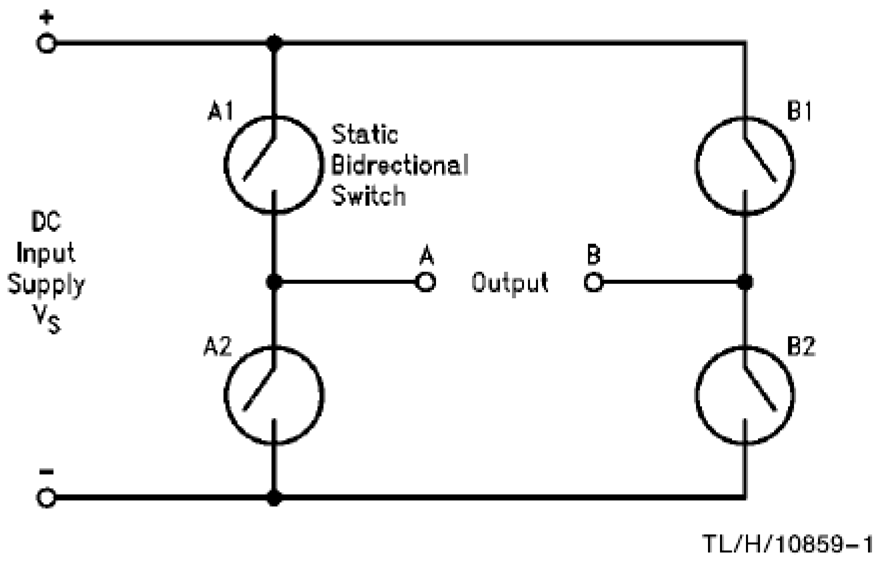
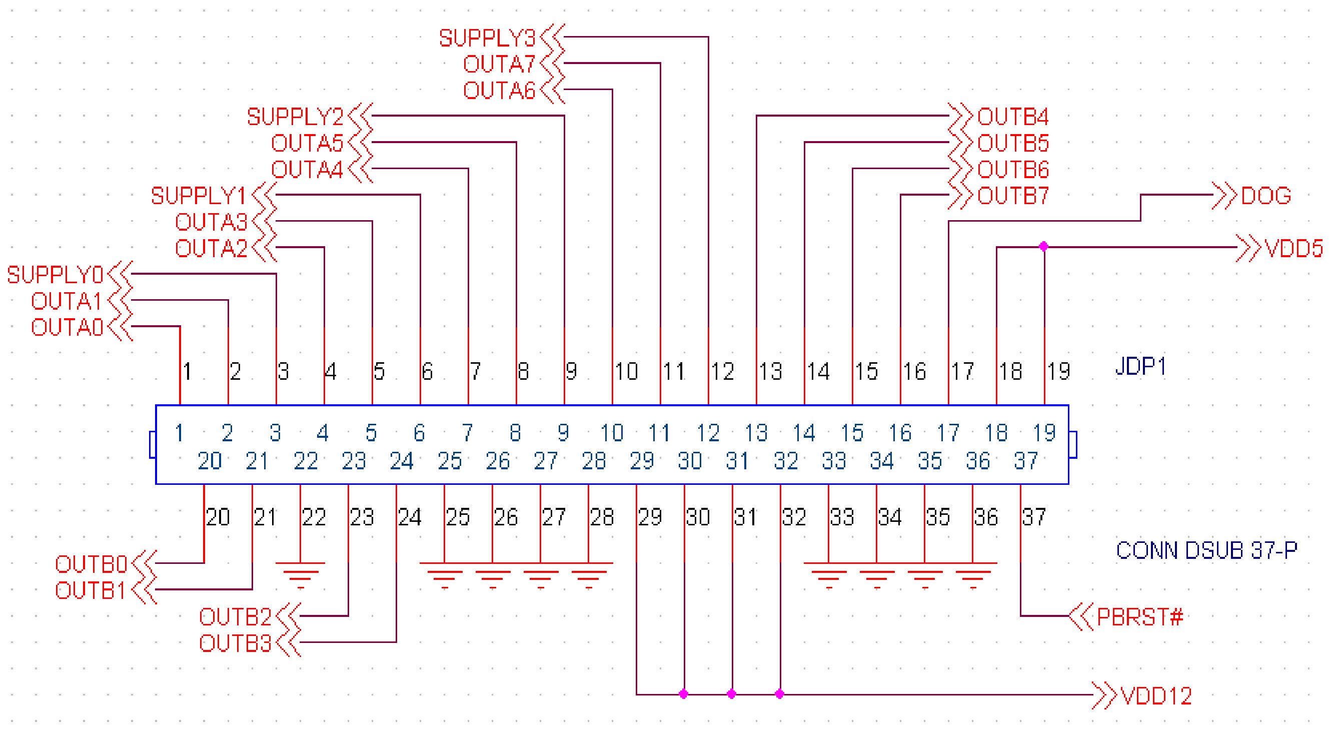


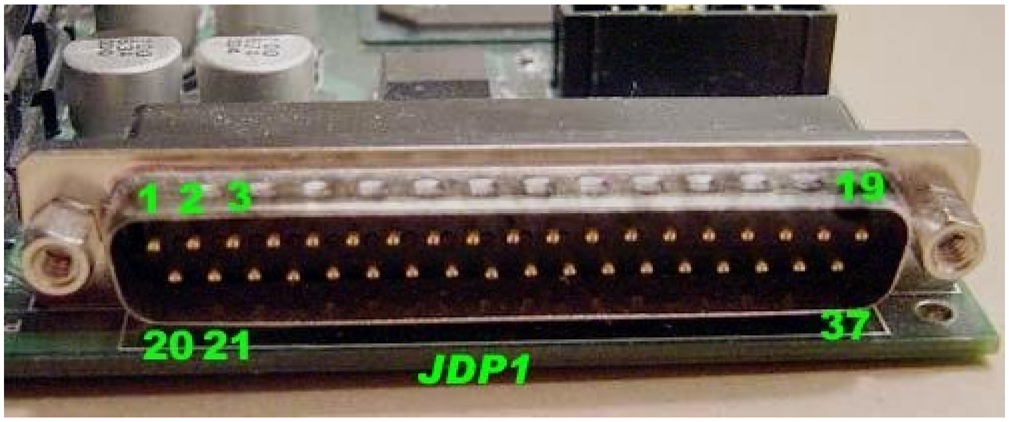
FIGURE 1. Basic H-Bridge Circuit

Each axis (and pair of full bridge drivers) share a common power supply pin, heat sink, encoder input, and current sense circuitry. Each axis may be of a different type (DC-Brush, 3-phase brushless, or stepper) and may use a different supply voltage (12-55V DC). If a DC-Brush motor is used for an axis only one of the full bridge drivers are used, the other is left unconnected. 3-phase brushless motors use 1½ full bridges (3 - half bridges), and stepper motors have 2 coils which require both full bridges.

	Motor type DC -Brush	**Motor type - 3 Phase brushless**	**Motor type - Stepper**
Axis 0	Connect Motor across OUTA0-OUTB0 Leave OUTA1-OUTB1 disconnected	Connect Phase A to OUTA0 Connect Phase B to OUTB0 Connect Phase C to OUTA1 Leave OUTB1 disconnected	Connect Coil A across OUTA0-OUTB0 Connect Coil B across OUTA1-OUTB1
Axis 1	Connect Motor across OUTA2-OUTB2 Leave OUTA3-OUTB3 disconnected	Connect Phase A to OUTA2 Connect Phase B to OUTB2 Connect Phase C to OUTA3 Leave OUTB3 disconnected	Connect Coil A across OUTA2-OUTB2 Connect Coil B across OUTA3-OUTB3
Axis 2	Connect Motor across OUTA4-OUTB4 Leave OUTA5-OUTB5 disconnected	Connect Phase A to OUTA4 Connect Phase B to OUTB4 Connect Phase C to OUTA5 Leave OUTB5 disconnected	Connect Coil A across OUTA4-OUTB4 Connect Coil B across OUTA5-OUTB5
Axis 3	Connect Motor across OUTA6-OUTB6 Leave OUTA7-OUTB7 disconnected	Connect Phase A to OUTA6 Connect Phase B to OUTB6 Connect Phase C to OUTA7 Leave OUTB7 disconnected	Connect Coil A across OUTA6-OUTB6 Connect Coil B across OUTA7-OUTB7

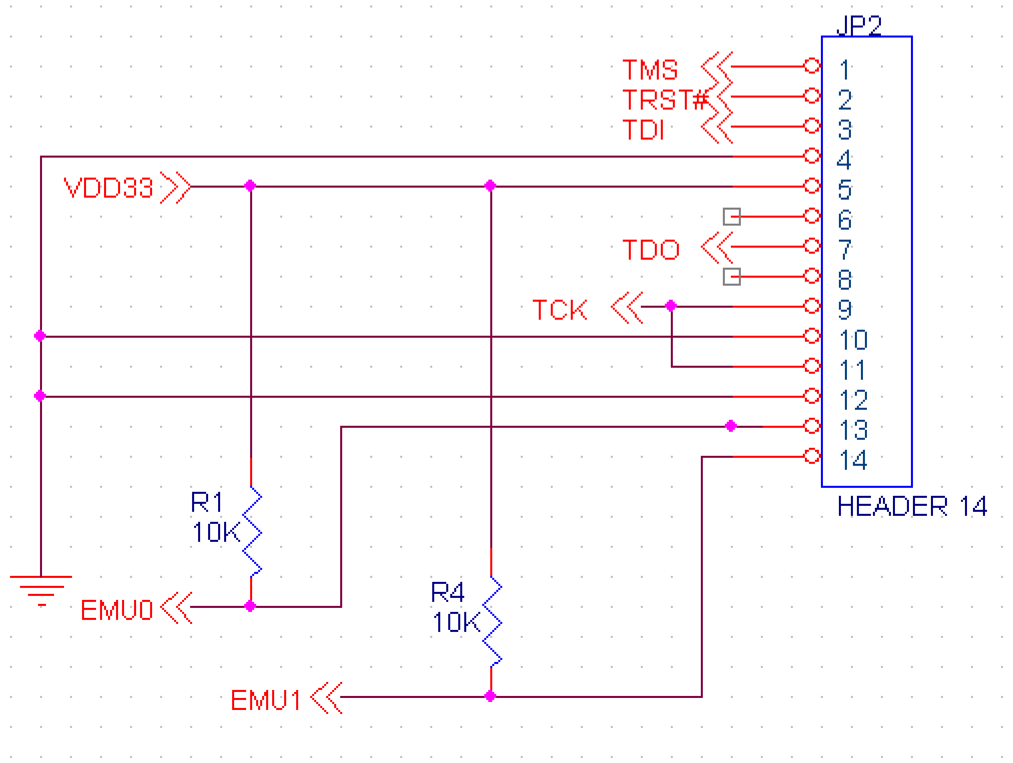
Also available on this connector is PBRST#. Short PBRST# to ground to reset the board. This signal is internally de-bounced. Under normal conditions, on-board power up reset should suffice so this pin may be left disconnected. Also connecting this pin to the DOG pin will enable the on board watchdog circuitry, whenever there is no DSP/FPGA activity detected for 1 second the board will be automatically reset. This is not normally required and may be left disconnected.





JP2 - JTAG

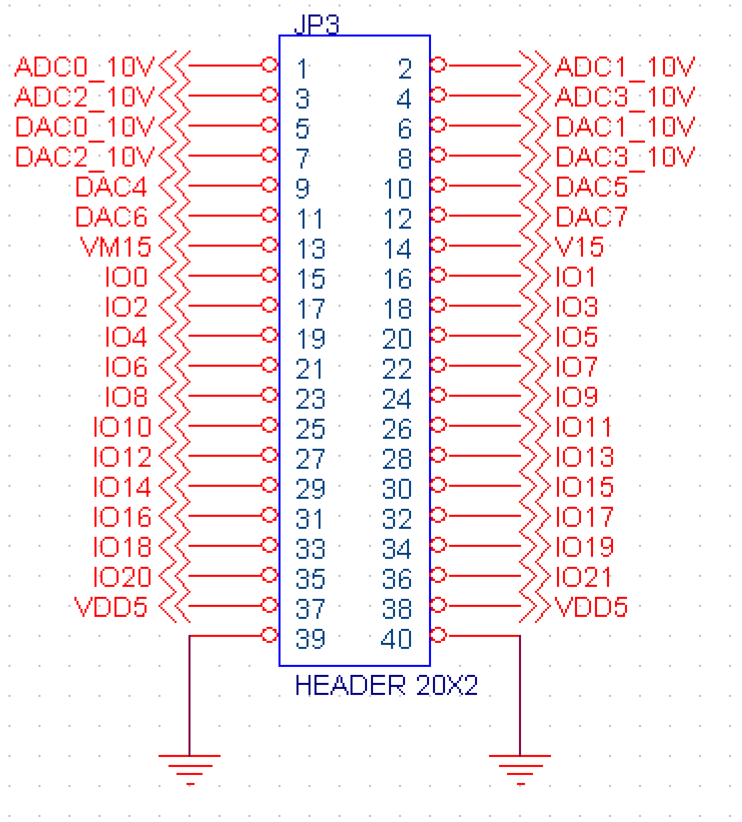
This connector is only used for advanced debugging using an XDS510 JTAG in circuit emulator. A small amount of regulated 3.3V (<0.5 Amp) is available on this connector if needed for external use.

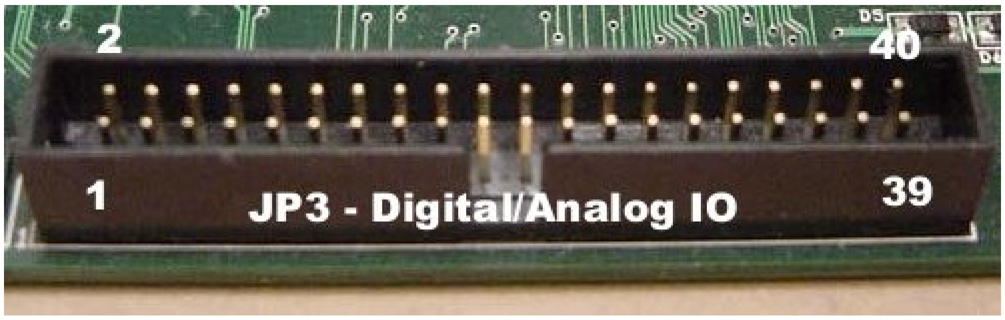


JP3 - Digital/Analog IO

4 channels of +/- 10V analog inputs, 4 channels of +/- 10V analog outputs, 4 channels of 0-4V analog outputs, 22 LVTTL bi-directional digital I/O, and +5, +15, -15 power supply outputs. Many Digital I/O bits are pre-defined as encoder, home, or limit inputs (see table below) but if not required for the particular application may be used as general purpose I/O. Digital Outputs may sink/source 10 ma. Digital I/O is LVTTL (3.3V) but is 5 V tolerant.

Caution! This connector contains +/- 15 v signals. Shorts to low voltage pins are likely to cause permanent damage to the board!





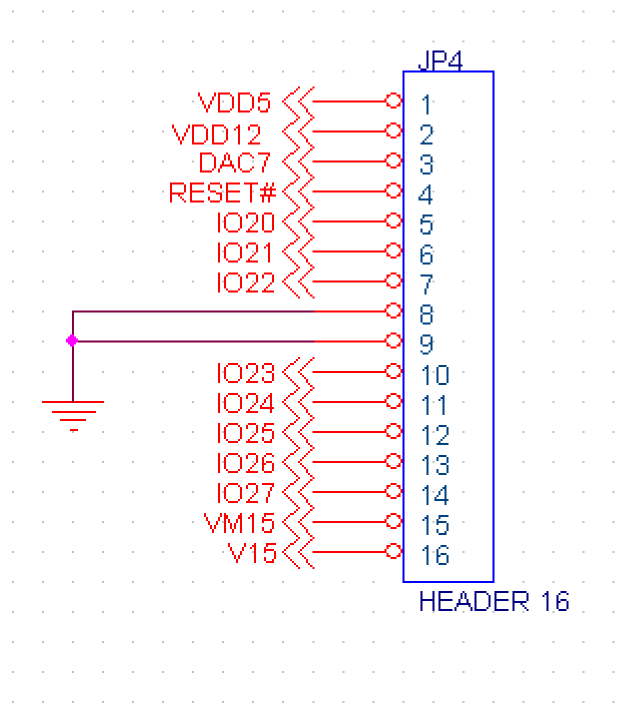
Pin	Name	Description
1	ADC0_10V	ADC Chan 0 +/- 10V input
2	ADC1_10V	ADC Chan 1 +/- 10V input
3	ADC2_10V	ADC Chan 2 +/- 10V input
4	ADC3_10V	ADC Chan 3 +/- 10V input
5	DAC0_10V	DAC Chan 0 +/- 10V output
6	DAC1_10V	DAC Chan 1 +/- 10V output

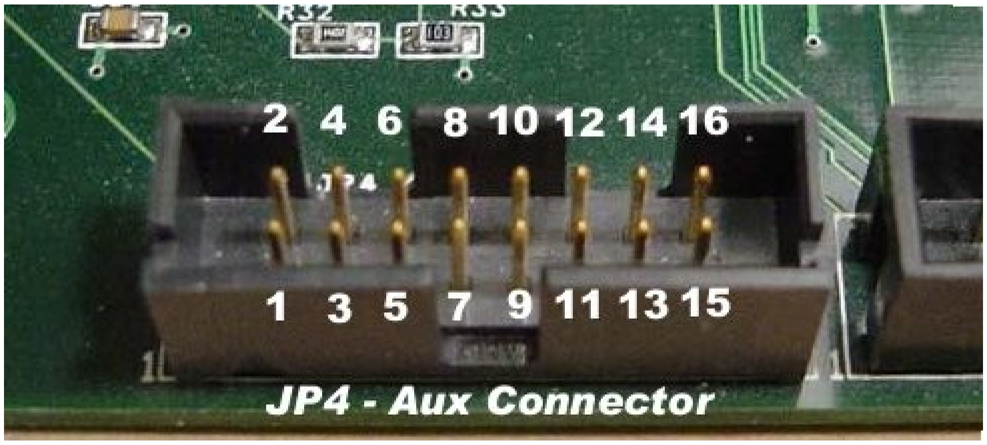
7	DAC2_10V	DAC Chan 2 +/- 10V output
8	DAC3_10V	DAC Chan 3 +/- 10V output
9	DAC4	DAC Chan 4 0-4V output
10	DAC5	DAC Chan 5 0-4V output
11	DAC6	DAC Chan 6 0-4V output
12	DAC7	DAC Chan 7 0-4V output
13	VM15	-15V @ 0.07 Amps Output
14	V15	+15V @ 0.07 Amps Output
15	IO0	Gen Purpose LVTTL I/O or Axis 0 Encoder Input Phase A
16	IO1	Gen Purpose LVTTL I/O or Axis 0 Encoder Input Phase B
17	IO2	Gen Purpose LVTTL I/O or Axis 1 Encoder Input Phase A
18	IO3	Gen Purpose LVTTL I/O or Axis 1 Encoder Input Phase B
19	IO4	Gen Purpose LVTTL I/O or Axis 2 Encoder Input Phase A
20	IO5	Gen Purpose LVTTL I/O or Axis 2 Encoder Input Phase B
21	IO6	Gen Purpose LVTTL I/O or Axis 3 Encoder Input Phase A
22	IO7	Gen Purpose LVTTL I/O or Axis 3 Encoder Input Phase B
23	IO8	Gen Purpose LVTTL I/O or Axis 0 Home
24	IO9	Gen Purpose LVTTL I/O or Axis 1 Home
25	IO10	Gen Purpose LVTTL I/O or Axis 2 Home
26	IO11	Gen Purpose LVTTL I/O or Axis 3 Home
27	IO12	Gen Purpose LVTTL I/O or Axis 0 + Limit
28	IO13	Gen Purpose LVTTL I/O or Axis 0 - Limit
29	IO14	Gen Purpose LVTTL I/O or Axis 1 + Limit
30	IO15	Gen Purpose LVTTL I/O or Axis 1 - Limit
31	IO16	Gen Purpose LVTTL I/O or Axis 2 + Limit

32	IO17	Gen Purpose LVTTL I/O or Axis 2 - Limit
33	IO18	Gen Purpose LVTTL I/O or Axis 3 + Limit
34	IO19	Gen Purpose LVTTL I/O or Axis 3 - Limit
35	IO20	Gen Purpose LVTTL I/O
36	IO21	Gen Purpose LVTTL I/O
37	VDD5	+5 Volts Output
38	VDD5	+5 Volts Output
39	GND	Digital and Analog Ground
40	GND	Digital and Analog Ground

JP4 - Aux Connector

Auxiliary connector which supplies power, reset, one 0-4V DAC, and 8 digital I/O normally connected to optional expansion daughter boards. If no expansion module is required these digital I/O may be used for general purpose use. Note: IO20 and IO21 are also routed to JP3

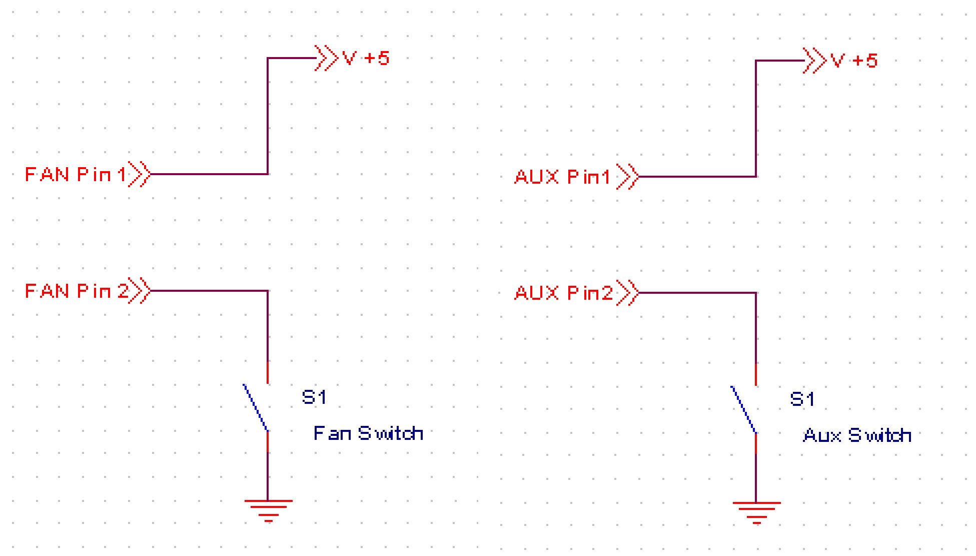




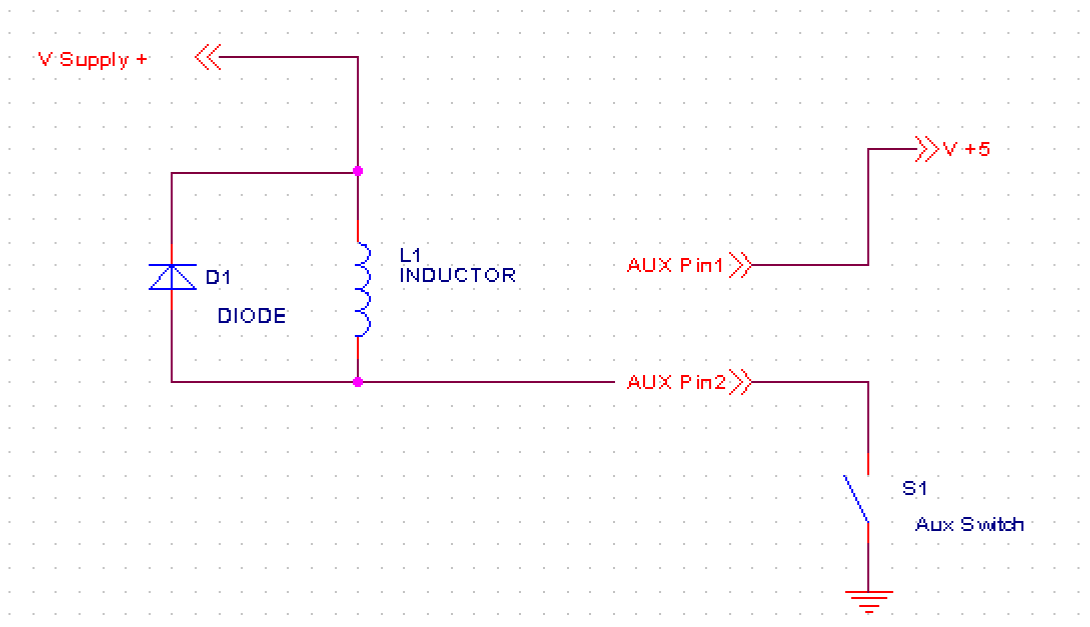
Pin	Name	Description
1	VDD5	+5 Volts Output
2	VDD12	+12 Volts Output
3	DAC7	DAC Chan 7 0-4V output
4	RESET#	Power up Reset (low true) output
5	IO20	Gen Purpose LVTTL I/O
6	IO21	Gen Purpose LVTTL I/O
7	IO22	Gen Purpose LVTTL I/O
8	GND	Digital and Analog Ground
9	GND	Digital and Analog Ground
10	IO23	Gen Purpose LVTTL I/O
11	IO24	Gen Purpose LVTTL I/O
12	IO25	Gen Purpose LVTTL I/O
13	IO26	Gen Purpose LVTTL I/O
14	IO27	Gen Purpose LVTTL I/O
15	VM15	-15V @ 0.07 Amps Output
16	V15	+15V @ 0.07 Amps Output

Fan and Aux Switches

Kmotion contains two SPST switches to ground capable of driving up to 30V @ 2A. One is dedicated to driving the cooling fan and one is available for another use. If the fan is not required by the application (less than 2Amps drawn on all axis) it may also be used to drive an external load. Each of the switches is connected to a 2 pin connector. One of the pins is connected to the 5V supply and the other pin is switched to ground. If the load requires 5V (i.e.. a 5V fan) it may be connected directly across the 2 pins. If other than a 5V load is required, then external user supplied wiring to the supply is required and only the pin that is switched to ground should be connected. See below.

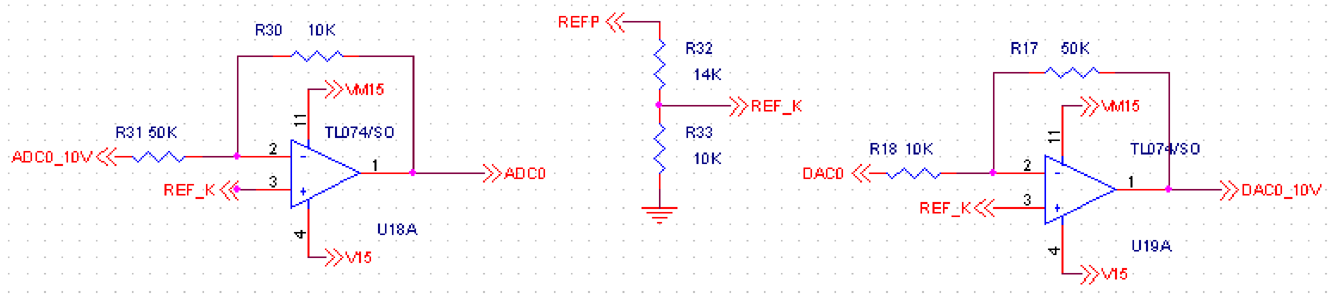


Here is an example of how a solenoid may be driven with the Auxiliary Switch:



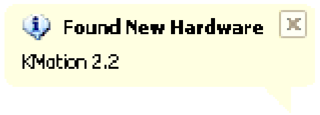
Analog I/O circuit

Circuit diagram of analog I/O buffers showing conversion from industry standard +/- 10V ranges to onboard 0-4V ADC/DAC converters. Note 50K input impedance. REFP = 4V, REFK = 1.43V



USB Installation

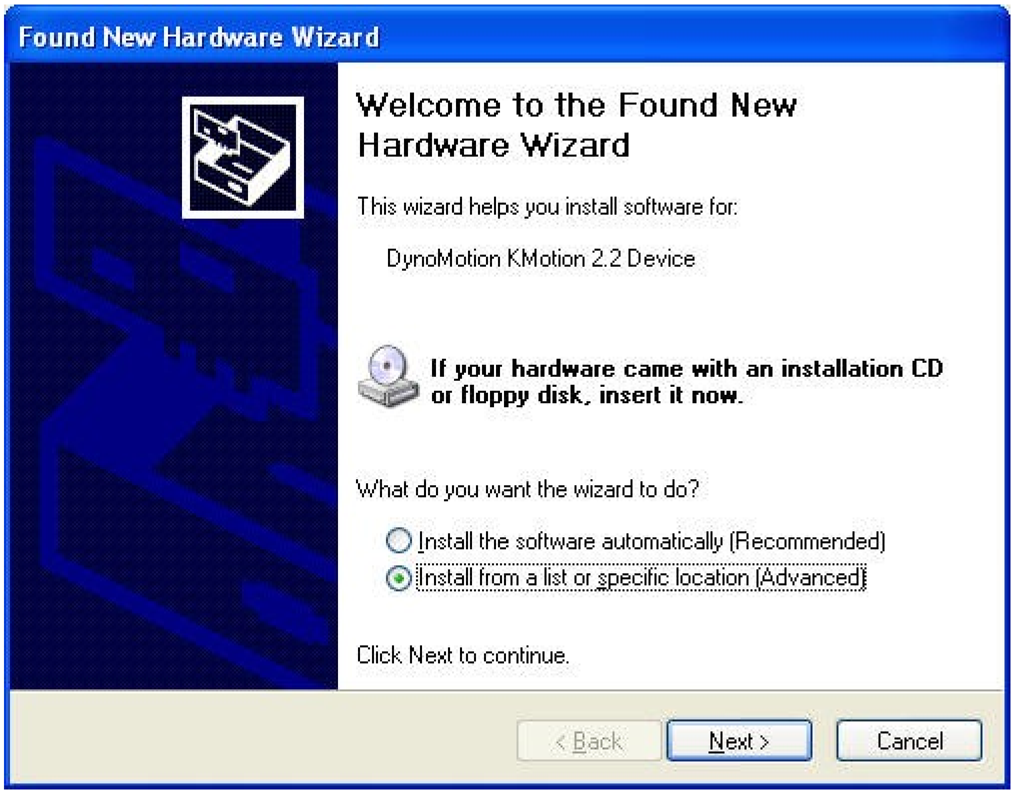
The first time the ***KMotion*** board is connected to a computer's USB port this message should be displayed near the Windows™ Task Bar.



Found New Hardware

KMotion 2.2

Shortly thereafter, the New Hardware Wizard Should appear.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

DynoMotion KMotion 2.2 Device

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

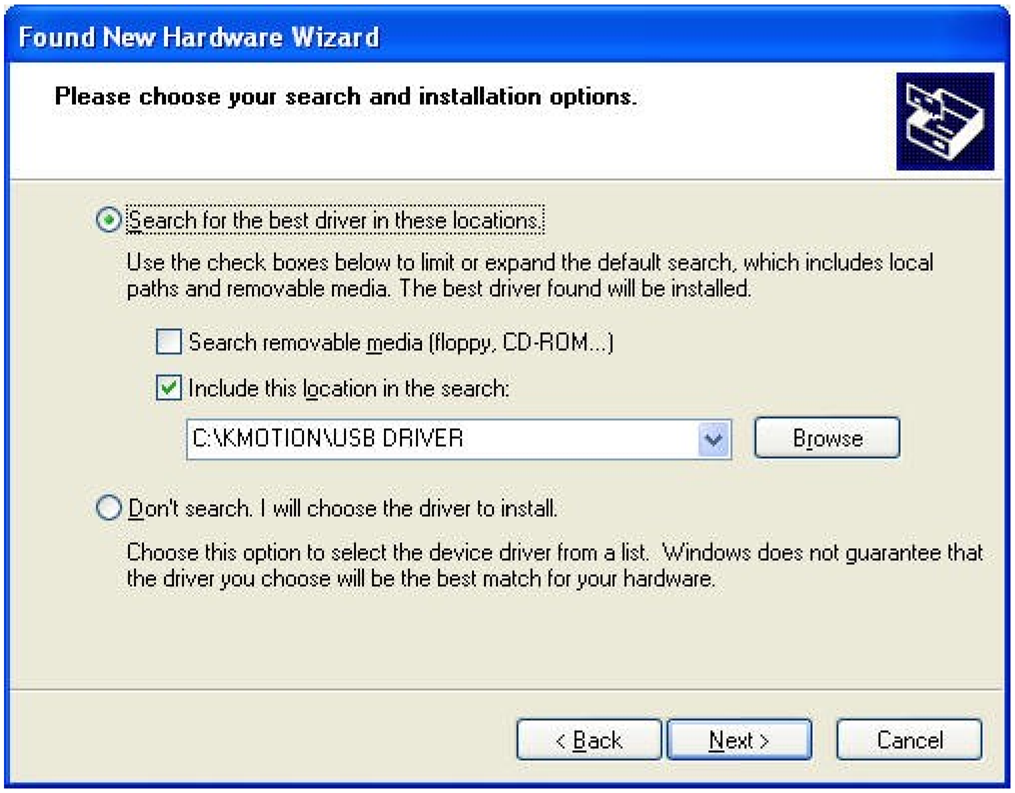
( ) Install the software automatically (Recommended)
(●) Install from a list or specific location (Advanced)

Click Next to continue.

If you haven't already, download and install the complete ***KMotion*** Software including drivers available at:

DynoMotion Software Download

Select: "**Install from a list or specific location (Advanced)**" and click **Next**


Found New Hardware Wizard

Please choose your search and installation options.

(● Search for the best driver in these locations.)

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

C:\KMOTION\USB DRIVER

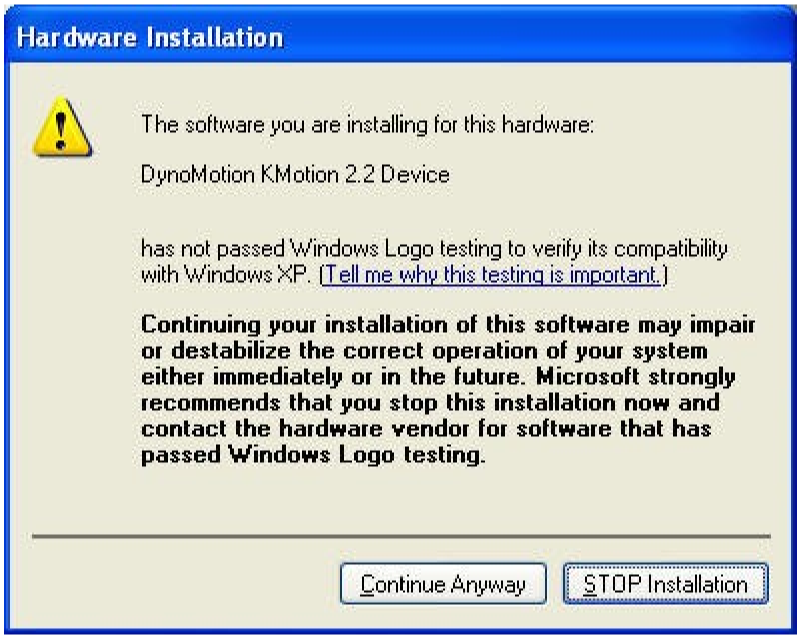
○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Browse to within the subdirectory where you selected the ***KMotion*** Software to be installed to the "**USB DRIVER**" subdirectory.

(If the software was installed into the default subdirectory, the location would be: **C:\KMOTION\USB DRIVER)**

Click **Next.**


Hardware Installation

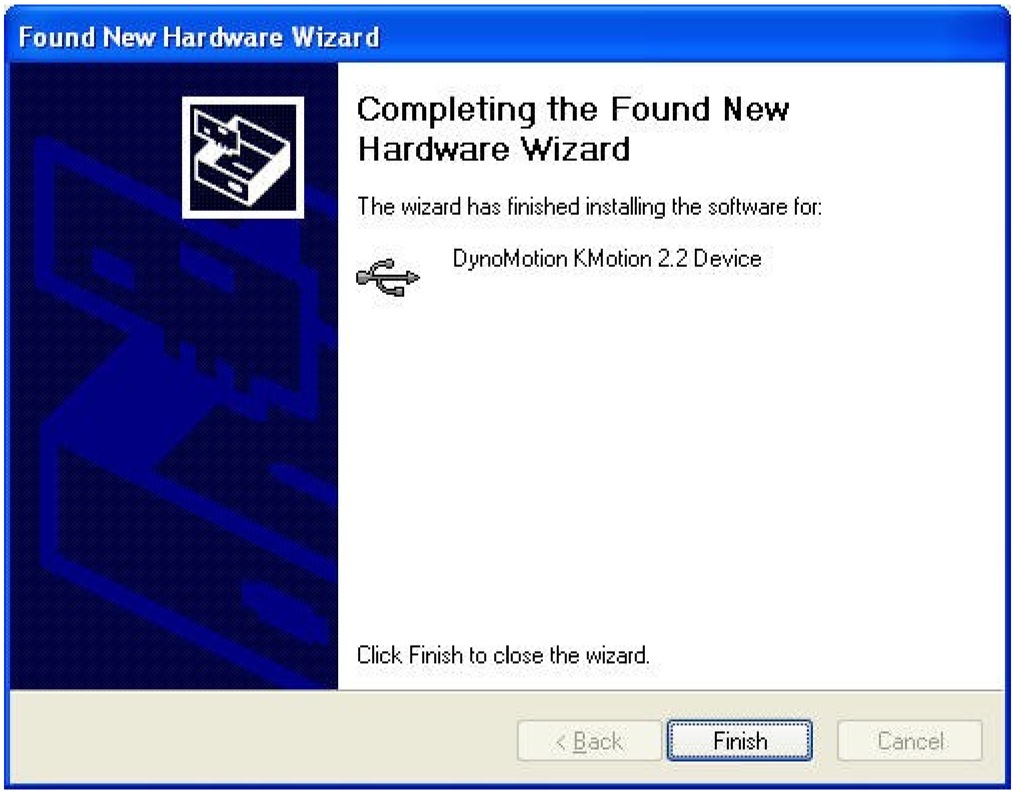
The software you are installing for this hardware:

DynoMotion KMotion 2.2 Device

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

If this screen appears select **Continue Anyway.**


Found New Hardware Wizard

Completing the Found New Hardware Wizard

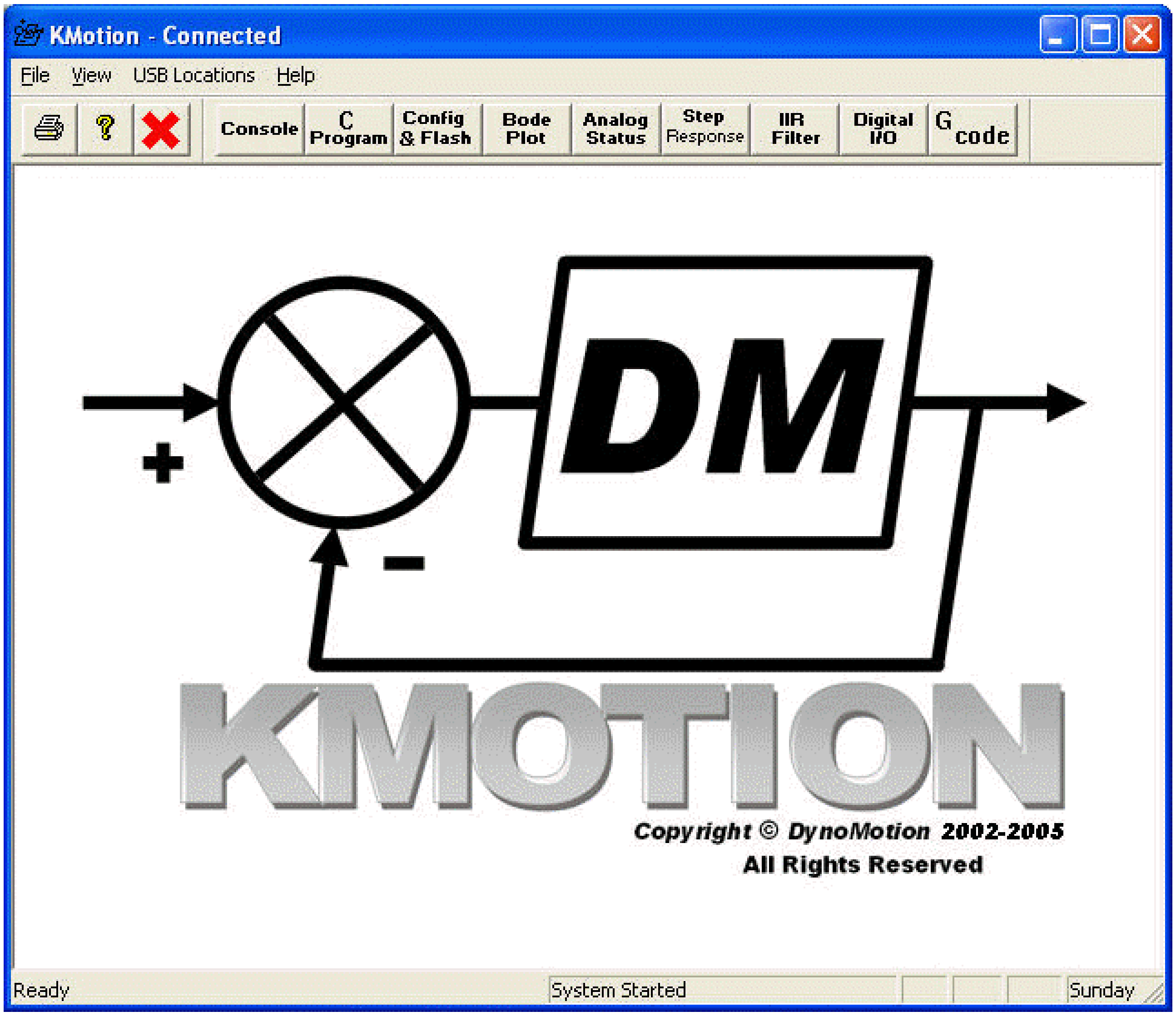
The wizard has finished installing the software for:

DynoMotion KMotion 2.2 Device

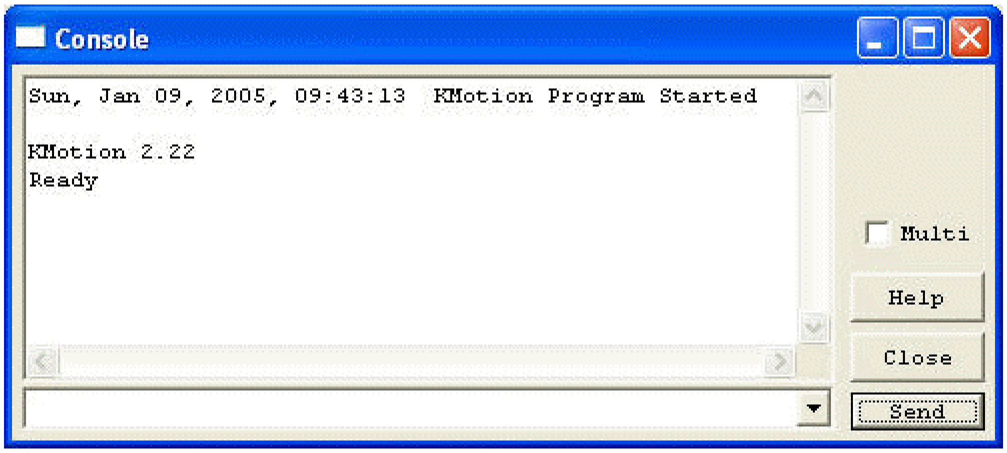
Click Finish to close the wizard.

If successful, the above screen should appear. Click **Finish.**

The ***KMotion*** Board is now ready for use. To verify proper USB connection to the KMotion Board, use the Windows™ Start Button to launch the ***KMotion*** Application.



At the main tool bar select the "Console" button to Display the Console Screen.



```
Sun, Jan 09, 2005, 09:43:13  KMotion Program Started

KMotion 2.22
Ready
```

Press the "Send" button to send a blank line to the ***KMotion*** Board. The following should be displayed indicating successful communication between the PC and ***KMotion*** Board.

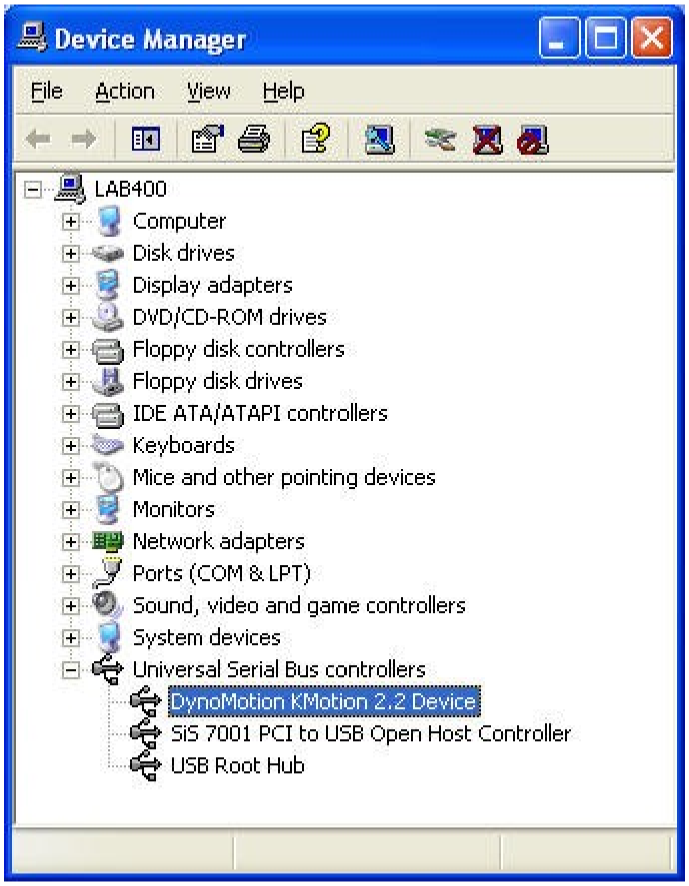
KMotion 2.2

Ready

]

USB Installation Trouble Shooting

If properly installed, a **DynoMotion KMotion 2.2 Device** should appear under **Universal Serial Bus controllers** within the **Device Manager**.



If within Control Panel – System - Device Manager – Universal Serial Bus controllers

There is an item "USB High Speed Serial Converter"

This is an incorrect driver.

Perform the following steps to change the driver:

Right Click – select "Update Driver"

Select “Install from a specific location (Advanced)” then Next

Select “Don’t Search I will choose the driver to install” then Next

Select “Have Disk”

Browse to within the directory chosen during the KMotion Installation (Default is C:\KMotion) to subdirectory “USB Driver”

Select file “ftd2xx.inf”

Select OK

Within the Hardware Update Wizard select to highlight “DynoMotion KMotion 2.2 Device”

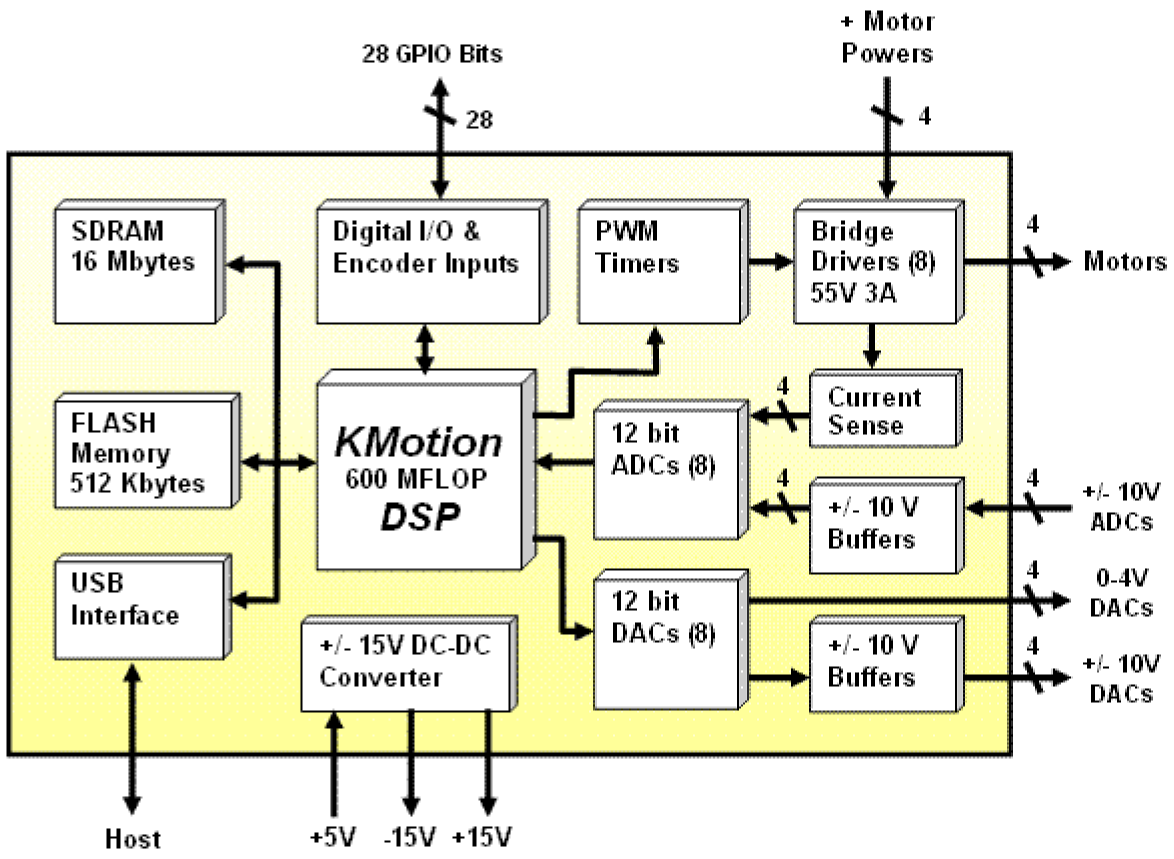
then Next

Select “Continue Anyway”

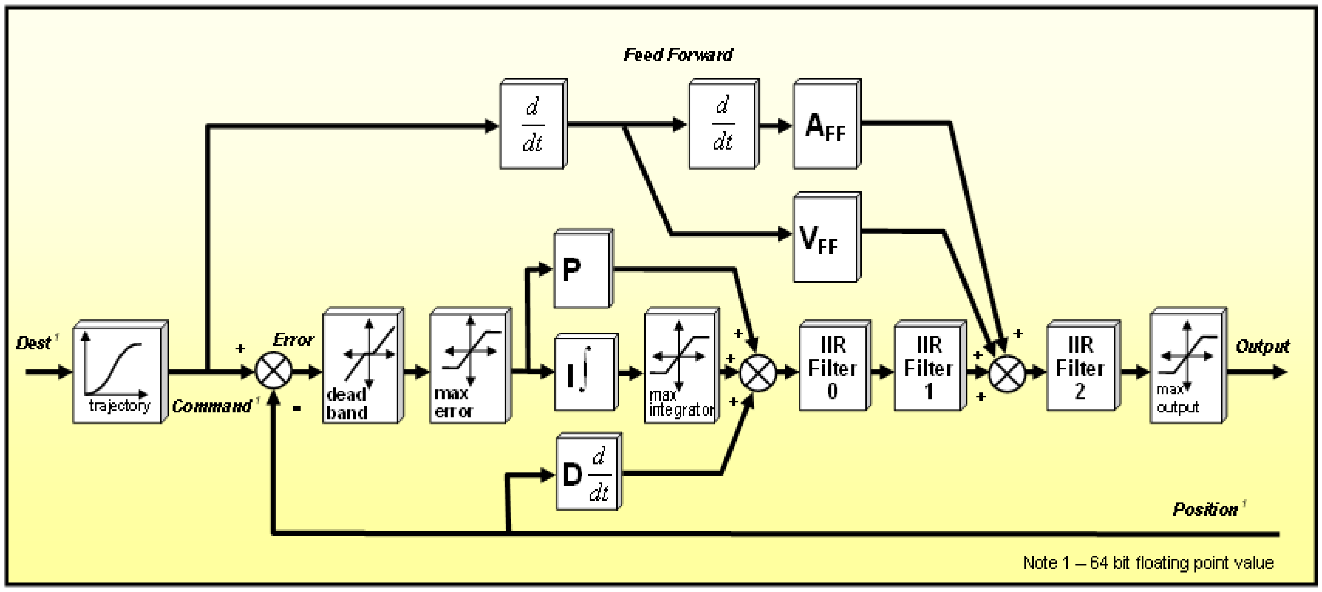
Select Finish

Within Control Panel – System - Device Manager – Universal Serial Bus controllers

There should now be an item “DynoMotion KMotion 2.2 Device”



KMotion 2.2 Block Diagram



KMotion 2.2 Servo Flow Diagram

Data Gathering

KMotion provides a flexible method for capturing data of all types every servo sample period (90µs). This same method is how ***KMotion*** gathers step response and Bode plot data.

Basically a list of addresses and data types are defined. An end address of where to stop capturing data is set, and when triggered the Servo Interrupt will capture the specified data values. All values are converted to double precision numbers before being placed into the gather buffer. The maximum size of the Gather Buffer is 1,000,000 double precision values (8 MBytes).

```
#define MAX_GATHER_DATA 1000000 // Size of gather buffer (number of doubles, 8 bytes each).
```

The following example shows how to setup to capture the two PWM drives (for a stepper motor) and the commaned destination for a 0.5 second time period, trigger the capture, make a simple move, wait until the capture is complete, and print the results.

```
#include "KMotionDef.h"

main()
{
     int i,n_Samples = 0.5 / TIMEBASE;

     gather.Inject = FALSE; // Don't inject any Data anywhere

     gather.list[0].type = GATHER_LASTPWM_TYPE; // Gather PWM 0
     gather.list[0].addr = &LastPWM[0];

     gather.list[1].type = GATHER_LASTPWM_TYPE; // Gather PWM 1
     gather.list[1].addr = &LastPWM[1];

     gather.list[2].type = GATHER_DOUBLE_TYPE; // Gather Dest axis 0
     gather.list[2].addr = &chan[0].Dest;

     gather.list[3].type = GATHER_END_TYPE;

     gather.bufptr = (double *)0xfffffffc; // force more than endbuf
     gather.endptr = gather_buffer + 3 * n_Samples;

     TriggerGather(); // start capturing data

     MoveRel(0,20); // Start a motion

     while (!CheckDoneGather()) ; // what till all captured

     // print all captured data (every 50th sample)

     for (i=0; i<n_Samples; i+=10)
```

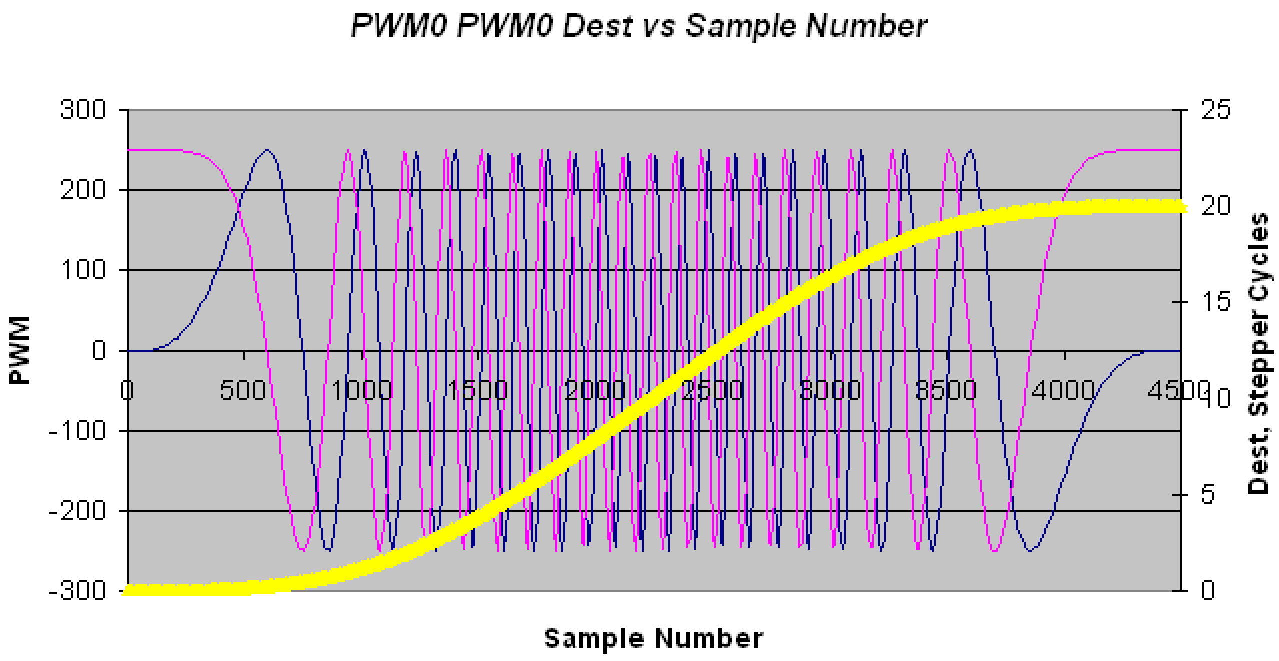
```
    printf("%d,%f,%f,%f\n", i,gather_buffer[i*3],
    gather_buffer[i*3+1],
    gather_buffer[i*3+2]);


}
```

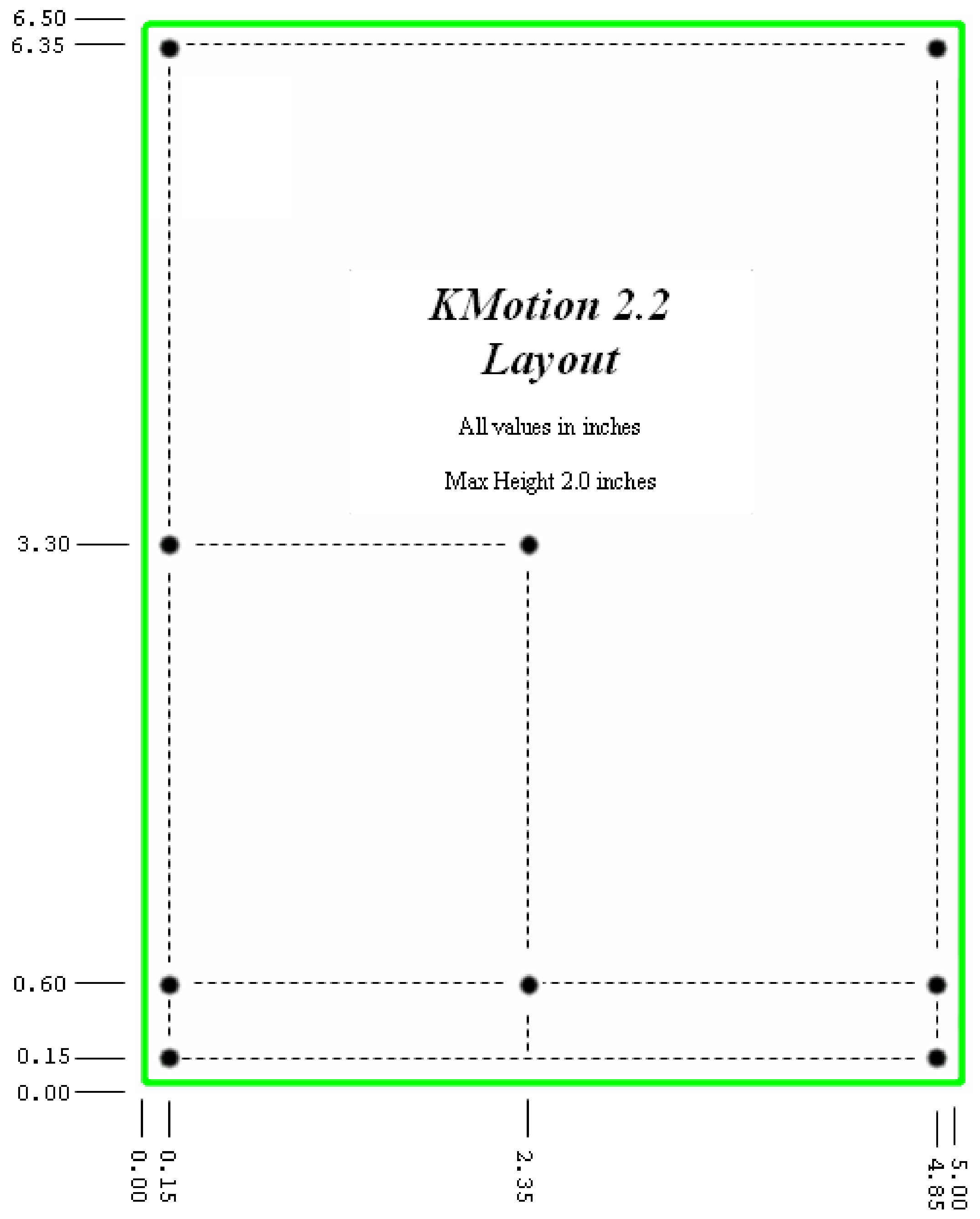
Data will be printed to the ***KMotion*** Console Screen which is also written to a permanent log file at:

<KMotionInstallDir>\KMotion\Data\LogFile.txt

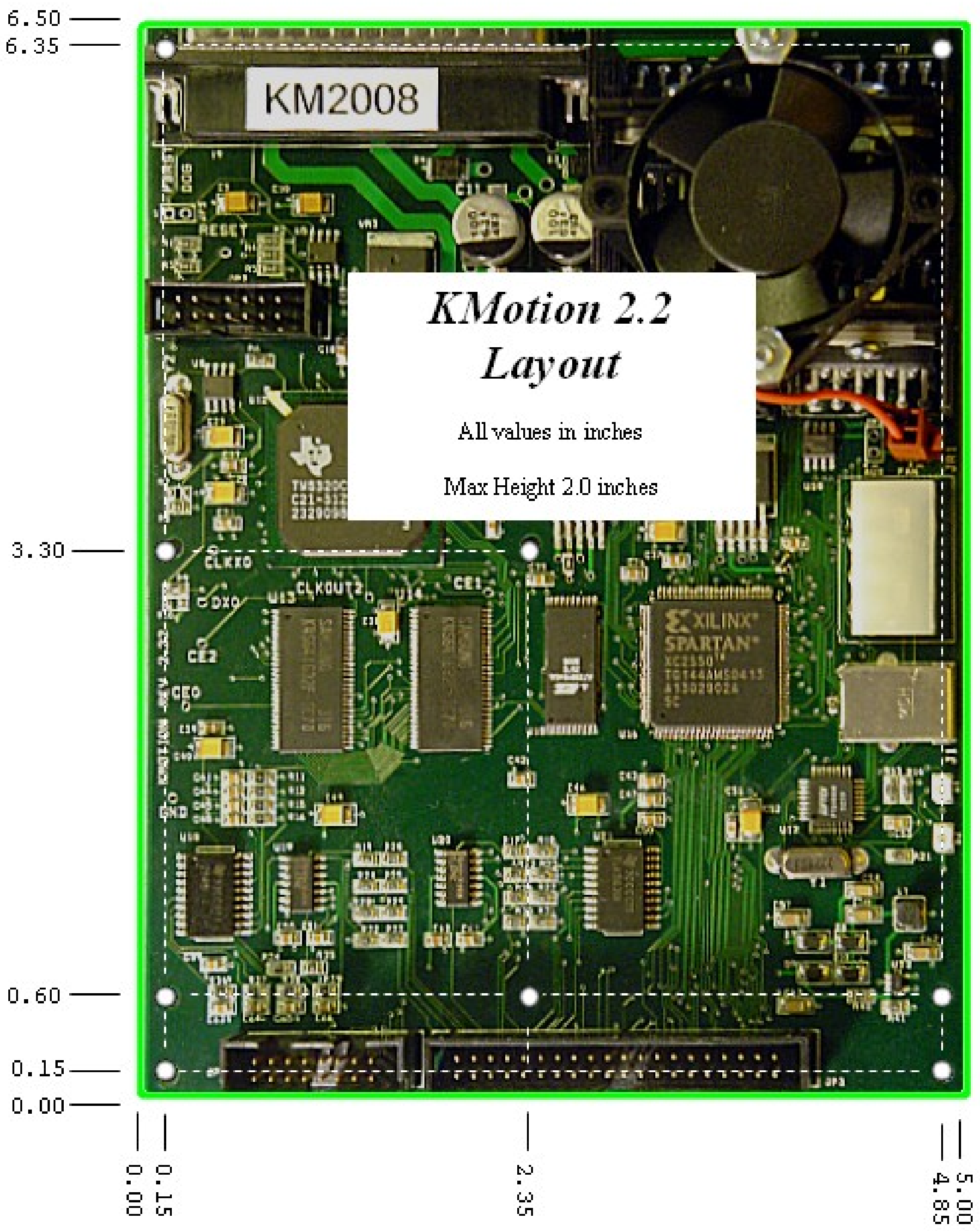
Normally data scrolls off of the Console Screen into the permanent log file, to flush all data into the log file, exit the ***KMotion*** application.

An Excel plot of the captured data is shown below.





KMotion 2.2
Layout
All values in inches
Max Height 2.0 inches
6.50
6.35
3.30
0.60
0.15
0.00
0.00
0.15
2.35
4.85
5.00


KMotion 2.2
Layout
All values in inches
Max Height 2.0 inches
6.50
6.35
3.30
0.60
0.15
0.00
0.00
0.15
2.35
4.85
5.00
KM2008

KMotionDLL

KMotion Quick Reference

Send Commands

WriteLine

WriteLineReadLine
ReadLineTimeOut

Board Locks

WaitToken

KMotionLock

ReleaseToken

Failed

Console

ServiceConsole

SetConsoleCallback

Coff Loader

LoadCoff

Compiler

CompileAndLoadCoff

USB

ListLocations

KMotion DLL Functions

`int WriteLine(int board, const char *s);`

Writes a null terminated string of characters to a specified ***KMotion*** Board. There is no wait for any response.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified)

Parameters

board

specifies which board in the system the command applies to

s

Null terminated string to send

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;


if (KM.WriteLine(0, "Move0=1000") MyError();
```

`int WriteLineReadLine(int board, const char *s, char *response);`

Writes a null terminated string of characters to a specified ***KMotion*** Board. Waits for a response string. This command is thread safe. It waits for the token for the specified board, sends the command, waits for the response, then releases the board.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified, timeout on the response)

Parameters

board

specifies which board in the system the command applies to

s

Null terminated string to send

response

Buffer to receive the null terminated string received as response

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp[256];

while
{
        if (KM.WriteLineReadLine(0, "CheckDone0",resp) MyError();

        if (strcmp(resp,"1")==0) break;

}
```

`int ReadLineTimeOut(int board,char *buf, int TimeOutms);`

Waits for a response string from a previously issued command. Note in a multi-process or multi thread environment the ***KMotion*** board should be locked prior to issuing a command that has a response(s), Otherwise there is a possibility that another process or thread may receive the expected response.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified, timeout on the response)

Parameters

board

specifies which board in the system the command applies to

buf

Buffer to receive the Null terminated string received as response

TimeOutms

Amount of time to receive a response

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp1[256];

char resp2[256];

char resp3[256];

 // first get the token for the board to allow uninterrupted access
if (KM.WaitToken(0)!=KMOTION_LOCKED) MyError();

// tell the board to send 24 (32 bit) words at offset 0
if (KM.WriteLine(0,"GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0,resp1)) MyError();
if (KM.ReadLineTimeout(0,resp2)) MyError();
if (KM.ReadLineTimeout(0,resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

`int WaitToken(int board);`

Waits until the token for the specified ***KMotion*** board can be obtained. Call this function whenever uninterrupted access to a ***KMotion*** board is required. For example before a command where several lines of response will be returned. Release the token as quickly as possible by calling the ReleaseToken function as all other access to the locked board will be blocked until released.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified)

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"

CKMotionDLL KM;

char resp1[256];

char resp2[256];

char resp3[256];


// first get the token for the board to allow uninterrupted access
if (KM.WaitToken(0)!=KMOTION_LOCKED) MyError();

// tell the board to send 24 (32 bit ) words at offset 0
if (KM.WriteLine(0,"GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0,resp1)) MyError();
if (KM.ReadLineTimeout(0,resp2)) MyError();
if (KM.ReadLineTimeout(0,resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

`int KMotionLock(int board);`

Attempts to obtain the token of the specified ***KMotion*** board.. Call this function whenever uninterrupted access to a ***KMotion*** board is required. For example before a command where several lines of response will be returned. Release the token as quickly as possible by calling the *ReleaseToken* function as all other access to the locked board will be blocked until released. This function is similar to the *WaitToken* function, except that it returns immediately (instead of waiting) if the board is already locked.

Return Value

KMOTION_LOCKED=0, // (and token is locked) if KMotion is available for use
KMOTION_IN_USE=1, // if already in use
KMOTION_NOT_CONNECTED=2 // if error or not able to connect

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;

char resp1[256];

char resp2[256];

char resp3[256];

int result;

// first get the token for the board to allow uninterrupted access
do

{

        result = KM.KMotionLock(0);

        if (result == KMOTION_NOT_CONNECTED) MyError();
        if (result == KMOTION_IN_USE) DoOtherProcessing();

}

// tell the board to send 24 (32 bit) words at offset 0
if (KM.WriteLine(0,"GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0,resp1)) MyError();
if (KM.ReadLineTimeout(0,resp2)) MyError();
if (KM.ReadLineTimeout(0,resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

void ReleaseToken(int board);

Releases the previously obtained token of the specified ***KMotion*** board. See WaitToken and LockKMotion functions. The token should always be released as quickly as possible as all other access to the locked board will be blocked until released.

Return Value

none - the function can not fail

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp1[256];
char resp2[256];
char resp3[256];
int result;
// first get the token for the board to allow uninterrupted access
do
{
        result = KM.KMotionLock(0);
        if (result == KMOTION_NOT_CONNECTED) MyError();
        if (result == KMOTION_IN_USE) DoOtherProcessing();
}
```

```
// tell the board to send 24 (32 bit) words at offset 0
if (KM.WriteLine(0,"GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0,resp1)) MyError();
if (KM.ReadLineTimeout(0,resp2)) MyError();
if (KM.ReadLineTimeout(0,resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

int Failed(int board);

This function should be called whenever an error is detected with a ***KMotion*** board. This function disconnects the driver, flags the board as disconnected, and displays the error message shown below. A user program may typically detect a timeout error or invalid data error if the ***KMotion*** board is powered down or unplugged while communication is in progress. Calling this function will force any subsequent attempts to access the board to wait for a board to be connected, re-connect, flush any buffers, etc...

"Read Failed - Auto Disconnect"

Return Value

always 0 - the function can not fail

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
if (KM.KMotionLock(0) == KMOTION_LOCKED) // see if we can get access
{
        // upload bulk status

        if (UploadStatus())
        {
                // error reading status
                KM.Failed(0);
        }
        KM.ReleaseToken(0);
}
```

int LoadCoff(int board, const char *Name, unsigned int *EntryPoint, bool PackToFlash);

This function downloads a compiled C program to the memory of the specified ***KMotion*** board.

C Programs that run in the ***KMotion*** Board are normally compiled using the included and integrated compiler in the ***KMotion*** Application. Using the ***KMotion*** Application the user's C Program should be loaded into a selected thread and compiled. This will automatically generate a COFF executable with the same name and in the same directory as the C Source code, but with a .out extension. It is the users responsibility to keep track of which thread the COFF executable was compiled to execute in.

This function is used to download the COFF executable to the ***KMotion***'s memory. The entry point of the executable will be extracted from the file and retuned to the caller. This entry point address should then be passed to the ***KMotion*** board and associated with the same thread that the file was compiled to execute in. The entry point should be passed using the "EntryPoint" command (See the example below).

The downloaded code may then be executed by issuing the "Execute" command

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

Name

Filename of coff file to download

EntryPoint

Address of the entry point of the C program extracted form the COFF file and returned to the caller

PackToFlash

Internal system command always specify as false

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
unsigned int EntryPoint;


if (KM.LoadCoff(0, "C:\\test.out", &EntryPoint, false)) return 1;

s.Format("EntryPoint 0 %08X", EntryPoint);
KM.WriteLine(0,s);

KM.WriteLine(0,"Execute 0");
```

`int ServiceConsole(int board);`

Services the ***KMotion*** *Console* data stream. The *Console* is a place where all unsolicited data, such as errors, or data "Printed" by user programs goes to. In between processing commands, ***KMotion*** uploads any unsolicited data it may have up to the host. The KMotionDLL driver buffers this data until some process declares itself as a *Console Handler (See* **`SetConsoleCallback)`** and makes calls to this function *ServiceConsole*.

This function should be called at regular intervals. If console data is available a call back to the Console Handler will

occur with one line of data.

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;

int ConsoleHandler(const char *buf)
{
        MyLogData(buf);
        return 0;
}

KM.SetConsoleCallback(0, ConsoleHandler);

KM.ServiceConsole(0);
```

int SetConsoleCallback(int board, CONSOLE_HANDLER *ch);

Sets the user provided console callback function.

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

ch

name of console handler function

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;

int ConsoleHandler(const char *buf)
```

```
{
        MyLogData(buf);
        return 0;
}

KM.SetConsoleCallback(0, ConsoleHandler);

KM.ServiceConsole(0);
```

`int CompileAndLoadCoff(int board, const char *Name, int Thread);`

or

```
int CompileAndLoadCoff(int board, const char *Name, int Thread,
                                 char *Err, int MaxErrLen);
```

Compiles the specified C Program file, downloads the object code to the specified Thread space, and sets the Entry Point, for the specified thread. Two versions of the function are supplied; one returns any error messages, the other does not.

The downloaded code may then be executed by issuing the Execute command.

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

Name

Filename of C Program to compile and download

Thread

Thread number where the program is to be compiled for and downloaded into. Valid range 1...7.

Err

Caller's supplied buffer for any error messages

MaxErrLen

Length of caller's supplied buffer for any error messages

Example

```
#include "KMotionDLL.h"
CKMotionDLL KMotion;

if (KM.CompileAndLoadCoff(0, "C:\\MyProgram.c", 1) MyError();
if (KM.WriteLine(0, "Execute1") MyError();
```

`int ListLocations(int *nlocations, int *list);`

Returns the number of currently connected KMotion boards and a list of their USB location identifiers

Return Value

returns 0 - if successful

Parameters

nlocations

pointer to integer where the number of locations should be returned

List

pointer to array to be filled in with the list of USB location identifiers

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
```

```
int n_boards;
int BoardList[256];

if (KM.ListLocations(&n_boards, BoardList) MyError();
```

Hardware

Function	Parameter	Specification
Processor	CPU Memory	TMS320C67-100MHz DSP 600 MFLOP 32/64-Bit Native Floating Point FLASH 512KBytes SDRAM 16 Mbytes
Interface	Host	USB 2.0 Full Speed
Connectors	Motor/Power I/O Aux IO USB System Power	37 pin DSUB 40 pin Header 16 pin Header Type B Molex 4-pin (Disk drive type)
Servo Loop	Sample Rate Compensation Feed Forward	90µs PID + (3) IIR bi-quad Stages/Axis Acceleration + Velocity
Axis	Number Type	4 MicroStep/Servo/Brush/Brushless
Power Amps	Number Type Cont. Current Peak Current Max Supply Min Supply Independent Supply Inputs Thermal Protection Over Current Protection Under Voltage Protection Current Measurement	8 Full H-Bridges Switching 30KHz 3 A each Axis 6 A +55V +12V Yes Yes Yes Yes Yes
Logic Supply	Voltage Max Current	+5V ±10% 2.5A
User I/O	Digital Encoders Analog	28 Gen Purpose LVTTL (4) single-ended, 1 MHz (4) ±10V ADCs - (4) 0-4V ADC - (4) ±10V DACs
Environment	Operating Temperature Storage Temperature Humidity	0-40º C 0-40º C 20-90% Relative Humidity, non-condensing
±15V	On-board DC-DC Generator	2 Watts (70ma each supply)
Dimensions	Length Width Height	6.5 inches (165 mm) 5.0 inches (127 mm) 2.0 inches (50 mm)

Software

Function	Parameter	Specification
User Programs	Language Number concurrent Stand alone mode	C 7 Yes
Host Requirements	OS Interface	MS Windows™ 2000, MS Windows™ XP USB 2.0
Interface Library	Multi-Thread Multi-Process Multi-Board MS Windows™ VC++ MS Windows™ VB	Yes Yes Yes Supported Supported
C Compiler	TCC67	Included
G Code	Interpreter	Included
Script Language	ASCII Commands	Included
Trajectory Planner	Coordinated Motion	4 Axis
Executive Application	Configuration Tuning User Programs G Code Command Console Status Display	Upload/Download/Save/Load Motor Config Move/Step Response, Bode Plot, Calc Filters Integrated IDE - Edit/Compile/Download/Exec Integrated ASCII Command Entry - Log Console Axis/Analog/Digital

Commands (by category):

Parameters

Accel <N>=<A>

CommutationOffset<N>=<X>

D<N>=<M>

DeadBandGain<N>=<M>

DeadBandRange<N>=<M>

Dest<N>=<M>

FFAccel<N>=<M>

FFVel<N>=<M>

I<N>=<M>

IIR<N> <M>=<A1> <A2> <B0> <B1> <B2>

InputChan<M> <N>=<C>

InputGain<M> <N>=<G>

InputMode<N>=<M>

InputOffset<M> <N>=<O>

InvDistPerCycle<N>=<X>

Jerk<N>=<J>

Lead<N>=<M>

LimitSwitch<N>=<H>

MaxErr<N>=<M>

MaxFollowingError<N>=<M>

MaxI<N> <M>

MaxOutput<N>=<M>

OutputChan<M> <N>=<C>

OutputMode<N>=<M>

P<N>=<M>

Pos<N>=<P>

StepperAmplitude<N>=<M>

Vel<N>=<V>

User Threads

EntryPoint<N> <H>

Execute<N>

Kill<N>

LoadData <H> <N>

SetStartupThread<N> <M>

I/O Commands /Status

ADC<N>

ClearBit<N>

ClearBitBuf<N>

DAC<N> <M>

GetBitDirection<N>

ReadBit<N>

SetBit<N>

SetBitBuf<N>

SetBitDirection<N>=<M>

SetStateBit<N>=<M>

SetStateBitBuf<N>=<M>

Output Stage

3PH<N>=<M> <A>

4PH<N>=<M> <A>

PWM<N>=<M>

PWMR<N>=<M>

Gather Commands

CheckDoneGather

GatherMove<N> <M> <L>

GatherStep<N> <M> <L>

GetGather <N>

GetGatherDec<N>

GetGatherHex<N> <M>

GetInject<N> <M>

Inject<N> <F> <A>

SetGatherDec <N> <M>

SetGatherHex<N> <M>

Motion Commands

Arc <XC> <YC> <RX> <RY>
<θ0> <dθ> <Z0> <Z1>
<a> <b> <c> <d> <tF>

ArcHex <XC> <YC> <RX> <RY>
<θ0> <dθ> <Z0> <Z1>
<a> <b> <c> <d> <tF>

CheckDone<N>

CheckDoneBuf

CheckDoneXYZA

DefineCS<X> <Y> <Z> <A>

DisableAxis<N>

EnableAxis<N>

EnableAxisDest<N> <M>

Enabled<N>

ExecBuf

ExecTime

Jog<N>=<V>

Linear <X0> <Y0> <Z0> <A0>
<X1> <Y1> <Z1> <A1>
<a> <b> <c> <d> <tF>

LinearHex <X0> <Y0> <Z0> <A0>
<X1> <Y1> <Z1> <A1>
<a> <b> <c> <d> <tF>

Move<N>=<M>

MoveAtVel<N>=<M> <V>

MoveRel<N>=<M>

MoveRelAtVel<N>=<M> <V>

MoveXYZA <X> <Y> <Z> <A>

OpenBuf

Zero<N>

	FLASH Commands	Misc Commands
	ClearFlashImage Flash LoadFlash<H> <N> ProgFlashImage	Echo <S> GetPersistDec<N> GetPersistHex<N> GetStatus Reboot! SetPersistDec <O> <D> SetPersistHex <O> <H> Version

3PH<*N*>=<*M*> <*A*>

Description

Sets the assigned PWMs of an axis to the specified magnitude and phase angle for a brushless 3 phase motor.

This command is useful for energizing a coil (or effective coil position). This is often required while initial homing or determining the commutation offset for a 3 phase brushless motor. If an effective coil position is energized, the motor rotor will normally align itself to the coil position. This is similar to the manner in which a stepping motor operates. Since the rotor location is then known, the commutation offset may then be determined. Alternately if an index mark is available, the effective coil position may be rotated by changing the phase angle until the index mark is detected.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Magnitude of output to apply.

Valid Range is -230 ... +230 PWM units

<A>

Commutation angle to be used.

Units are in Commutation cycles

Only fractional value will be used

Example

3PH0=230 0.5

4PH*<N>*=*<M>* *<A>*

Sets the assigned PWMs of an axis to the specified magnitude and phase angle for a brushless 4 phase motor.

This command is useful for energizing a coil (or effective coil position). This is often required while initial homing or determining the commutation offset for a 4 phase brushless motor. If an effective coil position is energized, the motor rotor will normally align itself to the coil position. This is similar to the manner in which a stepping motor operates. Since the rotor location is then known, the commutation offset may then be determined. Alternately if an index mark is available, the effective coil position may be rotated by changing the phase angle until the index mark is detected.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Magnitude of output to apply.

Valid Range is -250 ... +250 PWM units

<A>

Commutation angle to be used.

Units are in Commutation cycles

Only fractional value will be used

Example

```
4PH0=250 0.5
```

Accel <*N*>=<*A*>

or

Accel <*N*>

Description

Get or Set the max acceleration (for independent moves and jogs)

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<A>

The max acceleration. Units are in *Position units* per sec^2

Example

Accel0=1000.0

ADC<*N*>

Description

Display current ADC (Analog to Digital Converter). Display range -2048 to 2047

Channels 0-3 are ±10V general purpose inputs

Channels 4-7 are Motor Currents

Parameters

<N>

ADC channel

Valid range 0 ... 7

Example

ADC 0

Arc $<X_C>$ $<Y_C>$ $<R_X>$ $<R_Y>$ $<\theta_0>$ $<d\theta>$ $<Z_0>$ $<Z_1>$ $<a>$ $<b>$ $<c>$ $<d>$ $<t_F>$

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. See also KMotion Coordinated Motion. A path through space is defined where *x* and *y* are changing in an elliptical manner and *z* is changing in a linear manner forming a portion of a helix. A parametric equation is defined which describes which portion of the path as well as how as a function of time the path is to be traversed.

Although the Arc command may be sent directly, the Arc command is normally generated automatically to perform a planned trajectory by the coordinated motion library or GCode.

$(\boldsymbol{X_C},\boldsymbol{Y_C})$ - center of circle

$(\boldsymbol{R_X},\boldsymbol{R_Y})$ - *x* radius and *y* radius

$\boldsymbol{\theta_0}$ - initial angle for the beginning of the path

$\boldsymbol{d\theta}$ - amount of angular change for the path

$\boldsymbol{Z_0}$ - initial *Z* position of path

$\boldsymbol{Z_1}$ - final *Z* position of path

3rd order parametric equation where

$$p = \boldsymbol{a}\, t^3 + \boldsymbol{b}\, t^2 + \boldsymbol{c}\, t + \boldsymbol{d}$$

p is the position along the path as a function of time. When *p*=0 the (x,y,z) position will be at the beginning of the path $(\theta= \theta_0$ and $z=z_0)$. When *p*=1 the *(x,y,z)* position will be at the end of the path $(\theta= \theta_0+d\theta,$ and $z=z_1)$.

This motion segment will be performed over a time period of $\boldsymbol{t_F}$, where ***t*** varies from **0** ... $\boldsymbol{t_F}$. Note that it is not necessary that *p* vary over the entire range of 0 ... 1. This is often the case when there may be an acceleration, constant velocity, and deceleration phase phase over the path. ie: ***t*** might vary from 0.0->0.1 where p might vary from 0.3->0.7.

Parameters

<*X*$_C$> - X center of ellipse, units are position units of x axis

<*Y*$_C$> - Y center of ellipse, units are position units of y axis

<*R*$_X$> - X radius of ellipse, units are position units of x axis

<*R*$_Y$> - Y radius of ellipse, units are position units of y axis

<θ_0> - initial theta position on ellipse, radians (0 radians points in the +*x* direction)

<dθ> - change in theta position on ellipse, radians (+ theta causes CCW motion)

<Z_0> - initial Z position on path, units are position units of z axis

<Z_1> - final Z position on path, units are position units of z axis

<a> - parametric equation t^3 coefficient

<b> - parametric equation t^2 coefficient

<c> - parametric equation *t* coefficient

<d> - parametric equation constant coefficient

<t_F> - time for segment

Example (complete unit circle, centered at 0.5,0.5, no Z motion, performed in 10 seconds)

Arc 0.5 0.5 1.0 1.0 0.0 6.28 0.0 0.0 0.0 0.0 0.1 0.0 10.0

Arc*Hex <X_C> <Y_C> <R_X> <R_Y> <θ_0> <dθ> <Z_0> <Z_1> <a> <b> <c> <d> <t_F>*

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. This command is exactly the same as the Arc command above, except all 13 parameters are specified as 32-bit hexadecimal values which are the binary images of 32-bit floating point values. When generated by a program this is often faster, simpler, and more precise than decimal values. See also KMotion Coordinated Motion.

Parameters

See above.

Example (complete unit circle, centered at 0.5,0.5, no Z motion, performed in 10 seconds)

```
Arc 3f000000 3f000000 3f800000 3f800000 0 40c8f5c3 0 0 0 0 40c8f5c3 0 41800000
```

CheckDone<*N*>

Description

Displays:

 1 if axis N has completed its motion
 0 if axis N has not completed its motion
-1 if the axis is disabled

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

CheckDone0

CheckDoneBuf

Description

Displays the status of the Coordinated Motion Buffer. ***KMotion*** contains a Coordinated Motion Buffer where move segments (linear and arcs) and I/O commands may be downloaded and executed in real time.

Displays:

 1 if all coordinated move segments have completed

 0 if all coordinated move segments have not completed

-1 if any axis in the defined coordinate system is disabled

Parameters

None

Example

CheckDoneBuf

CheckDoneGather

Description

Displays the status of a data gather operation. ***KMotion*** contains a mechanism for capturing data from a variety of sources in real time. This mechanism is utilized when capturing data for Bode plots and Step response plots. It is also available for general purpose use. See the data gathering example.

Displays:

1 if data gather is completed

0 if data gather has not completed

Parameters

None

Example

CheckDoneGather

CheckDoneXYZA

Description

Displays status of a commanded MoveXYZA command. See also DefineCS.

Displays:

 1 if all axes in the defined coordinate system have completed their motion
 0 if any axis in the defined coordinate system has not completed its motion
-1 if any axis in the defined coordinate system is disabled

Parameters

None

Example

CheckDoneXYZA

ClearBit<*N*>

Description

Clears an actual I/O bit or virtual I/O bit. Note that actual IO bits must be previously defined as an output, see SetBitDirection

Parameters

<N>

Bit number specified as a decimal number. Valid range 0...31 for actual hardware I/O bits. Valid range of 32...63 for virtual I/O bits.

Example

ClearBit0

ClearBitBuf<*N*>

Description

Inserts into the coordinated move buffer a command to clear an IO bit N(0..30) or a Virtual IO bit (32..63) (actual IO bits must be defined as an output, see SetBitDirection)

Parameters

<N>

Bit Number to clear. Valid Range 0...63.

Example

ClearBitBuf0

ClearFlashImage

Description

Prepare to download FLASH firmware image. Sets entire RAM flash image to zero

Parameters

None.

Example

ClearFlashImage

CommutationOffset<N>=<*X*>

or

CommutationOffset<N>

Description

Get or Set 3 or 4 phase commutation offset. When brushless commutation is performed, the desired Output Magnitude is distributed and applied to the various motor coils as a function of position. The commutation offset shifts the manner in which the Output Magnitude is applied.

For a 3 phase brushless output mode, commutation offset is used in the following manner.

PhaseA = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π)

PhaseB = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π + 2π/3)

PhaseC = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π + 4π/3)

For a 4 phase brushless output mode, commutation offset is used in the following manner.

PhaseA = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π)
PhaseB = OutputMagnitude * cos((Position+CommutationOffset)*invDistPerCycle*2π)

See also invDistPerCycle and Configuration Parameters.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<X>

Offset in units of Position.

Example

CommutationOffset0=100.0

D<*N*>=<*M*>

or

D<N>

Description

Get or Set PID derivative Gain.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Derivative Gain value. The units of the derivative gain are in Output Units/Position Units x *Servo Sample Time*.

Example

D0=10.0

DAC*<N> <M>*

Description

DAC to value. DACs 0...3 have ±10 Volt ranges, DACs 4...7 have 0...4 Volt ranges. See also Analog Status Screen.

Parameters

<N>

DAC channel to set. Valid Range 0...7.

<M>

DAC value to set in counts. Valid Range -2048...2047.

Example

DAC0=2000

DeadBandGain<*N*>=<*M*>

or

DeadBandGain<*N*>

Description

Get or Set gain while error is within the deadband range. See DeadBand Description. See Servo Flow Diagram.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Gain to be applied. A value of 1.0 will have normal gain while within the deadband. A value less than 1.0 will have reduced gain within the deadband.

Example

DeadBandGain0=0.5

DeadBandRange<*N*>=<*M*>

or

DeadBandRange<*N*>

Description

Get or Set range where deadband gain is to be applied. See DeadBand Description. See Servo Flow Diagram.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

±Range in Position units,

Example

DeadBandRange0=1.0

DefineCS*<X> <Y> <Z> <A>*

or

DefineCS

Description

Set or get the defined X Y Z A coordinate system axis assignments. Unused axis are assigned an axis channel of -1.

See also Coordinated Motion.

Parameters

<X>

Assigned Axis channel number for *X*. Valid range -1 ... 3.

Use -1 if axis is not defined.

<Y>

Assigned Axis channel number for *Y*. Valid range -1 ... 3.

Use -1 if axis is not defined.

<Z>

Assigned Axis channel number for *Z*. Valid range -1 ... 3.

Use -1 if axis is not defined.

<A>

Assigned Axis channel number for *A*. Valid range -1 ... 3.

Use -1 if axis is not defined.

Example

DefineCS 0 1 -1 -1

Dest<*N*>=<*M*>

or

Dest<*N*>

Description

Set or get the last commanded destination for an axis. The Dest (destination) is normally set (or continuously updated) as the result of a motion command (Move, Jog, or Coordinated motion) , but may also be set with this command if no motion is in progress.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Value to set in Position units. Valid range - any.

Example

Dest0=100

or

Dest0

DisableAxis*<N>*

Description

Kill any motion and disable motor. Any associated output PWM channels for the axis will be set to 0R mode.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

DisableAxis0

Echo <*S*>

Description

Echo character string back to the Console Screen.

Parameters

<S>

Any character string < 80 characters

Example

Echo Hello

EnableAxis<*N*>

Description

Set an Axis' *destination* to the Current Measured Position and enable the axis. See also EnableAxisDest to explicitly set the desired destination for the axis. Note for a MicroStepper Axis (which normally has no measured position) this command will leave the Axis' destination unchanged. .

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

Enable0

EnableAxisDest<*N*> <*M*>

Description

Set an Axis' *destination* to the specified position and enable the axis. See also EnableAxis to set the desired destination to the current measured position***.***

<N>

Selected Axis for command. Valid range 0...7.

<M>

Destination for the axis. Position units. Valid range - any.

Example

EnableAxisDest0 1000.0

Enabled*<N>*

Description

Display whether the specified axis is enabled, 1 - if currently enabled, 0 - if not enabled.
Note: to enable an axis use EnableAxis or EnableAxisDest.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

Enabled0

EntryPoint*<N> <H>*

Description

Set execution start address of user thread to specified address. This operation if normally performed automatically when downloading a user program.

Parameters

<N>

User Thread number to set. Decimal number. Valid range 1...7.

<H>

Start address. 32 bit Hex number.

Example

Entrypoint1 80030000

ExecBuf

Description

Execute the contents of the coordinated motion buffer. Use CheckDoneBuf to determine when the buffer has been fully executed. See also Coordinated Motion.

Parameters

None

Example

ExecBuf

ExecTime

Description

Displays the amount of the Coordinated Motion Buffer that has been already executed in terms of *Time*. ***KMotion*** contains a Coordinated Motion Buffer where move segments (linear and arcs) and I/O commands may be downloaded and executed in real time. This command is useful for determining how long before the Coordinated Motion Buffer will complete. For example, if a number of segments have been downloaded where their total execution time is 10 seconds, and they are currently in progress of being executed, and the ExecTime command reports that 8 seconds worth of segments have been executed, then the remaining time before the queue completes (or starves for data) would be 2 seconds. This command is useful for applications where it is important not to download data too far ahead so changes to the Trajectory may be made. The value returned is a floating point decimal value in Seconds with 3 decimal places. If the Coordinated Motion has already completed the amount of time will be a negative value whose magnitude is the total time that was executed. See also Coordinated Motion.

Displays:

Executed time in seconds as a floating point decimal number with 3 decimal places

ie. 10.123

If the buffer has already completed the value will be negative

ie. -10.123

Parameters

None

Example

ExecTime

Execute<*N*>

Description

Begin execution of thread. Execution begins at the previously specified thread entry point. See also C Program Screen.

Parameters

<N>

Thread number to begin execution. Decimal number. Valid range 1...7.

Example

Execute1

FFAccel<*N*>=<*M*>

or

FFAccel<*N*>

Description

Set or get Acceleration *feed forward* for axis.

See also feed forward tuning.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Feed forward value. units are in Output units per Input Units per sec^2.

Example

FFAccel0=100.0

or

FFAccel0

FFVel<*N*>=<*M*>

or

FFVel<*N*>

Description

Set or get Velocity *feed forward* for axis.

See also feed forward tuning.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Feed forward value. units are in Output units per Input Units per sec.

Example

FFVel0=100.0

or

FFVel0

Flash

Description

Flash current *user programs,* persistent memory area*,* all axes configurations, tuning, and filter parameters to non-volatile memory. The entire state of the ***KMotion*** is saved to FLASH memory. Any active user programs will be paused during the flash operation

Parameters

None

Example

Flash

GatherMove<*N*> <*M*> <L>

Description

Performs a profiled move on an axis of the specified distance while gathering the specified number of points of data. This command is used while gathering data for the Step Response Screen plots.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Distance to move. Units are Position Units. Valid Range - any.

<L>

Number of servo samples to gather. Valid Range - 1...40000

Example

GatherMove0 1000.0 2000

GatherStep<*N*> <*M*> <L>

Description

Performs a step on an axis of the specified distance while gathering the specified number of points of data. This command is used while gathering data for the Step Response Screen plots.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Distance to step. Units are Position Units. Valid Range - any.

<L>

Number of servo samples to gather. Valid Range - 1...40000

Example

GatherStep0 1000.0 2000

GetBitDirection<*N*>

Description

Displays whether an IO bit N (0..30) is defined as input (0) or output (1)

Parameters

<N>

I/O bit number. Valid range 0...30

Example

GetBitDirection0

GetGather <*N*>

Description

Upload N data points from previous GatherMove or GatherStep command. Captured commanded destination, measured position, and output are uploaded as hex values (that represent binary images of 32-bit floating point values). Eight samples (24 values) per line.

Parameters

<N>

Number of points to upload. Valid range 1...40000.

Example

GetGather 1000

GetGatherDec*<N>*

Description

Reads a single word from the Gather Buffer at the specified offset. A single 32-bit value displayed as a signed decimal integer number will be displayed.

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

Example

GetGatherDec 1000

GetGatherHex*<N> <M>*

Description

Reads multiple words from the Gather Buffer beginning at the specified offset. Hexadecimal values will be displayed that will represent binary images of the contents of the gather buffer as 32 bit words.

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

<M>

Number of 32 bit words to display. Decimal integer. Valid range 1...2000000

Example

GetGatherHex 0 100

GetInject<*N*> <*M*>

Description

Display results of signal injection and gathering. Bode Plot measurement involves injecting a signal and measuring the response for each of N_CPLX (2048) samples. This command gets the result from the injection. 3 values per sample are uploaded. Injection value, position response (relative to destination), and servo output. All 3 values are printed as hexadecimal values which represent the image of a 32-bit floating point value. 8 samples (24 hex values) are printed per line.

Parameters

None

Example

GetInject

GetPersistDec<*N*>

Description

Read a single word from the *Persist Array* at the specified offset a single 32-bit value displayed as a **signed decimal number**. The persist array is a general purpose array of N_USER_DATA_VARS (100) 32-bit values that is accessible to the host as well as ***KMotion*** C Programs. It may be used to share parameters, commands, or information between programs.

C Programs may access this array as the integer array:

```
persist.UserData[n];
```

It also resides in the ***KMotion*** Persist memory structure so that if memory is flashed, the value will be present at power up.

See also GetPersistHex, SetPersistDec, SetPersistHex

Parameters

<N>

Offset into the integer array. Valid range 0...99.

Example

GetPersistDec 10

GetPersistHex<*N*>

Description

Read a single word from the *Persist Array* at the specified offset a single 32-bit value displayed as an **unsigned hexadecimal number**. The persist array is a general purpose array of N_USER_DATA_VARS (100) 32-bit values that is accessible to the host as well as ***KMotion*** C Programs. It may be used to share parameters, commands, or information between programs.

C Programs may access this array as the integer array:

`persist.UserData[n];`

It also resides in the ***KMotion*** Persist memory structure so that if memory is flashed, the value will be present at power up.

See also GetPersistDec, SetPersistDec, SetPersistHex

Parameters

<N>

Offset into the integer array. Valid range 0...99.

Example

GetPersistHex 10

GetStatus

Description

Upload Main Status record in hex format. ***KMotion*** provides a means of quickly uploading the most commonly used status. This information is defined in the PC-DSP.h header file as the MAIN_STATUS structure. The entire stucture is uploaded as a binary image represented as 32-bit hexadecimal values.

Parameters

None

Example

GetStatus

I<*N*>=<*M*>

or

I<*N*>

Description

Get or Set PID Integral Gain.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Integral Gain value. The units of the derivative gain are in Output Units x Position Units x *Servo Sample Time*.

Example

I0=10.0
or
I0

IIR<*N*> <*M*>=<*A1*> <*A2*> <*B0*> <*B1*> <*B2*>

or

IIR<N> <M>

Description

Set or get IIR Z domain servo filter.

See also IIR Filter Screen

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Filter number for axis. Valid range 0...2.

<A1> <A2> <B0> <B1> <B2>

Filter coefficients represented as floating point decimal values.

Example

IIR0 0=1.5 2.5 -3.5 4.5 5.5

or

IIR0 0

Inject<*N*> <*F*> <A>

Description

A Inject random stimulus into an axis with the specified cutoff frequency and amplitude. Useful for generating Bode plots.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<F>

Cuttoff Frequency in Hz. Valid range - any.

<A>

Amplitude in position units. Valid range - any.

Example

Inject0 100.0 20.0

InputChan<*M*> <*N*>=<*C*>

or

InputChan<*M*> <*N*>

Description

Get or Set the first or second *Input Channel* of an axis. See description of this parameter on the Configuration Screen.

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<C>

Channel number to assign. Valid range 0...7.

Example (set first input channel of axis 3 to 3)

InputChan0 3=3

or

InputChan0 3

InputGain<*M*> <*N*>=<*G*>

or

InputGain<*M*> <*N*>

Description

Set or get first or second Input Gain of an axis. See description of this parameter on the Configuration Screen.

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<C>

Input Gain. Valid range - any.

Example

InputGain0 3=1.0

InputMode<*N*>=<*M*>

or

InputMode<*N*>

Description

Set or get the position input mode for an axis. See description of this parameter on the Configuration Screen.

Valid modes are:

ENCODER_MODE 1

ADC_MODE 2

RESOLVER_MODE 3

USER_INPUT_MODE 4

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Mode. Valid range 1...4

Example

SetInputMode0=1

InputOffset*<M>* *<N>*=*<O>*

or

InputOffset*<M>* *<N>*

Description

Set or get first or second Input Offset of an axis. See description of this parameter on the Configuration Screen.

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<O>

Input Offset. Valid range - any.

Example

InputOffset0 3=0.0

InvDistPerCycle<*N*>=<*X*>

Description

Get or Set distance per cycle (specified as an inverse) of an axis. May specify the cycle of either a Stepper of Brushless Motor.

See description of this parameter on the Configuration Screen.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<X>

Inverse (reciprocal) of distance for a complete cycle. Inverse position units. Should be specified exactly or with very high precision (double precision accuracy ~ 15 digits). Valid range - any.

Example

InvDistPerCycle0=0.05

Jerk<*N*>=<*J*>

or

Jerk<*N*>

Description

Get or Set the max jerk (for independent moves and jogs)

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<J>

The max Jerk. Units are in *Position units* per sec^3

Example

Jerk0=10000.0

Jog<*N*>=<*V*>

Description

Move at constant velocity. Uses Accel and Jerk parameters for the axis to accelerate from the current velocity to the specified velocity. Axis should be already enabled. Specify zero velocity to decelerate to a stop.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<V>

new Velocity in position units/second. Valid range - any.

Example

Jog0=-200.5

Kill<*N*>

Description

Stop execution of a user thread.

Parameters

<N>

Thread to halt. Valid range 1..7

Example

Kill0

Lead<*N*>=<*M*>

or

Lead<N>

Description

Set or get Lead Compensation for an axis. Lead Compensation is used to compensate for lag caused by motor inductance.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Lead Compensation. Valid range - any.

Example

Lead0=10.0
or
Lead0

LimitSwitch<*N*>=<*H*>

Description

Configures Limit Switch Options. Specify Hex value where:

See also Configuration Screen.

Parameters
<N>

Selected Axis for command. Valid range 0...7.

<H>

32-bit hexadecimal value:

Bit 0 1=Stop Motor on Neg Limit, 0=Ignore Neg limit

Bit 1 1=Stop Motor on Pos Limit, 0=Ignore Pos limit

Bit 2 Neg Limit Polarity 0=stop on high, 1=stop on low

Bit 3 Pos Limit Polarity 0=stop on high, 1=stop on low

Bits 4-7 Action - 0 Kill Motor Drive

1 Disallow drive in direction of limit

2 Stop movement

Bits 16-23 Neg Limit Bit number

Bits 24-31 Pos Limit Bit number

Example

LimitSwitch2 0C0D0003

Linear <X_0> <Y_0> <Z_0> <A_0> <X_1> <Y_1> <Z_1> <A_1> <*a*> <*b*> <*c*> <*d*> <t_F>

Description

Place linear (in 4 dimensions) interpolated move into the coordinated motion buffer. See also KMotion Coordinated Motion. A path through space is defined where *x, y, z, and A* are changing in a linear manner. A parametric equation is defined which describes which portion of the path as well as how as a function of time the path is to be traversed.

Although the Linear command may be sent directly, the Linear command is normally generated automatically to perform a planned trajectory by the coordinated motion library or GCode.

$(\boldsymbol{X_0},\boldsymbol{Y_0},\boldsymbol{Z_0},\boldsymbol{A_0})$ - beginning of path

$(\boldsymbol{X_1},\boldsymbol{Y_1},\boldsymbol{Z_1},\boldsymbol{A_1})$ - end of path

3rd order parametric equation where

$$p = \boldsymbol{a}\, t^3 + \boldsymbol{b}\, t^2 + \boldsymbol{c}\, t + \boldsymbol{d}$$

p is the position along the path as a function of time. When *p*=0 the (x,y,z,A) position will be at the beginning of the path. When *p*=1 the *(x,y,z,A)* position will be at the end of the path.

This motion segment will be performed over a time period of $\boldsymbol{t_F}$, where ***t*** varies from **0** ... $\boldsymbol{t_F}$. Note that it is not necessary that *p* vary over the entire range of 0 ... 1. This is often the case when there may be an acceleration, constant velocity, and deceleration phase over the path. ie: ***t*** might vary from 0.0->0.1 where p might vary from 0.3->0.7.

Parameters

<X_0> - X begin point

<Y_0> - Y begin point

<Z_0> - Z begin point

<A_0> - A begin point

<X_0> - X end point

<Y_1> - Y end point

<Z_1> - Z end point

<A_1> - A end point

<θ_1> - initial theta position on ellipse, radians (0 radians points in the +*x* direction)

<*a*> - parametric equation t^3 coefficient

<b> - parametric equation t^2 coefficient

<c> - parametric equation *t* coefficient

<d> - parametric equation constant coefficient

<t_F> - time for segment

Example

Linear 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0

LinearHex <X_0> <Y_0> <Z_0> <A_0> <X_1> <Y_1> <Z_1> <A_1> <*a*> <*b*> <*c*> <*d*> <t_F>

Description

Place linear (in 4 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the Linear command above, except all 13 parameters are specified as 32-bit hexadecimal values which are the binary images of 32-bit floating point values. When generated by a program this is often faster, simpler, and more precise than decimal values. See also KMotion Coordinated Motion.

Parameters

See above.

Example

LinearHex 0 0 0 0 3F800000 3F800000 3F800000 3F800000 0 0 3F800000 0 3F800000

LoadData <*H*> <*N*>

<B> <B> <B> <B> <B> ...

Description

Store data bytes into memory beginning at specified address for N bytes. The data must follow with up to N_BYTES_PER_LINE (64) bytes per line. This command is normally only used by the COFF loader. Since this command spans several lines, it may only be used programatically in conjunction with a KMotionLock or WaitToken command so that it is not interrupted.

Parameters

<H>

32-bit hexadecimal address

<N>

Number of bytes to follow and to be stored

<B> <B> <B> <B> <B> ...

Bytes to store. 2 hexadecimal digits per byte, separated with a space.

Example

```
LoadData 80030000 4
FF FF FF FF
```

LoadFlash<*H*> <*N*>

<B> <B> <B> <B> <B> ...

Description

Store data into FLASH image. Only by ***KMotion*** for downloading a new firmware version. Store data bytes into memory beginning at specified address for N bytes. The data must follow with up to N_BYTES_PER_LINE (64) bytes per line. This command is normally only used by the COFF loader. Since this command spans several lines, it may only be used programmatically in conjunction with a KMotionLock or WaitToken command so that it is not interrupted.

Parameters

<H>

32-bit hexadecimal address

<N>

Number of bytes to follow and to be stored

<B> <B> <B> <B> <B> ...

Bytes to store. 2 hexadecimal digits per byte, separated with a space.

Example

LoadFlash FF00 4
FF FF FF FF

MaxErr<*N*>=<*M*>

or

MaxErr<*N*>

Description

Set or get *Maximum Error* for axis (Limits magnitude of error entering PID).
See Servo Flow Diagram and Step Response Screen for more information.
Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum Error. Valid range - any positive value. Set to a large value to disable.

Example

MaxErr0=100.0
or
MaxErr0

MaxFollowingError<*N*>=<*M*>

or

MaxFollowingError<*N*>

Description

Set or get the maximum allowed following error before disabling the axis.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum Following Error. Valid range - any positive value. Set to a large value to disable.

Example

MaxFollowingError0=100.0

or

MaxFollowingError0

MaxI<*N*> <*M*>

Description

Set or get Maximum Integrator "wind up" for axis. Integrator saturates at the specified value. See also Servo Flow Diagram and Step Response Screen for further information.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum Integrator value. Valid range - any positive value. Set to a large value to disable.

Example

MaxI0=100.0

or

MaxI0

MaxOutput<*N*>=<*M*>

or

MaxOutput<*N*>

Description

Set or get Maximum Output for an axis. Limits magnitude of servo output. Output saturates at the specified value.

See also Servo Flow Diagram and Step Response Screen for further information.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum output value. Valid range - any positive value. Set to a large value to disable.

Example

MaxOutput0=100.0

or

MaxOutput

Move<*N*>=<*M*>

Description

Move axis to absolute position. Axis should be already enabled. Uses Vel, Accel and Jerk parameters for the axis to profile a motion from the current state to the specified position.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

new *position* in position units. Valid range - any.

Example

Move0=100.1

Move*AtVel*<*N*>=<*M*> <V>

Description

Move axis to absolute position at the specified Velocity. Axis should be already enabled. Uses Accel and Jerk parameters for the axis to profile a motion from the current state to the specified position.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

new *position* in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

Example

MoveAtVel0=100.1 30.0

MoveRel<*N*>=<*M*>

Description

Move axis relative to current destination. Same as Move command except specified motion is relative to current destination.

Axis should be already enabled. Uses Vel, Accel and Jerk parameters for the axis to profile a motion from the current state to the specified position.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Distance to move in position units. Valid range - any.

Example

MoveRel0=100.1

Move*RelAtVel*<*N*>=<*M*> <V>

Description

Move axis relative to current destination at the specified Velocity. Same as MoveAtVel command except specified motion is relative to current destination. Axis should be already enabled. Uses Accel and Jerk parameters for the axis to profile a motion from the current state to the specified position.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

new *position* in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

Example

MoveRelAtVel0=100.1 30.0

MoveXYZA *<X> <Y> <Z> <A>*

Description

Move the 4 axes defined to be x,y,z,A (each axis moves independently). The defined coordinate system determines which axes channels are commanded to move.

Parameters

<X>

Position to move x axis. Valid range - any.

<Y>

Position to move x axis. Valid range - any.

<Z>

Position to move x axis. Valid range - any.

<A>

Position to move x axis. Valid range - any.

Example

MoveXYZA 100.1 200.2 300.3 400.4

OpenBuf

Description

Clear and open the buffer for coordinated motion.

Parameters

None

Example

OpenBuf

OutputChan<*M*> <*N*>=<*C*>

or

OutputChan<*M*> <*N*>

Description

Get or Set the first or second *Output Channel* of an axis. See description of this parameter on the Configuration Screen.

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<C>

Channel number to assign. Valid range 0...7.

Example (set first output channel of axis 3 to 3)

OutputChan03=3

OutputMode<*N*>=<*M*>

or

OutputMode<*N*>

Description

Set or get the position output mode for an axis. See description of this parameter on the Configuration Screen.

Valid modes are:

MICROSTEP_MODE 1

DC_SERVO_MODE 2

BRUSHLESS_SERVO_MODE 3

DAC_SERVO_MODE 4

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Mode. Valid range 1...4

Example

SetOutputMode0=1

P<*N*>=<*M*>

or

P<N>

Description

Get or Set PID Proportional Gain.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Proportional Gain value. The units of the derivative gain are in Output Units/Position Units.

Example

P0=10.0

Pos<*N*>=<*P*>

or

Pos<N>

Description

Set or get the measured position of an axis. Note setting the current position may effect the commutation of any motors based on the position (an adjustment in the commutation offset may be required).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<P>

value to be stored into the current position. units are position units. Valid range - any.

Example

Pos0=100.0

ProgFlashImage

Description

Program entire FLASH image, downloaded using LoadFlash commands, to FLASH Memory.

Parameters

None

Example

ProgFlashImage

PWM<*N*>=<*M*>

Description

Set PWM channel to locked anti-phase mode and to specified value.
See PWM Description and Analog Status Screen.

Parameters

<N>

PWM channel number. Valid range 0...7

<M>

PWM value. Valid range -255...255.

Example

PWM0=-99

PWMR*<N>*=*<M>*

Description

Set PWM channel to recirculate mode and to specified value.
See PWM Description and Analog Status Screen.

Parameters

<N>

PWM channel number. Valid range 0...7

<M>

PWM value. Valid range -511...511.

Example

PWMR0=-99

ReadBit<*N*>

Description

Displays whether an actual hardware I/O bit N (0...30) or Virtual IO bit (32...63) is high (1) or low (0) . A bit defined as an output (See SetBitDirection) may also be read back.

Parameters

<N>

Bit number to read. Valid range - 0...63

Example

ReadBit0

Reboot!

Description

Causes complete power up reset and re-boot from flash memory.

Parameters

None

Example

Reboot!

SetBit<*N*>

Description

Sets an actual hardware I/O bit N (0...30) or Virtual IO bit (32...63) to high (1) .

Parameters

<N>

Bit number to set. Valid range 0...63

Example

SetBit0

SetBitBuf<*N*>

Description

Inserts into the coordinated move buffer a command to set an I/O bit N(0...30) or Virtual IO bits (32...63) (actual IO bits must be defined as an output, see SetBitDirection)

Parameters

<N>

Bit number to set. Valid range 0...63

Example

SetBitBuf0

SetBitDirection<*N*>=<*M*>

Description

Defines the direction of an I/O bit to be an input or output.

See also Digital I/O Screen.

Parameters

<N>

Bit number to assign. Valid range 0...30

<M>

Direction 0 = input, 1 = output

Example

SetBitDirection0=1

SetGatherDec <*N*> <*M*>

Description

Writes a single word to the Gather Buffer at the specified offset. A single 32-bit value specified as a signed decimal integer number will be stored.

The corresponding value may be accessed by a ***KMotion*** user program using the pointer : **gather_buffer**. This pointer should be cast as an integer pointer in order to reference values as integers and to use the same index.

See also GetGatherDec, GetGatherHex, SetGatherHex

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

<M>

Value to be stored. Valid range -2147483648...2147483647

Example

SetGatherDec 1000 32767

SetGatherHex*<N>* *<M>*

<H> <H> <H> . . .

Description

Writes a *multiple* words to the Gather Buffer beginning at the specified offset. 32-bit values specified as a unsigned hexadecimal numbers must follow with 8 words per line separated with spaces. Since this command spans several lines, it may only be used programmatically in conjunction with a KMotionLock or WaitToken command so that it is not interrupted.

The corresponding values may be accessed by a ***KMotion*** user program using the pointer : **gather_buffer**. This pointer should be cast as an integer pointer in order to reference values as integers and to use the same index.

See also GetGatherDec, GetGatherHex, SetGatherDec

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

<M>

Number of value to be stored, specified as a decimal number. Valid range 0...19999999

<H> <H> <H> . . .

Values to be stored. Specified as unsigned Hexadecimal values. Valid range 0...FFFFFFFF.

Example

```
SetGatherHex 0 3
FFFFFFFF FFFFFFFF FFFFFFFF
```

SetPersistDec <*O*> <*D*>

Description

Write a single word into the Persistent *UserData* Array.   Persistent UserData Array is a general purpose array of 100 32-bit words that may be used as commands, parameters, or flags between any host applications or ***KMotion*** user programs. The array resides in a *persistent* memory area, so that if a value is set as a parameter and the User Programs are *flashed*, the value will persist permanently.

The corresponding value may be accessed by a ***KMotion*** user program as the integer variable : persist.UserData[offset]***.***

See also GetPersistDec, GetPersistHex, SetPersistHex

Parameters

<O>

Offset into the user data array specified as a decimal number. Valid Range 0 ... 99.

<D>

Value to be written to the array. Specified a signed decimal number. Valid Range

-2147483648 ... 2147483647

Example

SetPersistDec 10 32767

SetPersistHex <*O*> <*H*>

Description

Write a single word into the Persistent *UserData* Array.   Persistent UserData Array is a general purpose array of 100 32-bit words that may be used as commands, parameters, or flags between any host applications or ***KMotion*** user programs. The array resides in a *persistent* memory area, so that if a value is set as a parameter and the User Programs are *flashed*, the value will persist permanently.

The corresponding value may be accessed by a KMotion user program as the integer variable : persist.UserData[offset].

See also GetPersistDec, GetPersistHex, SetPersistDec.

Parameters

<O>

Offset into the user data array specified as a decimal number. Valid range 0 ... 99.

<H>

Value to be written to the array. Specified an unsigned hexadecimal number. Valid range

0...FFFFFFFF

Example

SetPersistHex 10 FFFFFFFF

SetStartupThread*<N>* *<M>*

Description

Defines whether a user thread is to be launched on power up.

Parameters

<N>

Selected User Thread. Valid range 1...7

<M>

Mode : 1=start on boot, 0=do not start on boot.

Example

SetStartupThread0 1

SetStateBit*<N>*=*<M>*

Description

Sets the state of an actual hardware I/O bit N (0...30) or Virtual IO bit (32...63) to either low (0) or high (1) . Actual I/O bits must be defined as an output, see SetBitDirection.

Parameters

<N>

Bit number to set. Valid range 0...63

<M>

State. Valid range 0...1

Example

SetStateBit0=1

SetStateBitBuf<*N*>=<*M*>

Description

Inserts into the coordinated move buffer a command to set the state of an I/O bit N(0...30) or Virtual IO bits (32...63) (actual IO bits must be defined as an output, see SetBitDirection)

Parameters

<N>

Bit number to set. Valid range 0...63

<M>

State. Valid range 0...1

Example

SetBitBuf0

SetStateBitBuf0=1

StepperAmplitude<*N*>=<*M*>

or

StepperAmplitude<*N*>

Description

Set or get the nominal output magnitude used for axis if in MicroStepping Output Mode to the specified value. This will be the output amplitude when stopped or moving slowly. If Lead Compensation is used, the amplitude while moving may be higher.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

PWM Stepper Amplitude. Valid range 0...255

Example

StepperAmplitude0=250

Vel<*N*>=<*V*>

or

Vel <*N*>

Description

Get or Set the max velocity for independent moves.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<V>

The max velocity. Units are in *Position units* per sec

Example

Vel0=100.0

Version

Description

Display DSP Firmware Version and Build date in the form:.

KMotion 2.22 Build 22:26:57 Feb 16 2005

Note it is important that when C Programs are compiled and linked, they are linked to a firmware file, DSP_KMotion.out, that matches the firmware in the KMotion where they will execute.

Parameters

None

Example

Version

Zero<*N*>

Description

Clear the measured position of axis.   Note for an axis that uses the *Position* to perform brushless motor commutation, the commutation offset may be required to be adjusted whenever the position measurement is changed.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

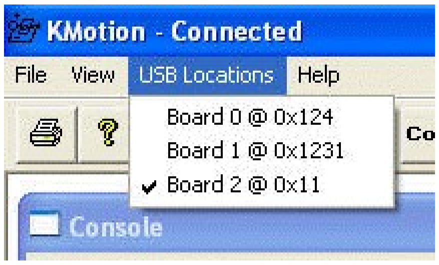
Example

Zero0

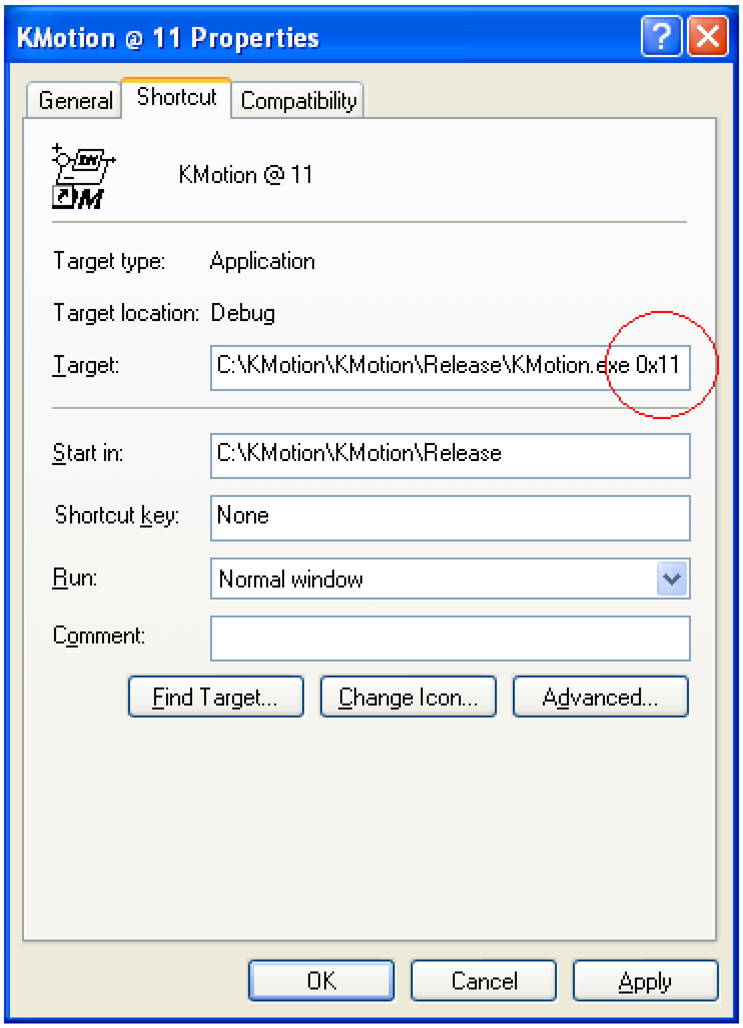
Using Multiple KMotion Boards

The ***KMotion*** Driver Library allows multiple PC processes (applications), each running multiple threads of execution, to communicate with multiple ***KMotion*** boards simultaneously. Each ***KMotion*** board is identified by a USB *location identifier* where it is connected. A USB location identifier is a 32 bit integer. USB devices are arranged in a tree structure of hubs and nodes. Each hexadecimal digit of the USB location specifies a branch in the tree structure. For the purposes of the ***KMotion*** Driver Library, a USB location identifier may simply be considered a unique integer that will remain the same as long as the structure of the USB tree is not altered. Adding or removing USB devices will not change the location of a connected ***KMotion*** board.

Selecting the USB Locations menu of the ***KMotion*** Setup and Tuning application, will display a list of all currently connected ***KMotion*** boards. The Checkmark indicates which board the application is currently communicating with. To switch to a different board, select the USB location from the list.



When launching the ***KMotion*** Setup and Tuning application, a command line parameter may be specified to connect to a specific USB location (see below on how to setup a shortcut to connect to a specific location). Multiple shortcuts may be setup to connect to individual boards.

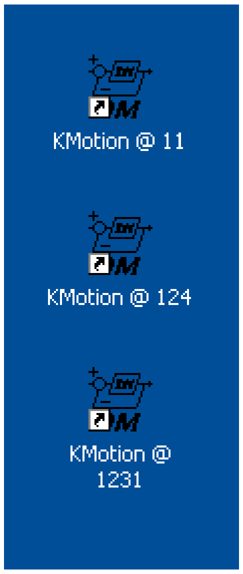


KMotion @ 11 Properties

General | Shortcut | Compatibility

KMotion @ 11

Target type:	Application
Target location:	Debug
Target:	C:\KMotion\KMotion\Release\KMotion.exe 0x11
Start in:	C:\KMotion\KMotion\Release
Shortcut key:	None
Run:	Normal window
Comment:	



The ***KMotion*** Driver Library has a function to list the USB locations of all currently connected ***KMotion*** boards. See:

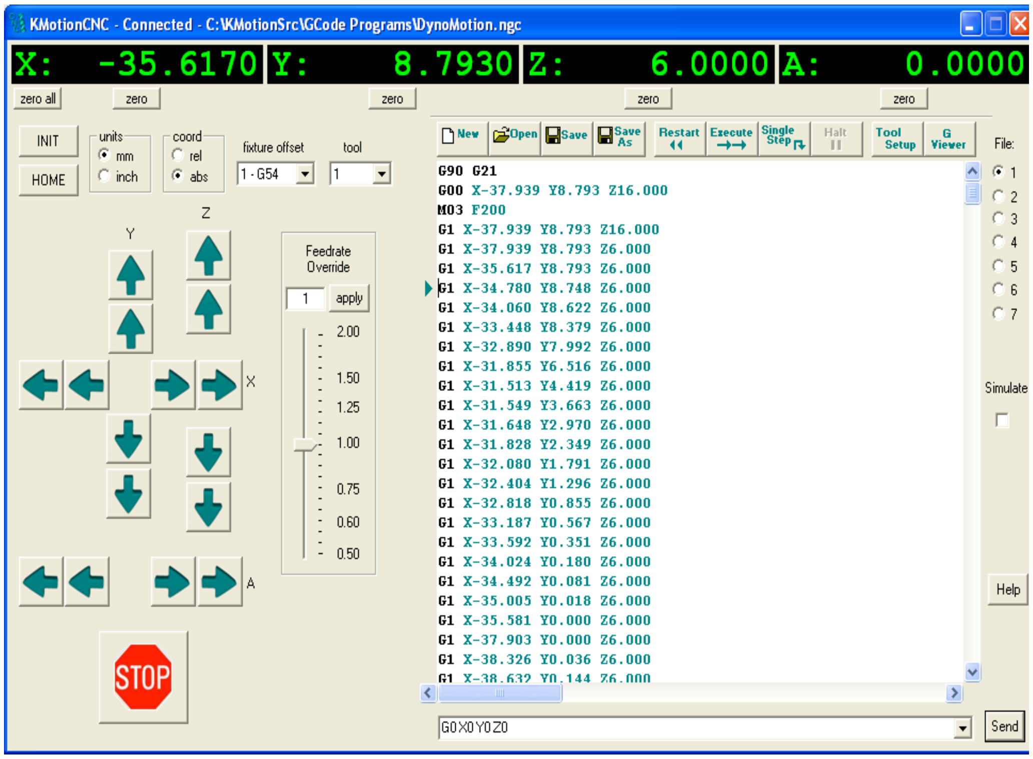
`int ListLocations(int *nlocations, int *list);`

When making ***KMotion*** Driver Library function calls specify the USB *location identifier* of the desired board as the board parameter shown in the example below. Specifying a board value of 0 may be used if there is only one board in a particular system. This will result in a connection to the first available board.

```
int CKMotionDLL::WriteLineReadLine(int board, const char *s, char *response)
```

G Code Quick Reference

Other KMotionCNC Screens
G Code Viewer Screen
G Code Viewer Setup Screen
Tool Setup Screen
see also ***KMotion OnLine Help***



```
G90 G21
G00 X-37.939 Y8.793 Z16.000
M03 F200
G1 X-37.939 Y8.793 Z16.000
G1 X-37.939 Y8.793 Z6.000
G1 X-35.617 Y8.793 Z6.000
G1 X-34.780 Y8.748 Z6.000
G1 X-34.060 Y8.622 Z6.000
G1 X-33.448 Y8.379 Z6.000
G1 X-32.890 Y7.992 Z6.000
G1 X-31.855 Y6.516 Z6.000
G1 X-31.513 Y4.419 Z6.000
G1 X-31.549 Y3.663 Z6.000
G1 X-31.648 Y2.970 Z6.000
G1 X-31.828 Y2.349 Z6.000
G1 X-32.080 Y1.791 Z6.000
G1 X-32.404 Y1.296 Z6.000
G1 X-32.818 Y0.855 Z6.000
G1 X-33.187 Y0.567 Z6.000
G1 X-33.592 Y0.351 Z6.000
G1 X-34.024 Y0.180 Z6.000
G1 X-34.492 Y0.081 Z6.000
G1 X-35.005 Y0.018 Z6.000
G1 X-35.581 Y0.000 Z6.000
G1 X-37.903 Y0.000 Z6.000
G1 X-38.326 Y0.036 Z6.000
G1 X-38.632 Y0.144 Z6.000
```

```
G0X0Y0Z0
```

(Click on Image to Jump to related help)

KMotionCNC allows the user to edit, execute, and view G Code Programs. GCode is a historical language for defining Linear/Circular/Helical Interpolated Motions often used to program numerically controlled machines (CNC Machines).

G Codes

G0 X3.5 Y5.0 Z1.0 A2.0 (Rapid move)

G1 X3.5 Y5.0 Z1.0 A2.0(linear move)

G2 X0.0 Y0.5 I0 J0.25 (CW Arc move)

G3 X0.0 Y0.5 I0 J0.25 (CCW Arc move)

G4 P0.25

(Dwell seconds)

G10L2Pn
G10L2P1X0Y0Z0

(Set Fixture Offset #n)

G20 Inch units

G21 mm units

G28 Move to Reference Position #1

G30 Move to Reference Position #2

G40 Tool Comp Off

G41 Tool Comp On

Left of Contour)

G42 Tool Comp On (Right of Contour)

G43 Hn

(Tool #n length comp On)

G49 (Tool length comp off)

See the Quick Reference at left for commonly used G Code commands.

G53 Absolute Coord

G54 Fixture Offset 1

G55 Fixture Offset 2

G56 Fixture Offset 3

G57 Fixture Offset 4

G58 Fixture Offset 5

G59 Fixture Offset 6

G59.1 Fixture Offset 7

G59.2 Fixture Offset 8

G59.3 Fixture Offset 9

G90 Absolute Coordinates

G91 Relative Coordinates

G92 Set Global Offset Coordinates G92 X0Y0 Z0

M Codes:

M0 (Program Stop)

M2 (Program End)

M3 Spindle CW

M4 Spindle CCW

M5 Spindle Stop

M6 Tool Change

M7 Mist On

M8 Flood On

M9 Mist/Flood Off

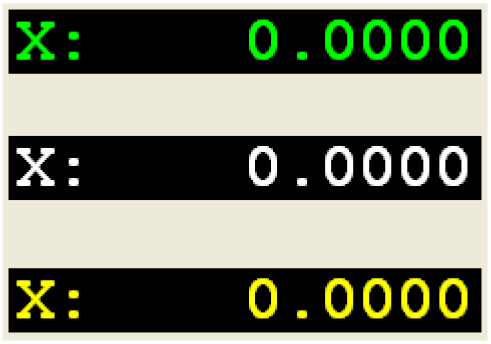
Other Codes:

F (Set Feed rate in/min or mm/min)

S (Spindle Speed)

D (Tool)

Display



The 4 axis Display along the top of the screen indicated the current position of each axis. The units of the display are in either mm or inches depending on the current mode of the interpreter (see Coordinate System Units).

The displayed position will match the g -code programmed position (i.e. last G1 commanded position) which is not necessarily the actual machine tool position if global or fixture offsets are in use.

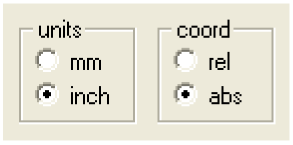
The color of the display gives an indication of current status.

Green - indicates normal status, hardware is connected, axis is enabled, and the displayed position in the current tool position.

White - indicates simulation mode is selected. The displayed position is the current position after the last line of interpreted G code.

Yellow - indicates hardware disconnected or axis disabled. The displayed position is invalid

Coordinate System Units / Mode



units: mm [✓ inch]

coord: rel [✓ abs]

Displays the current mode of the G code interpreter.

G20 selects English Inch units

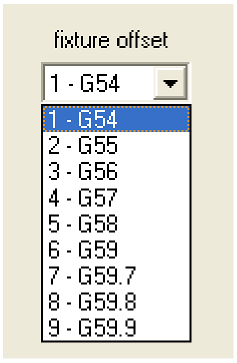
G21 selects Metric mm units

Comments: **G90** selects Absolute Coordinates

(Simple Comment) **G91** selects Relative Coordinates

(MSG,OK toContinue?)
(CMD,EnableAxis0)
(BUF,SetBitBuf29)

Fixture Offset

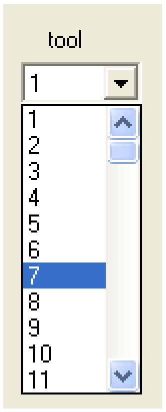


Displays and allows changing of the current Fixture Offset. *KMotionCNC* supports 9 Fixture offsets. Each Fixture may be programmed to introduce an arbitrary x,y,z,a offset. Use the **G10L2Pn** command to set the offset associated with the fixture #n. An example might be:

G10 L2 P3 X10.0 Y20.0 Z30.0 which sets Fixture Offset #3 to (10,20,30)

Executing the command **G56** (or by selecting 3 - G56 in the drop down list) will cause Fixture offset #3 to be in use in subsequent commands until a different Fixture is selected. (See also - G Code Offsets).

Tool



Displays and allows changing of the currently selected Tool. KMotionCNC supports up to 99 tool definitions. The tool definitions are specified in a text file that is selected on the ToolSetup Screen.

A tool definition consists of the tool number (FMS), the pocket where it is located if there is a tool changer available (POC), the length of the tool (LEN), the diameter of the tool (DIAM), and an optional comment.

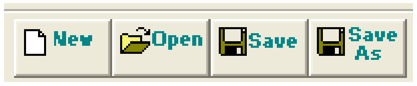
Executing the command **D3** (or by selecting 3 in the drop down list) will cause Tool #3 to be in use in subsequent commands until a different Tool is selected.

See below for an example Tool Table

```
POC FMS  LEN   DIAM  COMMENT
```

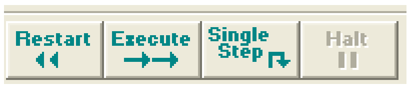
1	1	0.0	0.0	first tool
2	2	0.0	0.0	
3	3	1.0	0.5	
4	4	2.0	1.0	
32	32	0.0	0.0	last tool

File New / Open/ Save



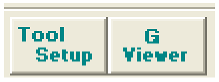
This group of pushbuttons allow a G Code file to be loaded or saved to or from the edit window. The edit window allows the user to quickly switch between 7 loaded G Code files. Once a file is loaded into one of the edit windows, the name of that file will persist between sessions.

Execute Controls



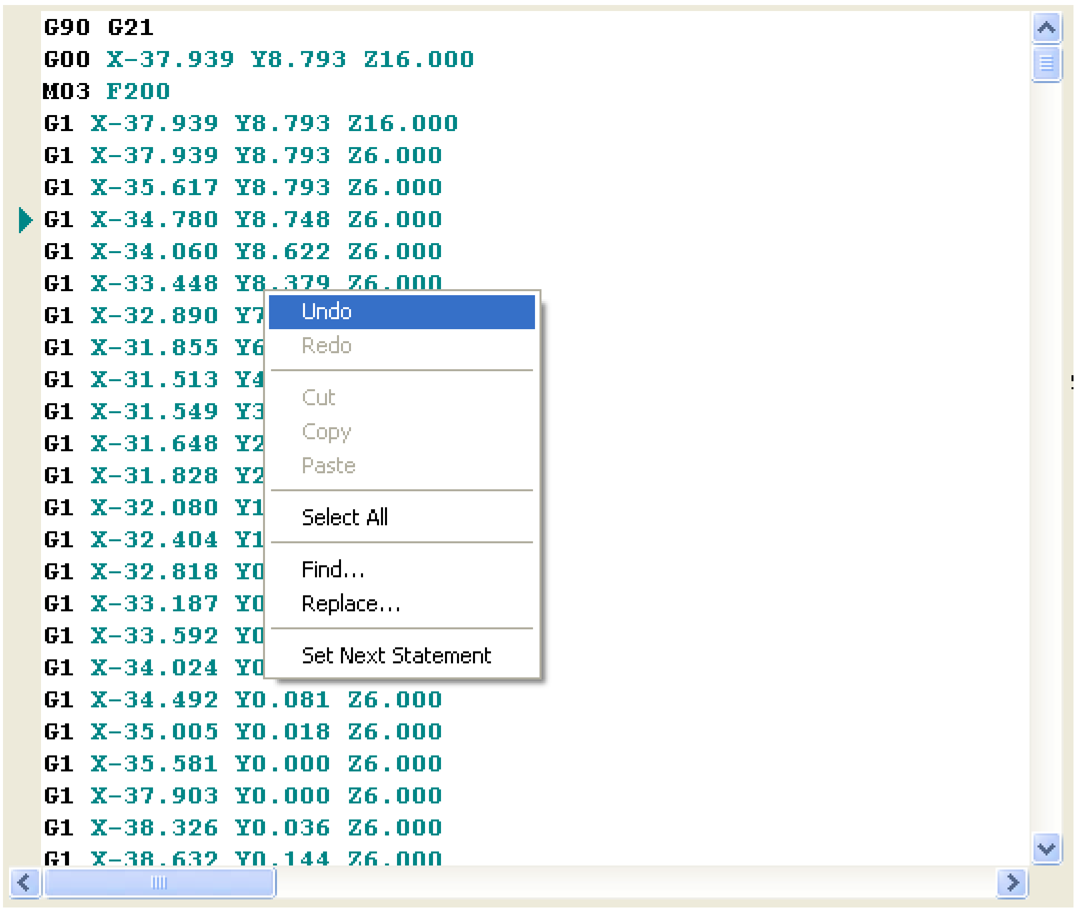
This group of pushbuttons allow the control of G Code execution. ***Restart*** will reset the instruction pointer ▶ to the first line of the file. ***Execute*** will begin continuous execution from where the current instruction pointer ▶ is located on the left of the edit window. ***Single Step*** will execute one single line of G code where the current instruction pointer is currently pointing. Note that the instruction pointer may be moved to any line by right clicking on the line within the edit window and selecting Set Next Statement.

Show Tool Setup / G Code Viewer Screens



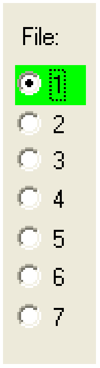
This group of buttons bring additional screens into view. The Tool Setup Screen is used to configure the system's parameters. Machine Axis distance/velocity/accelerations. M Code and User Button Actions Actions, Tool and Setup definition files, and Jog Button and Joystick rates. The G Code Viewer Screen allows real-time, 3D viewing of the machine tool paths either during actual machine operation or during simulation.

G Code Edit Window



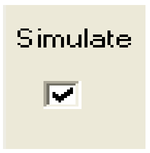
```
G90 G21
G00 X-37.939 Y8.793 Z16.000
M03 F200
G1 X-37.939 Y8.793 Z16.000
G1 X-37.939 Y8.793 Z6.000
G1 X-35.617 Y8.793 Z6.000
G1 X-34.780 Y8.748 Z6.000
G1 X-34.060 Y8.622 Z6.000
G1 X-33.448 Y8.379 Z6.000
G1 X-32.890 Y7
G1 X-31.855 Y6
G1 X-31.513 Y4
G1 X-31.549 Y3
G1 X-31.648 Y2
G1 X-31.828 Y2
G1 X-32.080 Y1
G1 X-32.404 Y1
G1 X-32.818 Y0
G1 X-33.187 Y0
G1 X-33.592 Y0
G1 X-34.024 Y0
G1 X-34.492 Y0.081 Z6.000
G1 X-35.005 Y0.018 Z6.000
G1 X-35.581 Y0.000 Z6.000
G1 X-37.903 Y0.000 Z6.000
G1 X-38.326 Y0.036 Z6.000
G1 X-38.632 Y0.144 Z6.000
```

File Selector



The file selector shows which of the 7 loaded G Code files is currently active for editing. The Main Window Title also displays the loaded filename for the selected file. The file number is highlighted in green when that file is currently executing G Code. Only one G Code file is allowed to execute at a particular time.

Simulate



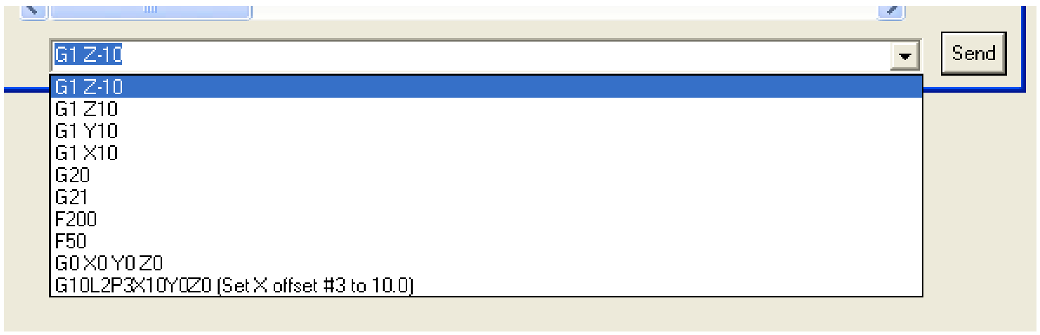
Enables Simulation Mode which allows viewing and verification of a G Code Program with or without any actual hardware connected. When Simulation mode is enabled no actual machine motion will be made. Executing or Single Stepping through a G Code program will change the Displayed Position and Plot the machine tool path on the G Code Viewer Screen. In Simulation Mode the Numeric Display Color changes to white to indicate the display is not showing the actual machine tool position. While in Simulation mode the Jog Buttons and Gamepad buttons will also change the displayed position and tool position on the G Code Viewer Screen without causing any actual machine tool motion.

Emergency Stop

Emergency Stop may be used to immediately stop all motion. Any commands in motion will be aborted and all axes will be disabled. After depressing Emergency Stop the system must be re-initialized and the G Code Interpreter state will be lost. Use Halt to stop in a controlled manner after the next line of GCode has been completed.

The ESC key may also be use to initiate an Emergency Stop whenever the KMotionCNC Screen has the focus.

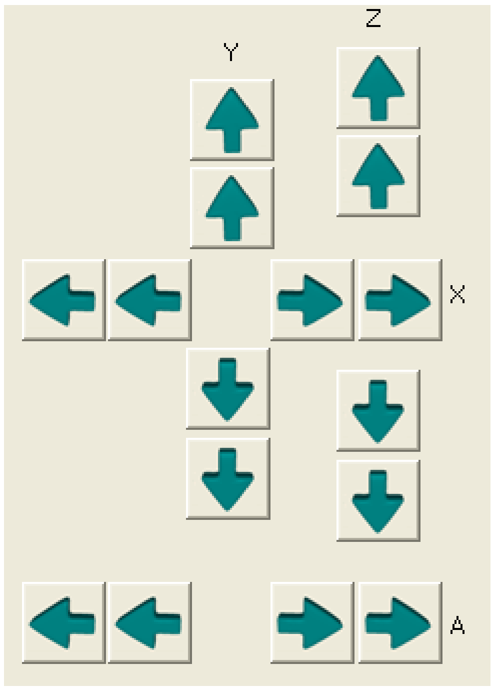
Manual Entry



```
G1 Z-10
G1 Z-10
G1 Z10
G1 Y10
G1 X10
G20
G21
F200
F50
G0 X0 Y0 Z0
G10L2P3X10Y0Z0 (Set X offset #3 to 10.0)
```

The Manual Entry cell allows the user to quickly enter a single line of G Code and Send it to the interpreter for execution. The last 10 entered commands are saved in a drop down list for quick re-entry.

Jog Buttons



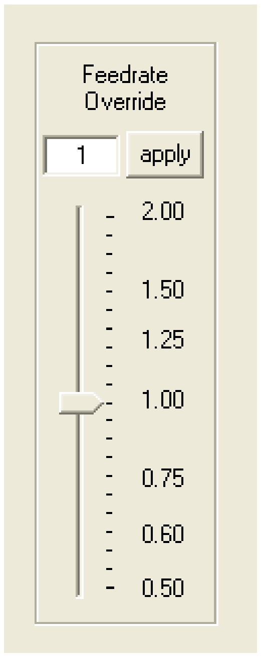
The Jog buttons may be used to move any of the axes. Pushing and holding any of the buttons will cause continuous motion. There are 2 buttons in each direction for each axis. The second button moves at twice the rate as the first. The speeds for each axis may be specified in the Tool Setup Screen.

A USB Gamepad such as the one shown below may also be used to Jog the System. Simply connect the Gamepad and it should become immediately active. The left Joystick controls x and y and the right joystick controls z and a. The same speed parameters in the Tool Setup Screen control both the Jog pushbuttons on the screen and the Gamepad.

The Jog buttons and Gamepad are also active in simulation mode

Feed Rate Override

Feed Rate Override provides a means to adjust the feedrate while the machine is in operation without having to modify the G Code. The Feed Rate is specified within the G Code using the F command and the Feed Rate Override is a multiplicative factor that is applied to that value. For example F100 would specify a Feed rate of 100 inches/minute (or 100 mm/minute if the interpreter is in metric mm mode), with a feed rate override of 1.5 the actual attempted feed rate would be 150 inches/minute (or 150 mm/minute in mm mode).

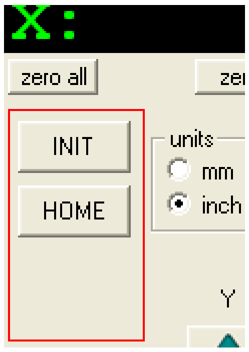


Note that this speed will only be used if all the axes involved will remain below the maximum allowed speeds set on the Tool Setup Screen. Additionally, short vectors with sharp corners (greater than the specified break angle) that require stopping may be limited due to acceleration limits. KMotionCNC uses complex algorithms to ensure that velocities and accelerations remain below user specified limits. So if the Feed Rate Override or (Specified Feed rate itself) doesn't seem to be having the expected result, check if the maximum velocities and accelerations are limiting.

The Feed Rate Override may be changed either by using the slider or by typing a value and pressing the apply button.

Note that because motions are planned ahead and downloaded to the Motion Controller ahead of time, that the feed rate override will take a short amount of time to have an effect. The amount of time that the trajectory planner looks ahead is specified on the Tool Setup Screen and is normally set at from 1 to 3 seconds. The main limitation to making this value very short is the worst case response time of Microsoft Windows™ and the PC hardware being used.

Custom Buttons



KMotionCNC allows up to 5 Custom Buttons to be displayed and defined for special operations. Which of these buttons are visible, what they display as a title, and what action they perform are all definable on the Tool Setup Screen.

The area shown within the red rectangle is where the Custom Buttons will appear if defined. The actions that the buttons perform are defined in the same manner as the actions that M Codes perform. These may be simple actions such as setting an Output to turn something on or may be a complex operation that involves invoking a program. Normally one or more buttons will be used to initialize and configure the motion controller and/or home the machine.

Other GCode Commands

KMotion's G Code interpreter was derived from the Open Source EMC G Code Interpreter. Click here for the EMC User Manual (Only the G Code portions of the manual, Chapters 10-14 pertain to ***KMotion*** G Code)

Specially coded comments embedded within a GCode program may be used to issue ***KMotion*** Console Script commands directly to ***KMotion***.

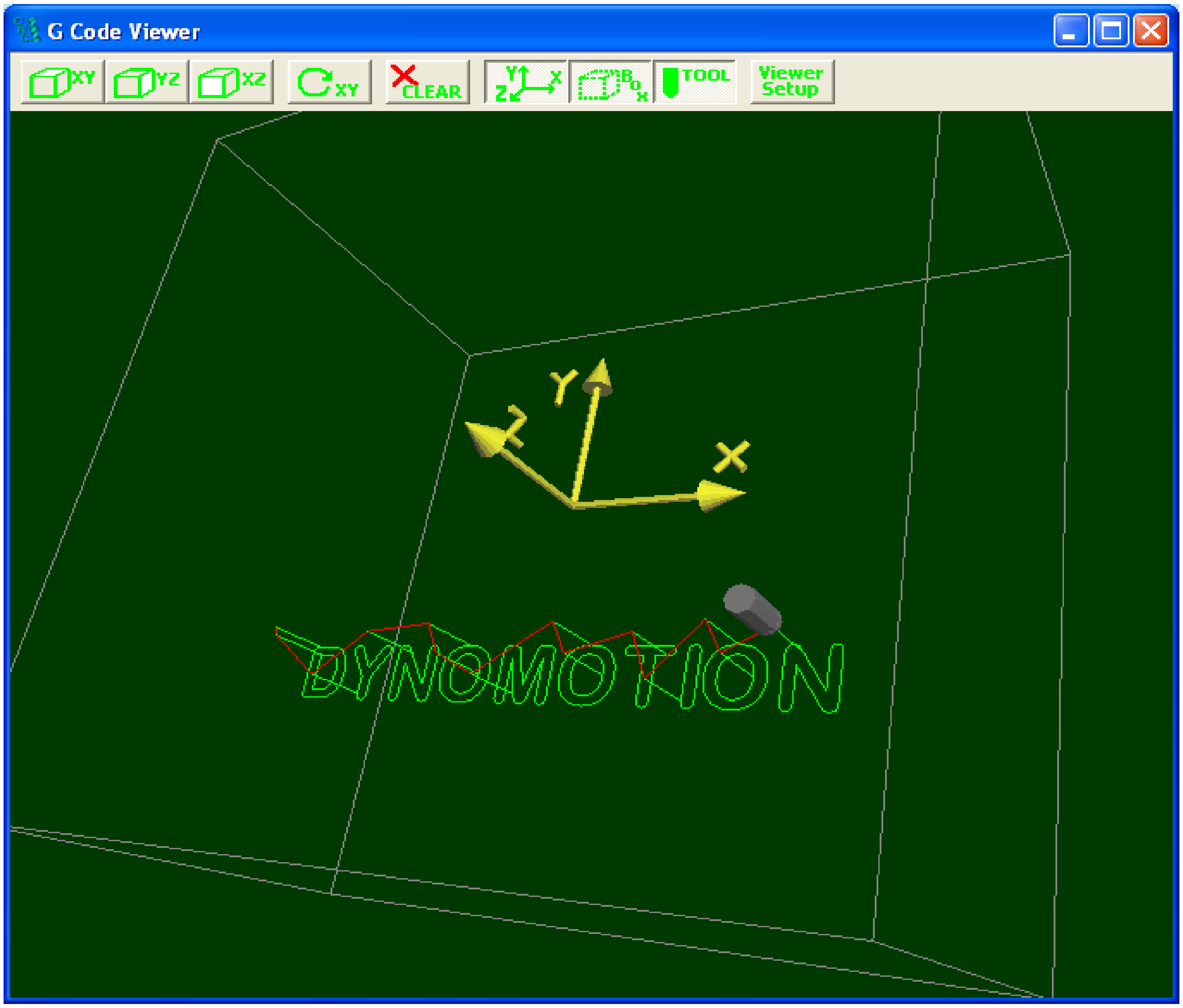
A comment in the form: (CMD,xxxxxx) will issue the command xxxxxx immediately to ***KMotion*** as soon as it is encountered by the Interpreter. Any ***KMotion*** command that does not generate a response may be used in this manner.

A comment in the form: (BUF,xxxxxx) will place the command xxxxxx into ***KMotion's*** coordinated motion buffer. Coordinated motion sequences are download to a motion buffer within ***KMotion*** before they are executed. This guarantees smooth uninterrupted motion. The BUF command form allows a command to be inserted within the buffer so they are executed at an exact time within the motion sequence. Only the following ***KMotion*** Script commands may be used in this manner.

SetBitBuf, ClearBitBuf, SetStateBitBuf.

Additionally, a comment in the form: (MSG,xxxxxx) will pause GCode Execution and display a pop-up message window displaying the message xxxxxxx.

G Code Viewer Screen



The G Code Viewer Screen displays the 3D motions of the machine tool as a GCode program is executing. To Display an entire G Code Program quickly without any physical motion of the machine tool, select Simulate on the main KMotionCNC Screen and execute the program. Linear and Circular Feed Motions (G1, G2, G3) are displayed as green paths. Rapid Motions (G0) are displayed as red lines. Note that rapid motions may not actually be performed as straight lines since during rapid motions each axis moves independently as quickly as possible.

Click and drag the **left** mouse Button to translate **up/down/left/right.**

Click and drag the **right** mouse Button to translate **closer or farther.**

Click and drag **both** mouse Buttons to **rotate**.

Top/Side/Front Views

Use these buttons to view directly from the Top, Side, or Front respectively. The camera is positioned at a distance away to view the entire box extents or current path extents whichever is greatest.

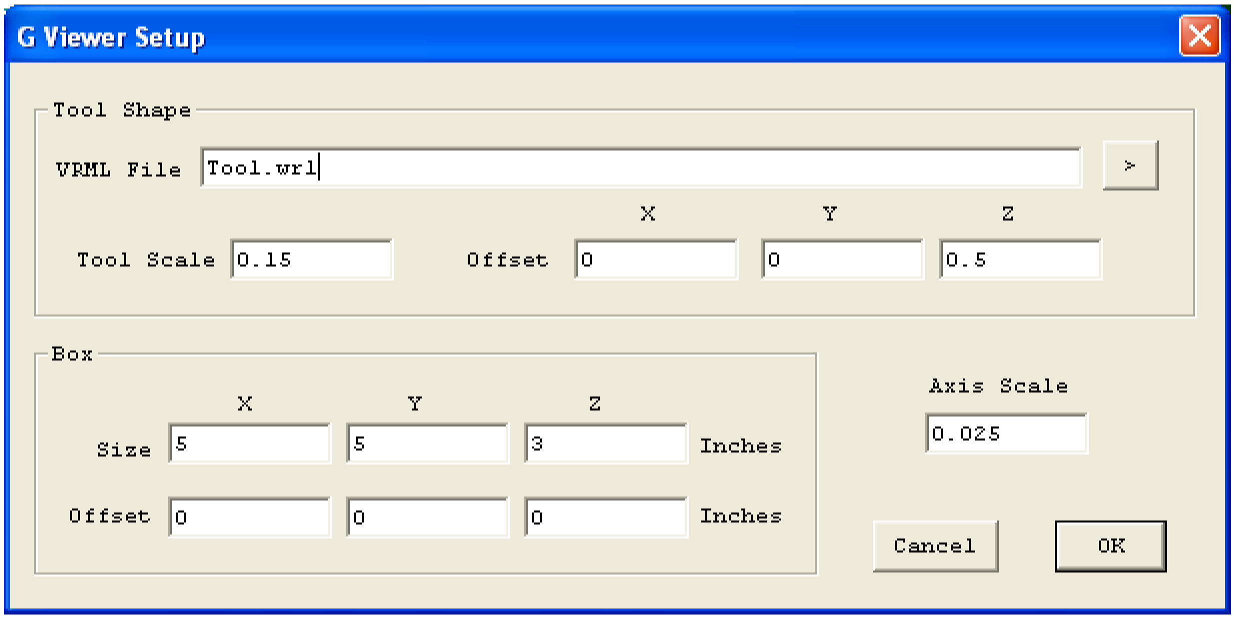
This is a latching toggle button. When latched down rotation is in the x y plane about the z axis. When unlatched, rotation is up/down/left/right.

Clears all path vectors. As a G Code Program executes motion paths are are saved and displayed. Pushing this button clears all vectors from the internal buffer. All path vectors are also automatically cleared when the first line of a G Code program executes.

These are three latching toggle buttons which determine whether the Axis, Box, or Tool Shapes respectively are to be displayed. When latched down the item is displayed. When latched up the item is hidden. The size and shape of these items may be changed on the G Viewer Setup Screen.

This button brings up the G Viewer Setup Screen which allows some customization of the G Code Viewer Screen.

G Viewer Setup Screen



VRML File: Tool.wrl

	X	Y	Z
Tool Scale: 0.15 — Offset	0	0	0.5

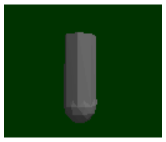
	X	Y	Z	
Size	5	5	3	Inches
Offset	0	0	0	Inches

Axis Scale: 0.025

The G Viewer Setup Screen sets the imaging parameters for the G Code Viewer Screen.

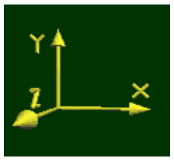
Besides the G code tools paths, the G Code Viewer Screen displays several objects, namely a Tool, a Box, and an Axis Symbol. The Tool Object and Axis Symbol are 3D VRML files that may be changed if desired. KMotionCNC comes with default files shown below.

Tool Shape



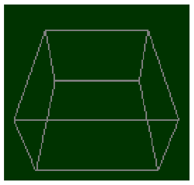
The default Tool Shape is located at: **<Install Dir>\KMotion\Data\Tool.wrl**, but maybe easily changed by entering or browsing to a new file**.** An excellent free program to create VRML files is available at http://www.artofillusion.org. The default Tool file is a 1 inch diameter sphere at the end of a 1 inch diameter cylinder which is 3 inches long. In the VRML file the origin (0,0,0) is at the center of the sphere. The Tool Scale and offsets allow the Tool Shape to be shifted and scaled as desired. As shown above an offset of (0,0,0.5) shifts the tool such that the origin is at the very tip of the tool. The scale of 0.15 then reduces the tool "size" from 1 inch to 0.15 inch.

Axis Scale



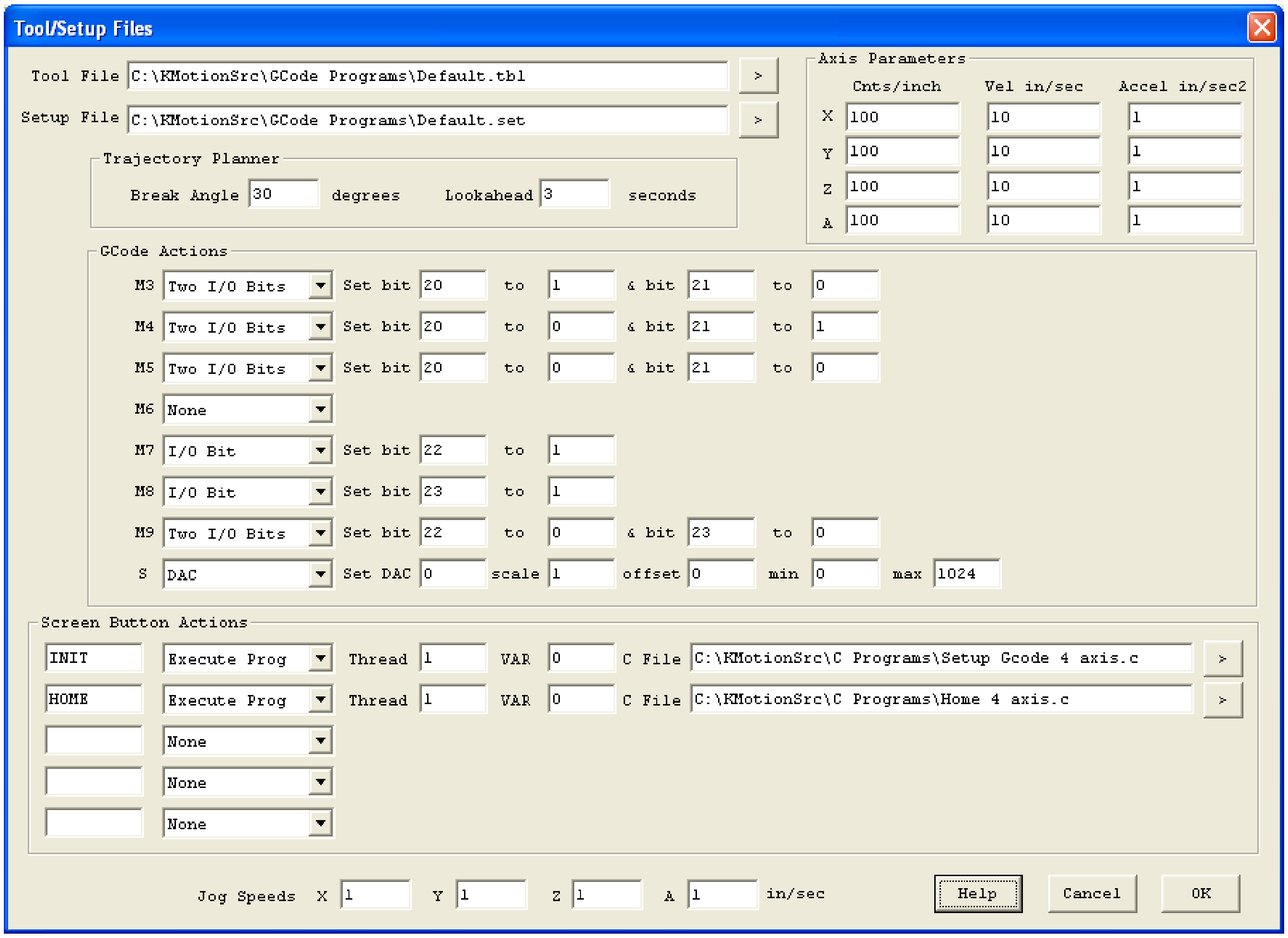
The Axis file name and location is hard coded as: **<Install Dir>\KMotion\Data\Axis.wrl.** The Axis shape is always drawn at the origin. It's size may be changed by changing the Axis Scale value. If a different Axis Shape is desired the Axis.wrl file must be overwritten with a new file.

Box



The Box serves two purposes. If enabled for display, it aids the 3D visualization and perspective of the tool paths. It is also expected to represent the working extents of the machine. When "zooming out" to one of the preset Top, Side, or Front Views, it is used to frame the view extents. Therefore the Box Size parameters should be set to the working extents of the machine. With an offset of 0,0,0 the origin will be located at the center of the Box. This is the preferred arrangement, however if required the Box may be displayed offset by entering Box Offset values other than zero

Tool Setup Screen



Tool File C:\KMotionSrc\GCode Programs\Default.tbl

Setup File C:\KMotionSrc\GCode Programs\Default.set

Trajectory Planner

Break Angle 30 degrees   Lookahead 3 seconds

Axis Parameters

	Cnts/inch	Vel in/sec	Accel in/sec2
X	100	10	1
Y	100	10	1
Z	100	10	1
A	100	10	1

GCode Actions

M3	Two I/O Bits	Set bit	20	to	1	& bit	21	to	0		
M4	Two I/O Bits	Set bit	20	to	0	& bit	21	to	1		
M5	Two I/O Bits	Set bit	20	to	0	& bit	21	to	0		
M6	None										
M7	I/O Bit	Set bit	22	to	1						
M8	I/O Bit	Set bit	23	to	1						
M9	Two I/O Bits	Set bit	22	to	0	& bit	23	to	0		
S	DAC	Set DAC	0	scale	1	offset	0	min	0	max	1024

Screen Button Actions

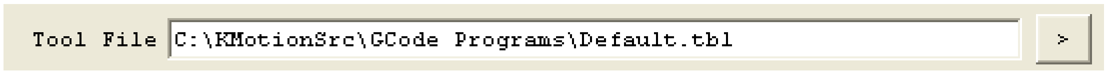
INIT	Execute Prog	Thread	1	VAR	0	C File	C:\KMotionSrc\C Programs\Setup Gcode 4 axis.c
HOME	Execute Prog	Thread	1	VAR	0	C File	C:\KMotionSrc\C Programs\Home 4 axis.c
	None						
	None						
	None						

Jog Speeds X 1 Y 1 Z 1 A 1 in/sec

(Click on Image to Jump to related help)

The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to different I/O requirements with regard to Spindle control and such. Additionally a machine may have different initialization and homing requirements. *KMotionCNC* has a flexible mechanism for defining what type of action is to be performed for various M Codes and Custom Buttons.

Tool Table File

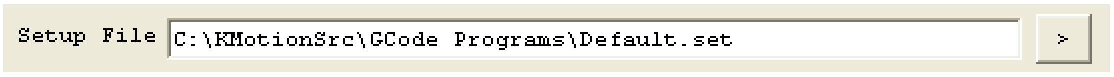


The Tool Table File specifies the disk text file which contains the table of tool definitions. In some cases the G Code Interpreter needs to know the length and diameter of the selected tool for tool path compensation. This file is used to define up to 99 tools. See also Selecting Tools.

See below for an example Tool Table

POC	FMS	LEN	DIAM	COMMENT
1	1	0.0	0.0	first tool
2	2	0.0	0.0	
3	3	1.0	0.5	
4	4	2.0	1.0	
32	32	0.0	0.0	last tool

Setup File

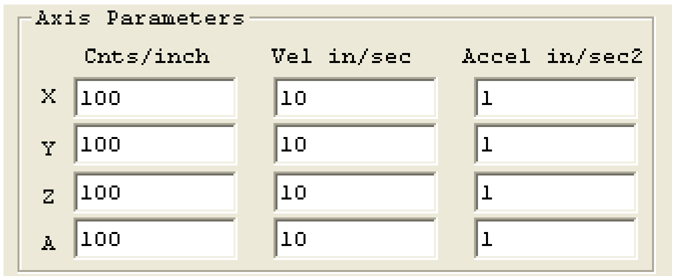


Setup File: C:\KMotionSrc\GCode Programs\Default.set

The Setup File specifies the disk text file which contains the setup table for the G Code Interpreter. In some machine tools the Interpreter may require a special initialization state. Below is the default Setup file. Modifications to the setup file should not normally be required.

Attribute	Value	Other Possible Values
axis_offset_x	0.0	any real number
axis_offset_y	0.0	any real number
axis_offset_z	0.0	any real number
block_delete	ON	OFF
current_x	0.0	any real number
current_y	0.0	any real number
current_z	0.0	any real number
cutter_radius_comp	OFF	LEFT, RIGHT
cycle_r	0.0	any real number
cycle_z	0.0	any real number not less than cycle_r
distance_mode	ABSOLUTE	INCREMENTAL
feed_mode	PER_MINUTE	INVERSE_TIME
feed_rate	5.0	any positive real number
flood	OFF	ON
length_units	MILLIMETERS	INCHES
mist	OFF	ON
motion_mode	80	0,1,2,3,81,82,83,84,85,86,97,88,89
plane	XY	YZ, ZX
slot_for_length_offset	1	any unsigned integer less than 69
slot_for_radius_comp	1	any unsigned integer less than 69
slot_in_use	1	any unsigned integer less than 69
slot_selected	1	any unsigned integer less than 69
speed_feed_mode	INDEPENDENT	SYNCHED
spindle_speed	1000.0	any non-negative real number
spindle_turning	STOPPED	CLOCKWISE, COUNTERCLOCKWISE
tool_length_offset	0.0	any non-negative real number
traverse_rate	199.0	any positive real number

Axis Motion Parameters



Axis Parameters

	Cnts/inch	Vel in/sec	Accel in/sec2
X	100	10	1
Y	100	10	1
Z	100	10	1
A	100	10	1

The Axis Motion Parameters define the scale, maximum feed velocities, and maximum feed accelerations for each of the four axis.

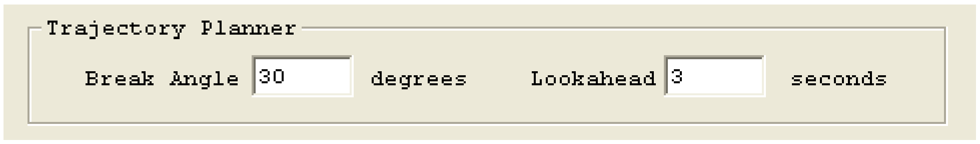
The first parameter is the axis's scale in counts/inch. For the example value of 100 shown, *KMotionCNC* will command the motion controller to move 100 counts for each inch of motion in the G Code Program. This value is always in counts/inch regardless of the units used in the interpreter. *KMotionCNC* will automatically perform any conversions.

The second parameter is the maximum allowed feed rate for the axis in inches/sec. Note that the G Code Interpreter Feed Rate is defined in inches per minute or (mm per minute) so be aware of the different time units. These are *maximum* feed rates for each axis. If a vector tool motion at a particular feed rate has any component velocity that exceeds the corresponding axis's maximum velocity, the feed rate for the vector will be reduced so that all axes remain at or below their maximum allowed velocity.

The third parameter is the maximum allowed acceleration for the axis in inches/sec^2. The G Code Language has no provisions for specifying acceleration rates. Therefore the acceleration (and deceleration) along a vector used will always be the largest acceleration such that each axis's acceleration is at or below the specified limit.

The velocity and acceleration limits apply only to linear and circular feed motions (G1, G2, G3). Rapid motions (G0) use the settings in the motion controller (velocity, acceleration, and Jerk) to move each axis independently from one point to another (which is likely not to be a straight line). To change the speed of Rapid motions change the configuration settings in the motion controller.

Trajectory Planner

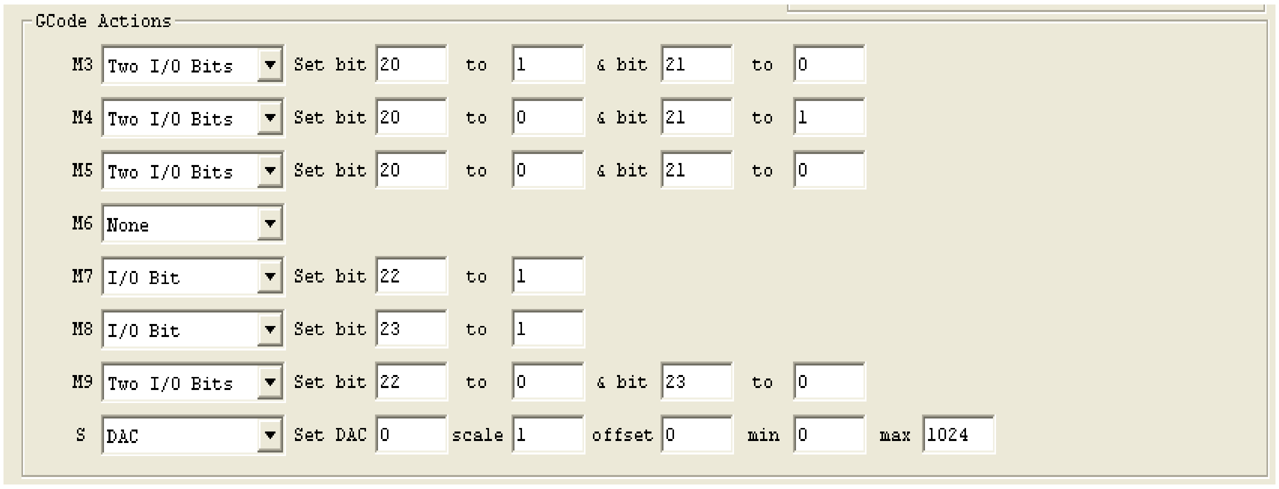

Trajectory Planner

Break Angle 30 degrees   Lookahead 3 seconds

KMotionCNC contains a powerful Trajectory Planner. The Trajectory Planner utilizes a "break angle" concept to decide when a stop must be made. Vectors that are in the same direction within the "break angle" are traversed without stopping. When a "sharp" angle is detected a complete stop will be made before beginning on the next vector. The Break Angle Parameter allows the user to specify the angle in degrees that will be considered a "sharp" angle. *KMotionCNC* considers the change in direction in 3 dimensions (x,y,z ignoring a). The Trajectory Planner is capable of optimizing the acceleration and deceleration through many short (or long) vectors all of which may have different acceleration and velocity limitations.

The Trajectory Planner also has a "lookahead" parameter. With KMotionCNC the G Code Program itself, the G Code Interpreter, and the Trajectory Planner all reside within the PC running Microsoft Windows™. Since the Microsoft Windows™ is *not* a real-time OS, a certain amount of motion must be buffered in the motion controller to handle the cases where the PC program doesn't have a chance to execute for a while. These time periods are typically very short (a few milliseconds), but in some extreme cases may occasionally be as long as one or several seconds. The worst case is often a factor of the hardware (disk network adapters, etc) and associated drivers that are loaded into the system. The lookahead parameter is used to determine how much motion, in terms of time, should be downloaded to the motion controller *before* actual motion is initiated. Furthermore, after motion has begun, the lookahead parameter is used to pause the trajectory planner to avoid having the Trajectory Planner get too far ahead of the actual motion. The disadvantage of having the Trajectory Planner get too far ahead is that if the User decides to override the feed rate, since the motion has already been planned and downloaded, the rate will not be changed quickly. A value of 3 seconds is very conservative on most systems. If more responsive feed rate override is desirable, an experiment with a smaller value might be made.

G Code Actions



GCode Actions

M3	Two I/O Bits	Set bit	20	to	1	& bit	21	to	0	
M4	Two I/O Bits	Set bit	20	to	0	& bit	21	to	1	
M5	Two I/O Bits	Set bit	20	to	0	& bit	21	to	0	
M6	None									
M7	I/O Bit	Set bit	22	to	1					
M8	I/O Bit	Set bit	23	to	1					
M9	Two I/O Bits	Set bit	22	to	0	& bit	23	to	0	
S	DAC	Set DAC	0	scale	1	offset	0	min	0	max 1024

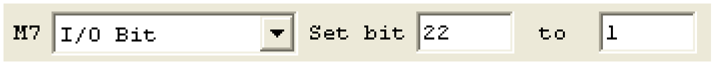
The G Code Actions section of the Tool Setup Screen defines what action is to be performed when a particular G Code Command (Mostly M Codes) is encountered. The Action that can be performed can be one of several things:

- None
- Set or Reset one I/O Bit
- Set or Reset two I/O Bits
- Set a DAC to a variable's value
- Execute a C Program in the KMotion Control Board

To specify a particular action first select the Action Type. Each Action Type requires a different number and type of parameters. Next fill in the appropriate parameters. The one and two bit I/O commands are inserted directly into the coordinated motion control buffer. In this way they are exactly synchronized with any motion before or after the I/O commands. This is useful in systems where a fast shutter or other operation is required at precise times relative to the motion.

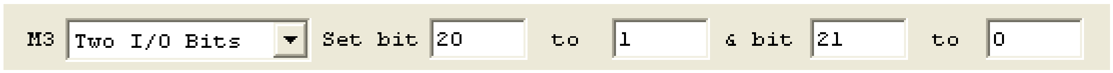
The four Action Types are described below:

For **one I/O** bit specify the I/O bit number and the state 0 or 1 to set it to.



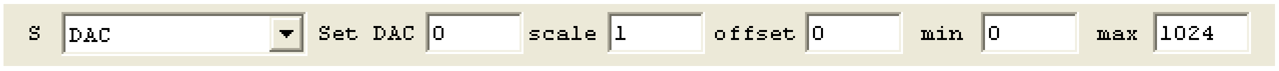
M7 I/O Bit Set bit 22 to 1

For **two I/O** bits specify the I/O bit numbers and the state 0 or 1 to set each to. Often systems with two direction spindle control will require setting two outputs that control on/off and cw/ccw. This command is designed to handle those situations.

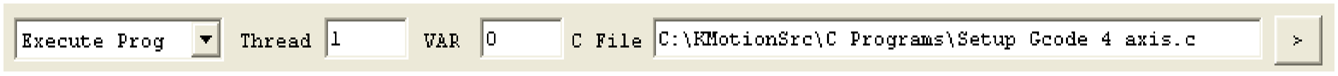


M3 Two I/O Bits Set bit 20 to 1 & bit 21 to 0

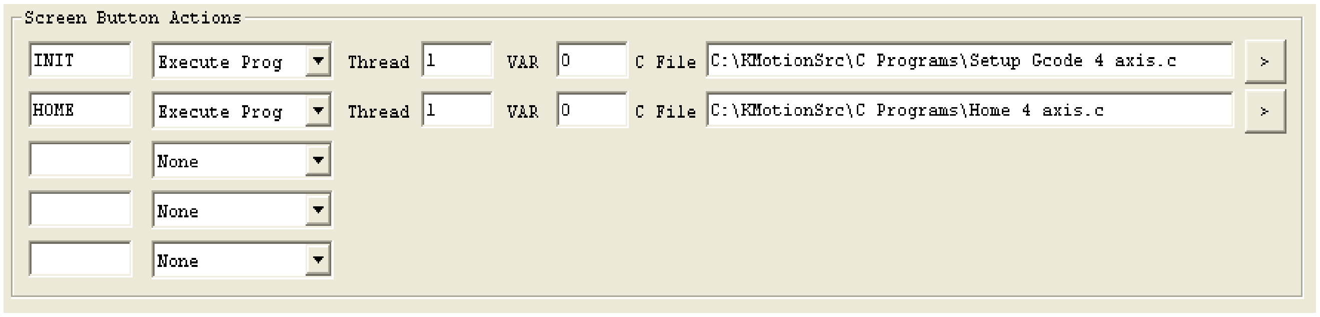
For **DAC** specify the DAC (Digital to analog converter) channel number, how to scale and offset the associated variable, and the minimum and maximum allowed DAC settings. This command is primarily designed for use with the S (Spindle speed) G Code Command


S DAC Set DAC 0 scale 1 offset 0 min 0 max 1024

For **Execute Prog** specify the program Thread (1 through 7) where the program is to be downloaded and executed, a Persist Variable (1-99) that will be set before the program executes, and the name of the C Program to be Compiled, Downloaded, And Executed. If the Filename is left blank, then it will be assumed that a program has been previously downloaded and will just be re-executed. This method is very powerful in that anything that may be programmed in C may be invoked. See the KMotion documentation for information on writing C Programs for the KMotion Motion Control Board. There are a number of example C programs in the <Install Dir>\C Programs folder. The Example "Setup Gcode 4 axis.c" s an example which completely configures all necessary parameters on the KMotion Board to drive 4 stepping motors.


Execute Prog Thread 1 VAR 0 C File C:\KMotionSrc\C Programs\Setup Gcode 4 axis.c

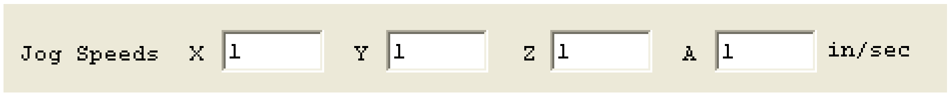
Custom Buttons


Screen Button Actions

INIT	Execute Prog	Thread 1	VAR 0	C File C:\KMotionSrc\C Programs\Setup Gcode 4 axis.c	>
HOME	Execute Prog	Thread 1	VAR 0	C File C:\KMotionSrc\C Programs\Home 4 axis.c	>
	None				
	None				
	None				

Custom Button Actions function in exactly the same manner as the G Code Actions described above with the only difference being that they are invoked by the User pushing a button rather than by a command encountered within a G Code Program. There is an additional Parameter on the very left of the Action Type which is the Title to be placed on the Custom Button. The example above shows two buttons defined with titles "INIT" and "HOME". Up to 5 buttons may be defined. Common uses for the Buttons are to invoke programs that initialize and/or home the machine. Any Button with an empty title field will cause the button to be hidden on the main *KMotionCNC* screen. See here for how these defined buttons will appear in the main *KMotionCNC* Screen.

Jog Speeds



Defines the Jog Speeds for both the Jog Buttons and any attached Gamepad controller. These speed are the maximum Jog speed which is the double arrow jog button or the GamePad joystick fully depressed. See Also Jog Buttons.

G Code Offsets

KMotionCNC supports G Code *Fixture Offsets* (G54 through G59.9) as well as a Global Offset (G92). So there are two methods that may be used to offset a G Code Program. Both may be used concurrently.

The following formula is used:

Tool Position = Programmed Position + Currently Selected Fixture Offset + Global Offset

Fixture Offsets are specified in a straightforward manner by using the G10 L2 Pn command to specify the value of the offset. The system has memory to remember 9 fixture offsets. Any of the fixture offsets may then be selected for use using the G54 - G59.9 commands. One and only one fixture offset is always in use (one exception to this is a line containing the G53 command, which temporarily causes no offsets whatsoever to be used for that one line only).

The Global Offset is specified in an indirect manner by using the G92 command to declare the current tool position to be a specified programmed position. The specified position may be zero or any location. For example, if it is known that a particular G Code program expects a particular location on the part to be (-1,-1,-1), then the tool could be physically moved to that point on the part and the command:

G92 X-1 Y-1 Z-1

could be entered. The system will then automatically calculate the appropriate Global Offset to make the current programmed position (-1,-1,-1). No physical motion will occur. The Displayed Position will change to the position specified (-1,-1,-1). Any current Fixture Offset will be taken into consideration as well

Example:

Consider the program shown below. Initially there are no offsets.

After line N1 is executed,

Tool Position (-0.1,-0.2,-0.3) = Programmed Position (-0.1,-0.2,-0.3) + Fixture Offset (0,0,0) + Global Offset (0,0,0)

After line N2 is executed, no physical motion occurs but a Global Offset of (-0.1,-0.2,-0.3) is calculated and:

Tool Position (-0.1,-0.2,-0.3) = Programmed Position(0,0,0) + Fixture Offset (0,0,0) + Global Offset (-0.1,-0.2,-0.3)

After line N3 movement to programmed position of (1,2,3) occurs and:

Tool Position (0.9,1.8,2.7) = Programmed Position(1,2,3) + Fixture Offset (0,0,0) + Global Offset (-0.1,-0.2,-0.3)

After lines N3 and N4 specify a fixture offset and select it for use, no motion occurs but the program (displayed) position changes to:

Tool Position (0.9,1.8,2.7) = Programmed Position(0.5,1.3,2.1) + Fixture Offset (0.5,0.7,0.9) + Global Offset (-0.1,-0.2,-0.3)

After line N5 is executed motion to programmed position (1,1,1) occurs and:

Tool Position (1.4,1.5,1.6) = Programmed Position(1,1,1) + Fixture Offset (0.5,0.7,0.9) + Global Offset (-0.1,-0.2,-0.3)

```
N1 G00 X-0.1 Y-0.2 Z-0.3     (move somewhere)
N2 G92 X0 Y0 Z0              (declare where we are to be zero)
N3 G00 X1 Y2 Z3

N4 G10 L2 P2 X0.5 Y0.7 Z0.9  (Set G55 - fixture offset #2)
N5 G55                       (Set to use fixture offset #2)
N6 G00 X1 Y1 Z1

N7 G53 X0 Y0 Z0
```

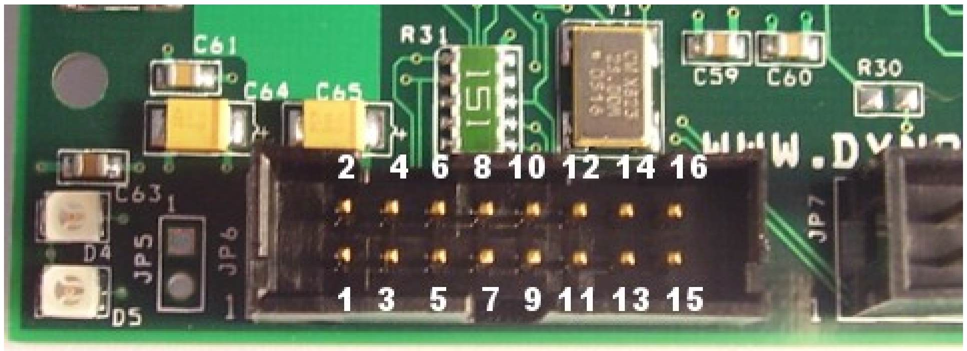
SnapAmp - Connector Pinouts

SnapAmp contains three connectors labeled JP1, JP6, and JP7.

JP6 - KMotion Communication & Logic Power

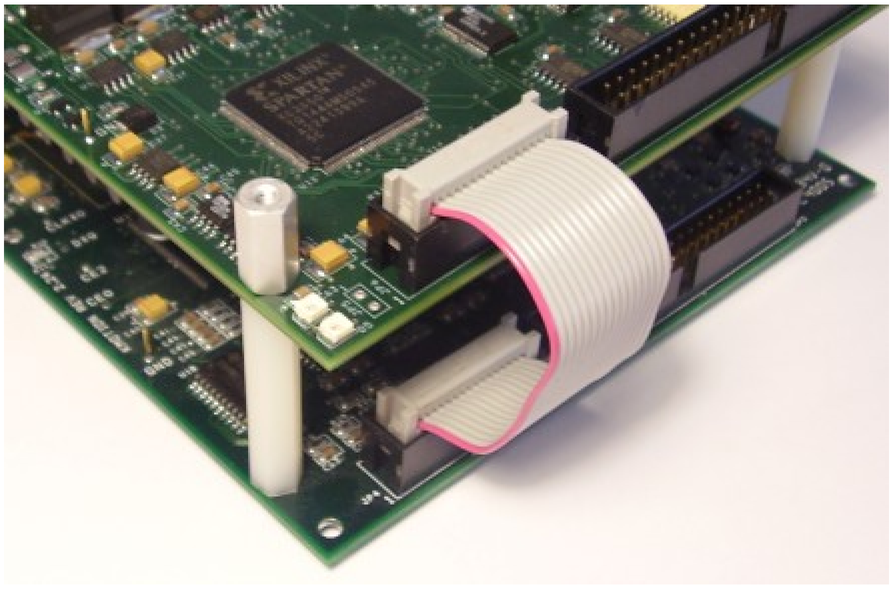
JP6 is a proprietary high-speed communication bus where command and status communication between the KMotion board and the SnapAmp Amplifier. Up to two SnapAmps may be attached to a single KMotion Board. The second of the two SnapAmps must have a configuration jumper installed into JP5.

This connection is required for proper operation of the SnapAmp and should be as short as possible.



16 pin ribbon cable connection between SnapAmp and KMotion.

Note: 4 - 40 x 1 3/8 inch standoffs are used between SnapAmp (top) and KMotion (bottom).



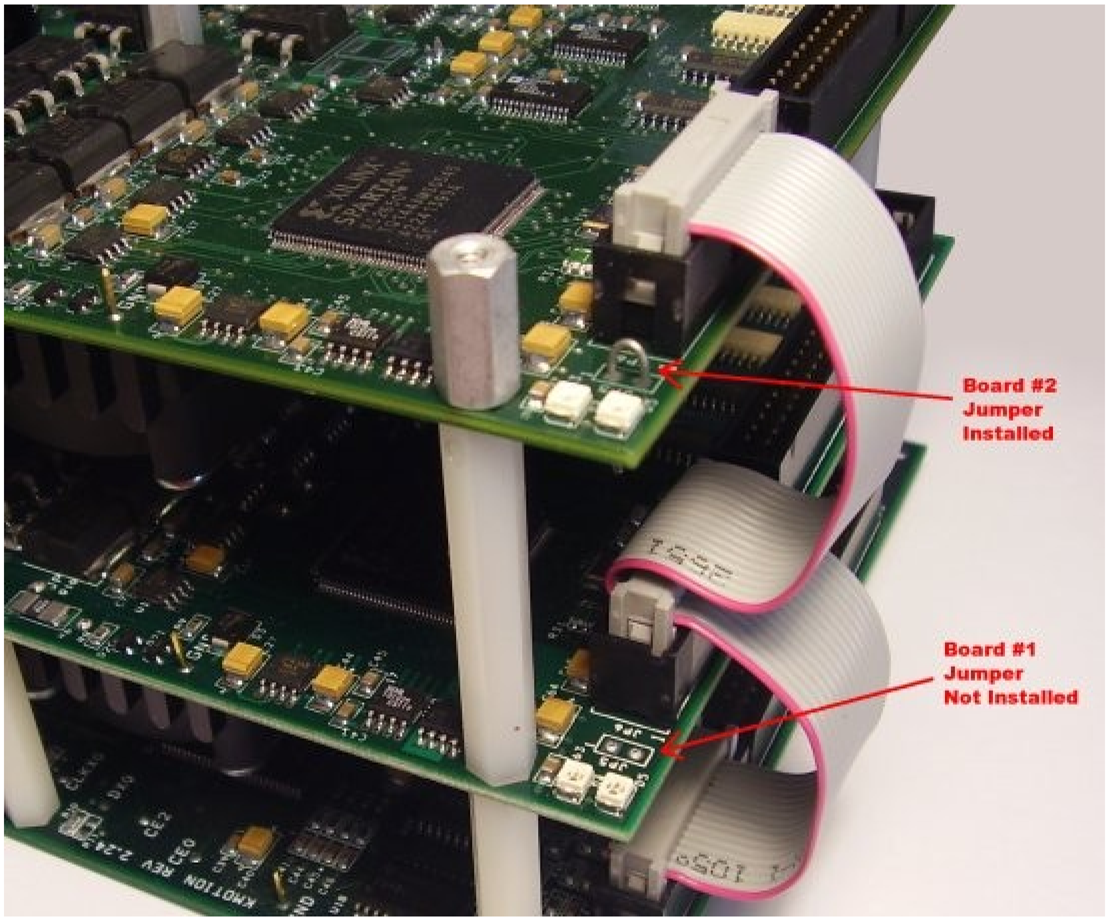
16 pin ribbon cable connection between Dual SnapAmps and KMotion

Notes:

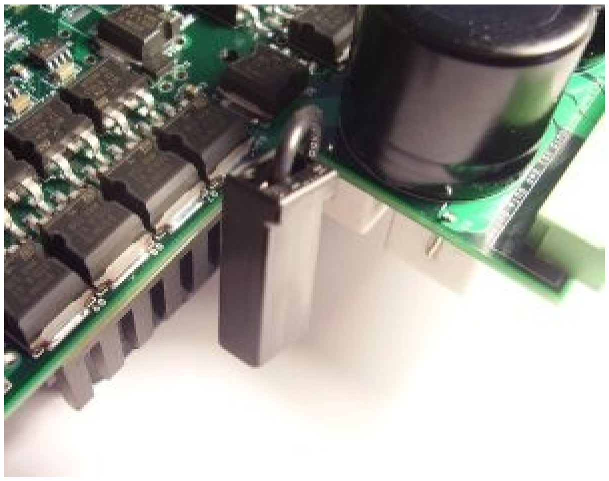
4 - 40 x 1 3/8 inch standoffs are used between SnapAmp and KMotion

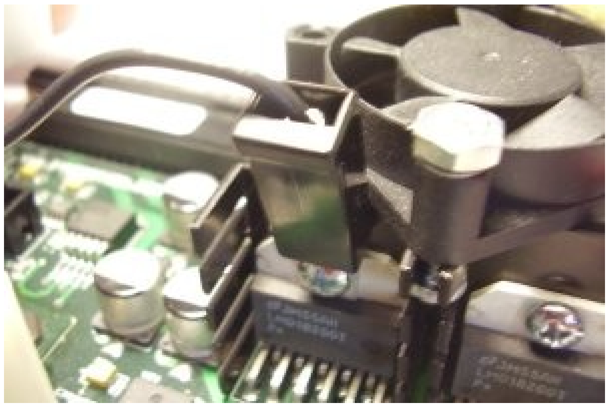
4 - 40 x 1 5/8 inch standoffs are used between SnapAmps

JP5 Jumper installed configures a SnapAmp as the 2nd SnapAmp.

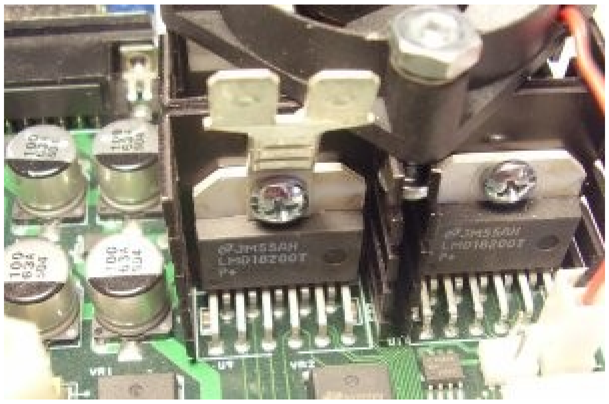


When attaching SnapAmp to the KMotion first attach main ground plug as shown.



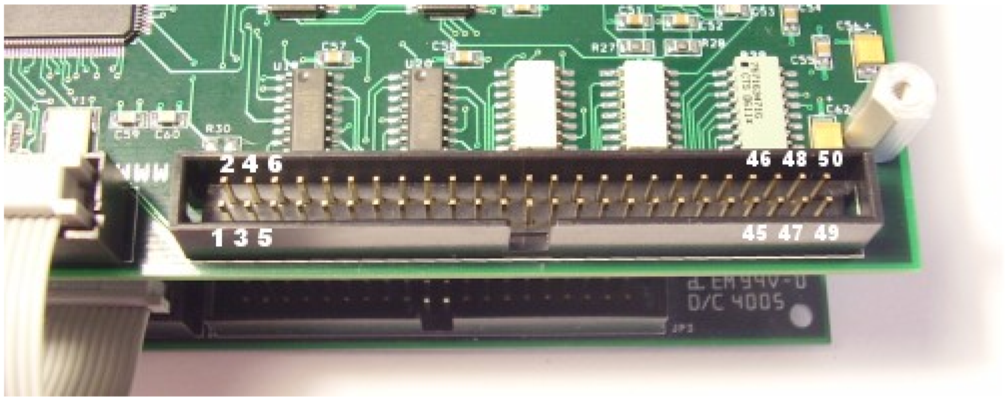


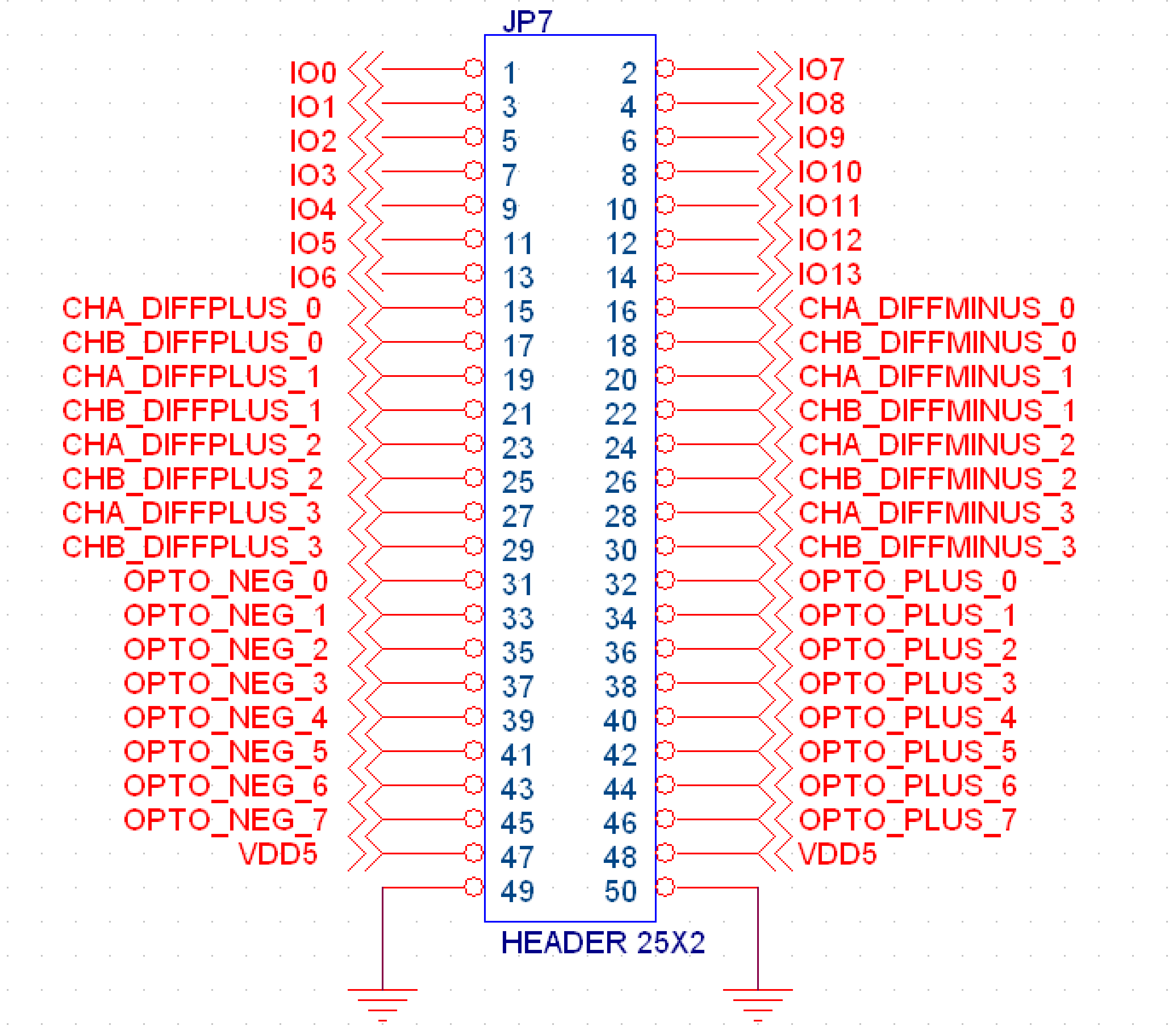
For Dual Snap Amp systems attach the 1:2 spade Y adapter before connecting the first SnapAmp



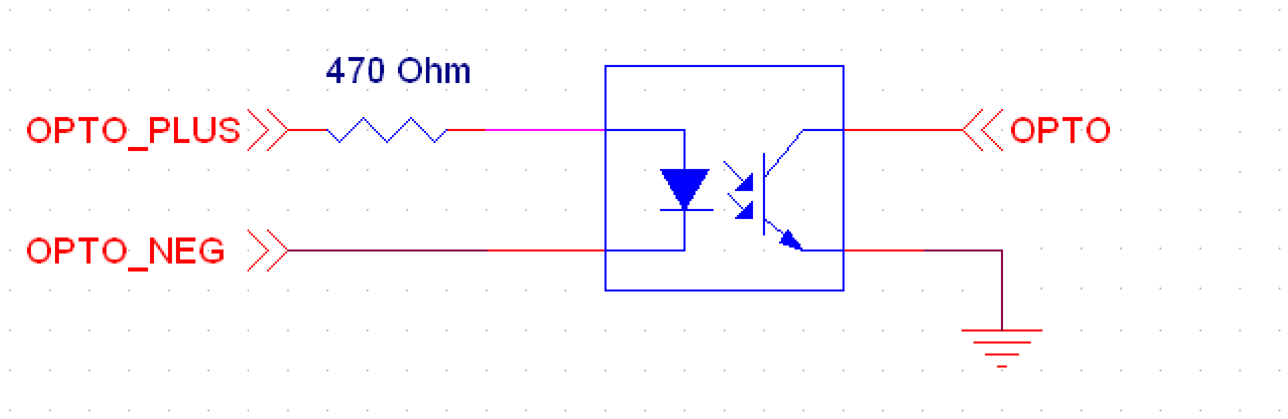
JP7 - I/O - General Purpose LVTTL - OPTO Isolated - Differential - Encoder Inputs

JP7 Is used for all Digital I/O. Fourteen General Purpose LVTTL I/O, Eight Differential Encoder Inputs, and Eight Optically Isolated Inputs.


2 4 6
46 48 50
1 3 5
45 47 49


JP7
IO0 1
IO1 3
IO2 5
IO3 7
IO4 9
IO5 11
IO6 13
CHA_DIFFPLUS_0 15
CHB_DIFFPLUS_0 17
CHA_DIFFPLUS_1 19
CHB_DIFFPLUS_1 21
CHA_DIFFPLUS_2 23
CHB_DIFFPLUS_2 25
CHA_DIFFPLUS_3 27
CHB_DIFFPLUS_3 29
OPTO_NEG_0 31
OPTO_NEG_1 33
OPTO_NEG_2 35
OPTO_NEG_3 37
OPTO_NEG_4 39
OPTO_NEG_5 41
OPTO_NEG_6 43
OPTO_NEG_7 45
VDD5 47
49
2 IO7
4 IO8
6 IO9
8 IO10
10 IO11
12 IO12
14 IO13
16 CHA_DIFFMINUS_0
18 CHB_DIFFMINUS_0
20 CHA_DIFFMINUS_1
22 CHB_DIFFMINUS_1
24 CHA_DIFFMINUS_2
26 CHB_DIFFMINUS_2
28 CHA_DIFFMINUS_3
30 CHB_DIFFMINUS_3
32 OPTO_PLUS_0
34 OPTO_PLUS_1
36 OPTO_PLUS_2
38 OPTO_PLUS_3
40 OPTO_PLUS_4
42 OPTO_PLUS_5
44 OPTO_PLUS_6
46 OPTO_PLUS_7
48 VDD5
50
HEADER 25X2

Optically isolated input circuit. 5-12V may be applied. Current requirements at 5V is approximately 6ma and at 12V is approximately 20ma.

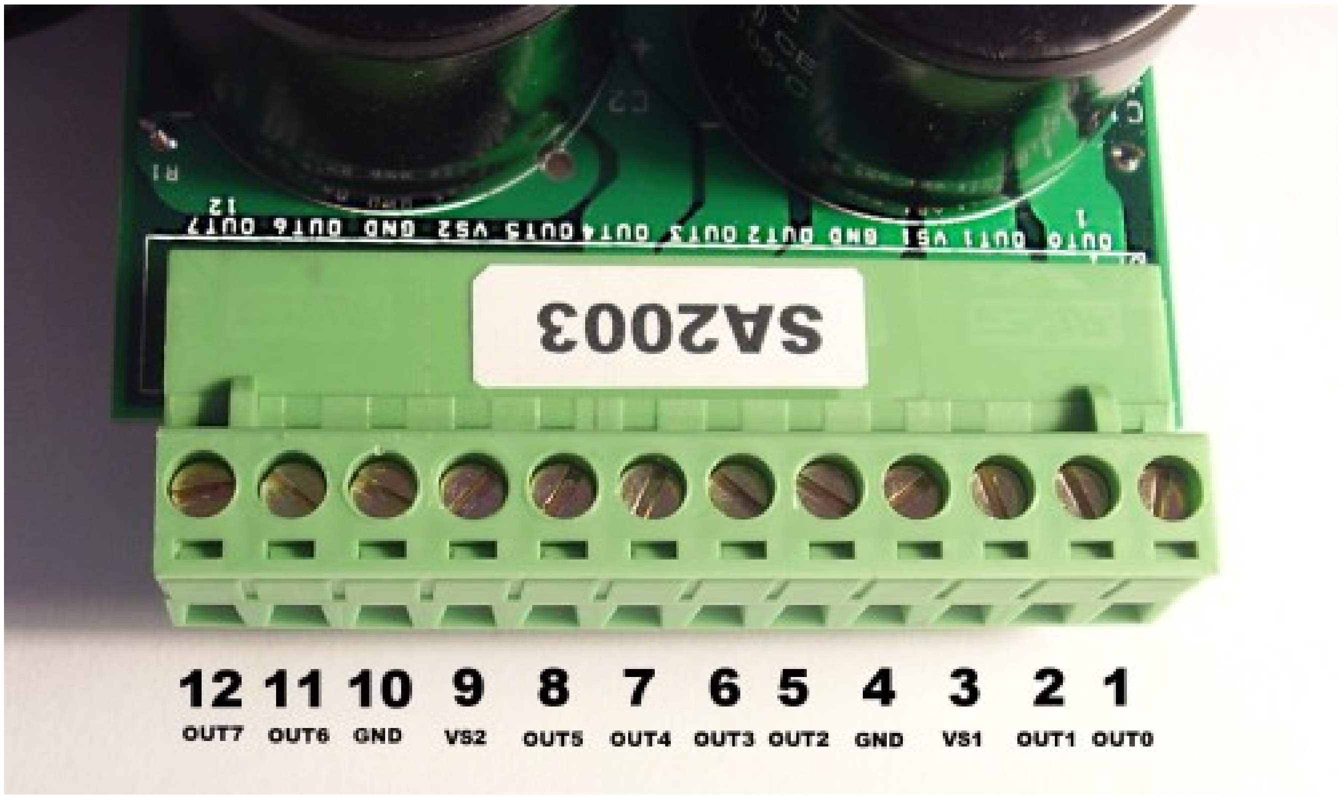


Pin	Name	Description
1	GPIO0	Gen Purpose LVTTL
2	GPIO1	Gen Purpose LVTTL
3	GPIO2	Gen Purpose LVTTL
4	GPIO3	Gen Purpose LVTTL
5	GPIO4	Gen Purpose LVTTL
6	GPIO5	Gen Purpose LVTTL
7	GPIO6	Gen Purpose LVTTL
8	GPIO7	Gen Purpose LVTTL
9	GPIO8	Gen Purpose LVTTL
10	GPIO9	Gen Purpose LVTTL
11	GPIO10	Gen Purpose LVTTL
12	GPIO11	Gen Purpose LVTTL
13	GPIO12	Gen Purpose LVTTL
14	GPIO13	Gen Purpose LVTTL
15	CHA DIFF PLUS 0	Differential Input + Encoder 0 Input Phase A
16	CHA DIFF MINUS 0	Differential Input - Encoder 0 Input Phase A
17	CHB DIFF PLUS 0	Differential Input + Encoder 0 Input Phase B

18	CHB DIFF MINUS 0	Differential Input - Encoder 0 Input Phase B
19	CHA DIFF PLUS 1	Differential Input + Encoder 1 Input Phase A
20	CHA DIFF MINUS 1	Differential Input - Encoder 1 Input Phase A
21	CHB DIFF PLUS 1	Differential Input + Encoder 1 Input Phase B
22	CHB DIFF MINUS 1	Differential Input - Encoder 1 Input Phase B
23	CHA DIFF PLUS 2	Differential Input + Encoder 2 Input Phase A
24	CHA DIFF MINUS 2	Differential Input - Encoder 2 Input Phase A
25	CHB DIFF PLUS 2	Differential Input + Encoder 2 Input Phase B
26	CHB DIFF MINUS 2	Differential Input - Encoder 2 Input Phase B
27	CHA DIFF PLUS 3	Differential Input + Encoder 3 Input Phase A
28	CHA DIFF MINUS 3	Differential Input - Encoder 3 Input Phase A
29	CHB DIFF PLUS 3	Differential Input + Encoder 3 Input Phase B
30	CHB DIFF MINUS 3	Differential Input - Encoder 3 Input Phase B
31	OPTO NEG 0	Opto Isolated Input 0 Negative Connection
32	OPTO POS 0	Opto Isolated Input 0 Positive Connection
33	OPTO NEG 1	Opto Isolated Input 1 Negative Connection
34	OPTO POS 1	Opto Isolated Input 1 Positive Connection
35	OPTO NEG 2	Opto Isolated Input 2 Negative Connection
36	OPTO POS 2	Opto Isolated Input 2 Positive Connection

37	OPTO NEG 3	Opto Isolated Input 3 Negative Connection
38	OPTO POS 3	Opto Isolated Input 3 Positive Connection
39	OPTO NEG 4	Opto Isolated Input 4 Negative Connection
40	OPTO POS 4	Opto Isolated Input 4 Positive Connection
41	OPTO NEG 5	Opto Isolated Input 5 Negative Connection
42	OPTO POS 5	Opto Isolated Input 5 Positive Connection
43	OPTO NEG 6	Opto Isolated Input 6 Negative Connection
44	OPTO POS 6	Opto Isolated Input 6 Positive Connection
45	OPTO NEG 7	Opto Isolated Input 7 Negative Connection
46	OPTO POS 7	Opto Isolated Input 7 Positive Connection
47	VDD5	+ 5V Encoder Power Output
48	VDD5	+ 5V Encoder Power Output
49	GND	Digital Logic Ground
50	GND	Digital Logic Ground

JP1 - Motor/Motor Supply (10-80V) Connector



	Motor type DC -Brush	Motor type - 3 Phase brushless	Motor type - Stepper
Axis 0	Connect Motor across OUT0-OUT1 Specify PWM Channel 8	Connect Phase A to OUT0 Connect Phase B to OUT1 Connect Phase C to OUT2 Leave OUT3 disconnected Specify PWM Channel = 8	Connect Coil A across OUT0-OUT1 Connect Coil B across OUT2-OUT3 Specify PWM Channels = 8 and 9
Axis 1	Connect Motor across OUT2-OUT3 Specify PWM Channel 9		
Axis 2	Connect Motor across OUT4-OUT5 Specify PWM Channel 10	Connect Phase A to OUT4 Connect Phase B to OUT5 Connect Phase C to OUT6 Leave OUT7 disconnected Specify PWM Channel 10	Connect Coil A across OUT4-OUT5 Connect Coil B across OUT6-OUT7 Specify PWM Channels= 10 and 11
Axis 3	Connect Motor across OUT6-OUT7 Specify PWM Channel		

	11			