SmartMove

Installation Manual

Limited Warranty
For a period of one (1) year from the date of original purchase, BALDOR will repair or replace without charge controls which our examination proves to be defective in material or workmanship. This warranty is valid if the unit has not been tampered with by unauthorized persons, misused, abused, or improperly installed and has been used in accordance with the instructions and/or ratings supplied. This warranty is in lieu of any other warranty or guarantee expressed or implied. BALDOR shall not be held responsible for any expense (including installation and removal), inconvenience, or consequential damage, including injury to any person or property caused by items of our manufacture or sale. (Some states do not allow exclusion or limitation of incidental or consequential damages, so the above exclusion may not apply.) In any event, BALDOR's total liability, under all circumstances, shall not exceed the full purchase price of the control. Claims for purchase price refunds, repairs, or replacements must be referred to BALDOR with all pertinent data as to the defect, the date purchased, the task performed by the control, and the problem encountered. No liability is assumed for expendable items such as fuses.
Goods may be returned only with written notification including a BALDOR Return Authorization Number and any return shipments must be prepaid.

Baldor Optimised Control Ltd.
178-180 Hotwell Road
Bristol
BS8 4RP
U.K.
Tel: (+44) (117) 987 3100
FAX: (+44) (117) 987 3101

Safety Information

Safety Notice:

Only qualified personnel should attempt the start-up procedure or troubleshoot this equipment.

This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper use can cause serious or fatal injury. Only qualified personnel should attempt to start-up, program or troubleshoot this equipment.

Operating Environment safety is discussed in Section 10.

Precautions:

⚠ **WARNING:** Do not touch any circuit board, power device or electrical connection before you first ensure that no high voltage present at this equipment or other equipment to which it is connected. Electrical shock can cause serious or fatal injury. Only qualified personnel should attempt to start-up, program or troubleshoot this equipment.

⚠ **WARNING:** Be sure that you are completely familiar with the safe operation of this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper use can cause serious or fatal injury. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

⚠ **WARNING:** Be sure that you are completely familiar with the safe programming of this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper programming of this equipment can cause serious or fatal injury. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

⚠ **WARNING:** Be sure all wiring complies with the National Electrical Code and all regional and local codes. Improper wiring may result in unsafe conditions.

⚠ **WARNING:** The stop input to this equipment should not be used as the single means of achieving a safety critical stop. Drive disable, motor disconnect, motor brake and other means should be used as appropriate. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

⚠ **WARNING:** Improper operation or programming of the control may cause violent motion of the motor shaft and driven equipment. Be certain that unexpected motor shaft movement will not cause injury to personnel or damage to equipment. Peak torque of several times the rated motor torque can occur during control failure.

⚠ **WARNING:** The motor shaft will rotate during the homing procedure. Be certain that unexpected motor shaft movement will not cause injury to personnel or damage to equipment.

⚠ **CAUTION:** To prevent equipment damage, be certain that the input power has correctly sized protective devices installed.

⚠ **CAUTION:** To prevent equipment damage, be certain that input and output signals are powered and referenced correctly.

⚠ **CAUTION:** To ensure reliable performance of this equipment be certain that all signals to/from the controller are shielded correctly.

Table of Contents

1. Read Me First

This manual contains information for installing and commissioning the *SmartMove* programmable position controller. The manual is split into the following main sections:

- **Overview:** Sections 1 through to 6. This contains an overview of the *SmartMove* controller, MINT"™, the motion programming language and options available for *SmartMove*.
- **Getting Started**: Sections 7 and 8. This section contains details of getting started with *SmartMove*, checking wiring and tuning the motors.
- **Introduction to MINT Programming Language:** Section 9. This provides an overview of the MINT motion programming language.
- **Hardware Guide:** Sections 10 – 12 provide a detailed description of the hardware interfaces on *SmartMove*. For example, the digital and analog I/O is discussed.
- **cTERM:** Section 13. Provides information on *cTERM*, the Windows Terminal Emulator for *SmartMove*.

It is advisable to read Section (7), Getting Started, if unfamiliar with *SmartMove*.

2. Key to Symbols Used in this Manual

Throughout this section various icons are used to indicate specific functions:

The screwdriver icon indicates that it is necessary to make a physical connection to SmartMove by way of the screw terminations on the front panel of the controller.

The disk icon together with filename is used to indicate that a MINT program (the motion control language used to program SmartMove) should be downloaded to the controller. The filename indicates the name of the buffer.

The prompt icon indicates that the following commands should be typed in directly to the terminal at the MINT **P>** or **C>** prompt.

[Ctrl]+[E] Type **Ctrl** and **E** at the same time.

3. Hardware Features

The SmartMove controller is a programmable high performance 16 bit motion controller contained within a single compact box and used for controlling up to 3 axes of servo motors. The major features of the controller are illustrated in Figure 1:

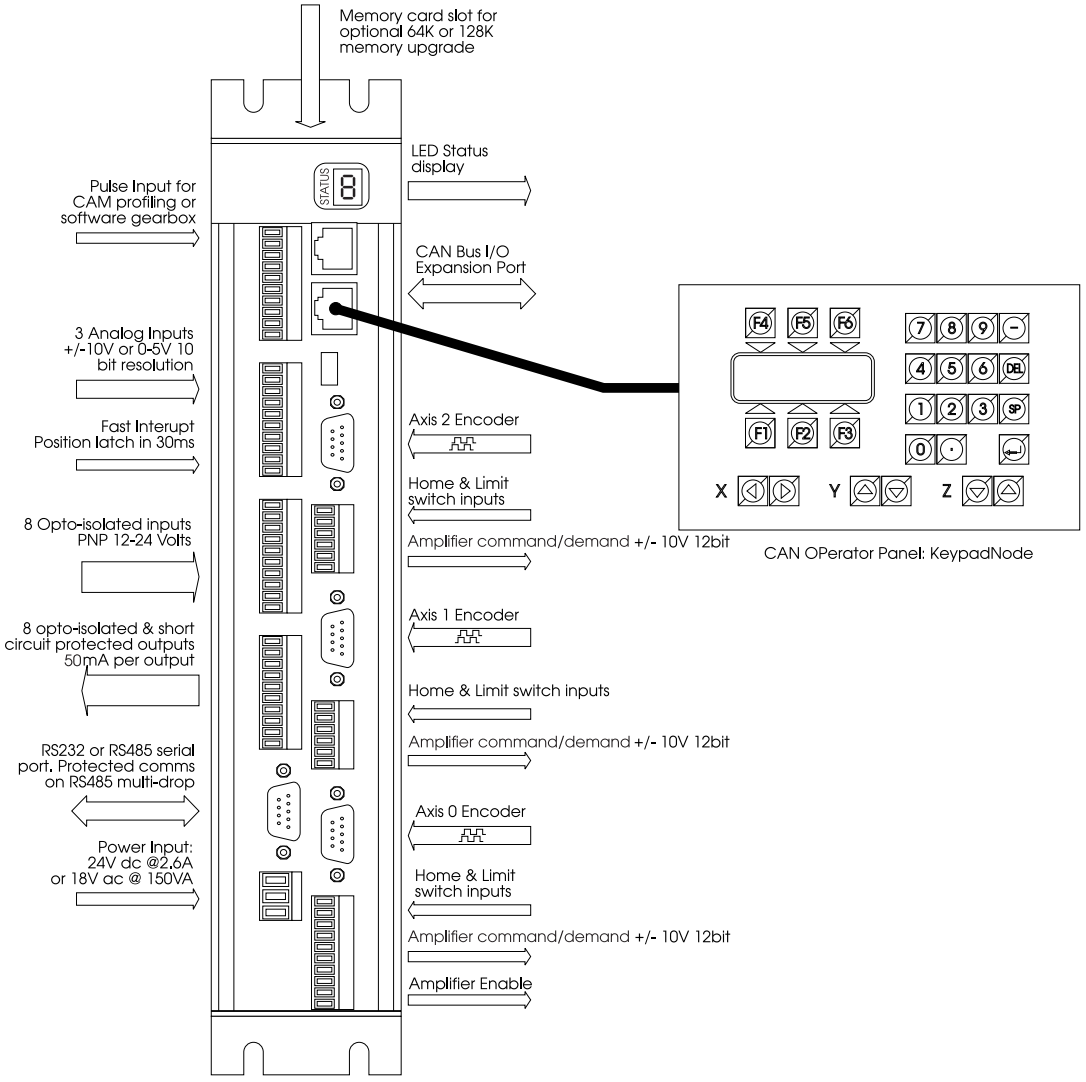


Figure 1: SmartMove Hardware

Features

- Three product variants for one, two or three axes of position control for closed loop servo motors or inverters.
- Incremental encoder feedback: two channel plus index.
- 12 bit analog +/-10V servo amplifier outputs. Programmable for 0-10V with direction.
- Stand-alone operation or host computer controlled over RS232/485 link. Up to 16 controllers on RS485 multi drop link.
- Panel mount box - bookcase format.
- Credit card memory interface for program storage, motion profiles or teach data.
- 8 uncommitted digital inputs for machine control with interrupt capability. Opto-isolated.
- 8 uncommitted outputs (short-circuit protected), for machine control. Opto-isolated.
- Limit and Home inputs per axis. Opto-isolated.
- External error input.
- High speed position latch input for applications such as print registration.
- Three 10 bit analog inputs providing interfaces to joy-stick or sensors.
- Pulse follower input for software gearboxes.
- 24VDC/18VAC input onboard power supply. Provides regulated voltage outputs for incremental encoders.
- Easy to use Basic like motion control language, MINT™, with onboard program editor:
    - 26k bytes non-volatile program/data memory.
    - 3 axis linear interpolation.
    - Optional circular interpolation.
    - Cam profiling and flying shears.
    - Software Gearboxes.
    - Named variables and subroutines.
    - Interrupt capability.

3.1 Technical Specification

3.1.1 Machine Control I/O

Limit Inputs:	• 1 input per axis. Provides end of travel protection. Crash stop all axes when active. • PNP Opto-isolated. Connect to normally closed switch to User V+. • Reverse bias protected. • Maximum input voltage: 30V • Operating Voltage: 12-24V • Input impedance: 2.2kΩ
Home Inputs:	• 1 input per axis. Provides reference position for the axis. • PNP Opto-isolated. Connect to normally closed switch to User V+. • Reverse bias protected. • Maximum input voltage: 30V • Operating Voltage: 12-24V • Input impedance: 2.2kΩ
Stop Input:	• 1 input per controller. Brings all axes to a controlled stop when active. • PNP Opto-isolated. Connect to normally closed switch to User V+. • Reverse bias protected. • Maximum input voltage: 30V • Operating Voltage: 12-24V • Input impedance: 2.2kΩ
Error Input:	• 1 input per controller. Brings all axes to a crash stop when active. • PNP Opto-isolated. Reverse bias protected. • Software configurable input level.
User Digital Inputs	• 8 input lines. • PNP opto-isolated. Reverse bias protected. • Logical one when floating or pulled low. • Logical zero when connected to User V+. • Maximum input voltage: 30V • Operating Voltage: 12-24V • Input impedance: 2.2kΩ
User Digital Outputs:	• 8 output lines driven by Darlington array. • 50mA continuous source on all channels. • 350mA max source per channel, 500mA max for all 8 outputs on simultaneously. • Over-current and over-temperature protection. An error is flagged to MINT.

Fast Interrupt:	• 1 input per controller. • PNP opto-isolated. Reverse bias protected. • Latches position of all axes within 30µs. • Maximum input voltage: • Input impedance:
Enable Output:	• Single pole double throw relay rated at 1A @ 24V. • 1 output per controller. • Fail safe operation: relay de-energized on an error.
Analog Inputs:	• 3 independent analog channels. • 10 bit resolution. • Jumper selectable for ±10V (differential) or 0-5V operation. • Input impedance: 2.2kΩ
Analog Outputs (drive demand/ command):	• 1 output per axis for motor demand/command signal. • ±10V output (±0.1%). • 12 bit resolution (4.9mV/bit). • Additional analog output provided for system tuning and general purpose use.
Encoder Inputs:	• Encoder input per axis for positional feedback via incremental encoder. • Operates with both single ended (TTL or open collector) or differential (TTL or RS422) output type. Differential is recommended. • Accepts three channel increment encoders (A, B and Z). A and B channels are quadrature decoded. • Minimum requirement: Channel A and B single ended TTL. • Maximum encoder frequency: 4.6MHz quadrature count.
Serial Port:	• RS232 or full duplex 4 wire RS485. • Connections brought out on 9 pin male D-type connector. • 9600 baud (software selectable), 1 start bit, 8 data bits, 1 stop bit, no parity. • RS232 or RS485 selectable via jumper settings (RS485 Requires firmware upgrade).

3.2 Miscellaneous and Mechanical Specification

Power Input:	• 24V dc at 2.6A. • 18V ac at 150VA.
Onboard Power Supply:	• On board power supply supplies the following: • Regulated +5V output at 500mA (not to be used for machine I/O). • Regulated (12V outputs at 200mA (not to be used for machine I/O). • 24V output for machine I/O. • 5V regulated output for the incremental encoders.
Operating Temperature:	• 0 - 45°C/32° - 113°F.
Battery Life:	• 5 years. • Non-volatile RAM contents retained for up to 12 months when fully charged.

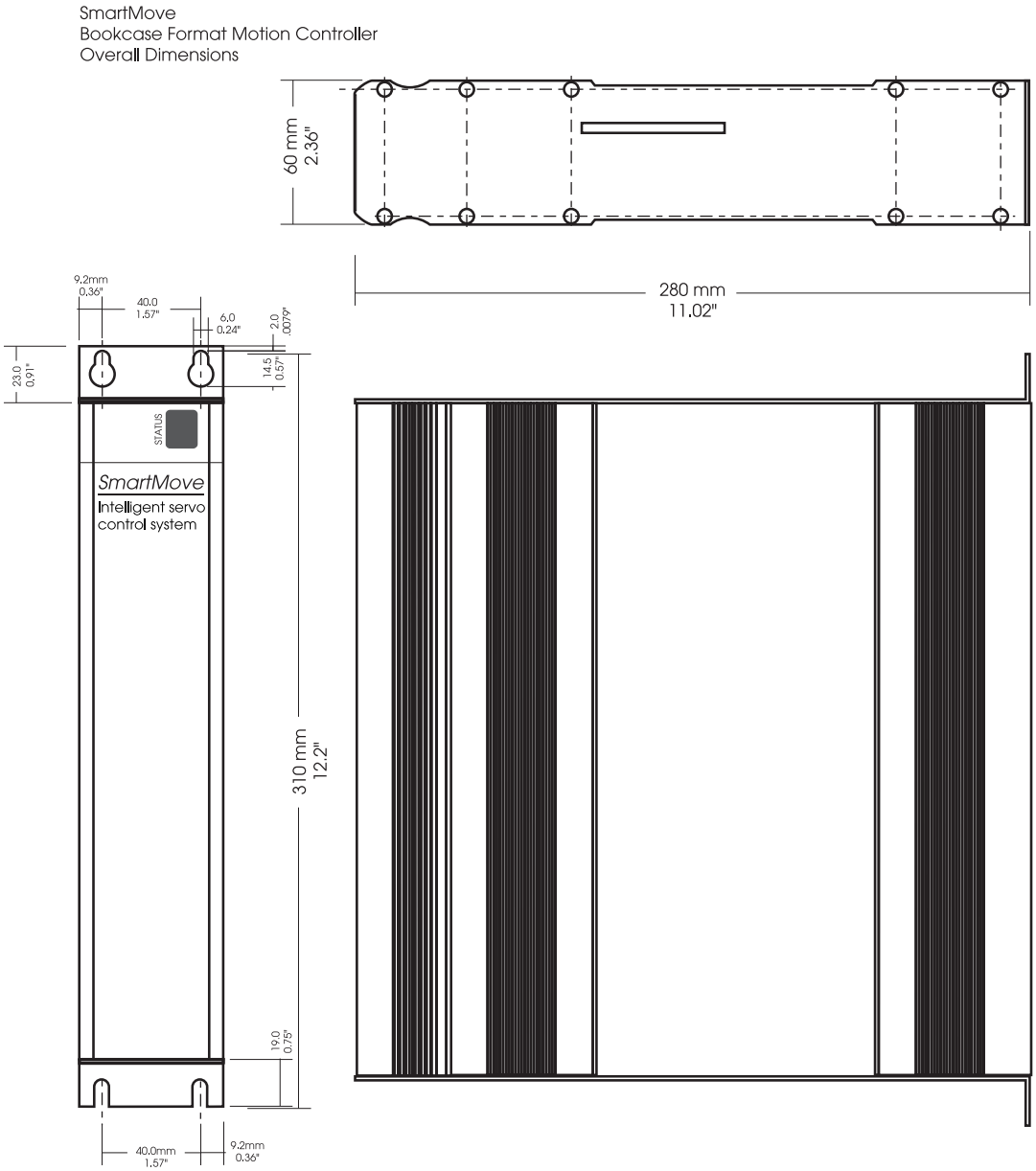


Figure 2: *SmartMove* Bookcase Format Overall Dimensions

4. MINT™ Features for SmartMove

MINT is a structured form of Basic which has been custom designed for motion control applications. The MINT language has been written to allow users to get started quickly and run simple motion programs, while also providing a wide range of more powerful commands for complex applications. MINT's onboard line editor and command line interface allows simple programs to be created quickly with only the aid of a terminal emulator. *cTERM* is a pre-configured Windows terminal emulator supplied with *SmartMove*. The installation disks can be found in the back of the manual.

MINT is used in thousands of applications worldwide, servicing many high demand industries such as textiles and packaging. Applications range from simple single axis applications to complex multi-axis, multi-controller applications via a multi-drop RS485 host-controlled link. It is MINT's flexible and powerful command set that is able to provide a solution to the vast number of industrial motion control applications. Some examples of these applications include:

- Packaging machines
- Multi-axis screen printers
- Milling machines and lathes
- Print registration
- High precision test machines
- Spin welding
- Textile robots

MINT Features for SmartMove:

- Support for up to 3 axes of control.
- Basic type programming language with commands such as **`PRINT, IF..THEN`** and **`FOR..NEXT`**
- Variable definitions where each variable can be given a meaningful name of up to 10 characters in length.
- Support for array variables. Size is limited only by available memory which can be extended using a RAM card.
- Subroutines are referenced by name rather than by a line number. A name can be up to 10 characters in length
- Extensive support for terminal I/O both over the serial and CAN based operator panel known as *Keypad*Node.

- Extensive support for digital and analog I/O including the ability to generate interrupts on digital inputs.
- Many move types including: speed control, positional control, torque control, cam profiles, flying shears, software gearboxes.
- Error recovery from position errors, end of travel limits and external errors.
- Protected communications over single point RS232 or multi-drop RS485 links allowing data transfer to an executing program.
- Trapezoidal and 'S' curve velocity profiles with separate acceleration and deceleration.
- Password protection on programs.

In addition to usual Basic type commands such as **PRINT, FOR..NEXT** and **IF..THEN**, MINT has a number of keywords dedicated to Motion Control and input/output. Keywords are provided for:

- Speed and positional control.
- Torque control; linear and circular interpolation.
- Cam profiling and flying shears.

There is also full control over basic motor control parameters such as the servo loop in addition to all the digital and analog I/O on the controller.

MINT splits motion keywords into 2 categories, motion variables and motion commands. Motion variables are keywords that can be read or written (although not necessarily both). For example:

```
SPEED[0,1] = 10,20
ACCEL[0,1] = 100,200
MOVER[0,1] = 50,100 : GO[0,1]
```

The use of the square brackets controls 2 axes in the system, where axis 0 is the first axis followed by axis 1 and so on. The use of the square brackets is optional as MINT is very flexible in its multi-axis syntax. In this example, a speed of 10 units/s, acceleration of 100 units/s^2 and a relative move of 50 units has been set-up on axis 0. **SPEED, ACCEL** and **MOVER** are all motion variables. The motion command **GO** is used to initiate motion. As with most motion keywords, they can be abbreviated to 2 letters, for example: **SPEED** can be replaced with **SP** and **MOVER** replaced with **MR**. This saves both on code space and typing.

The utility called *Squash* enables programs to be compressed by replacing keywords with their abbreviated counterparts, for example. Refer to Section 13.10.

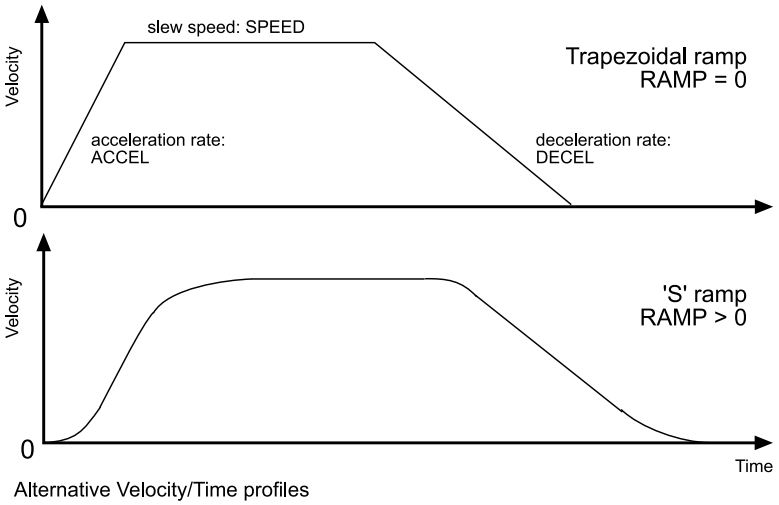


Figure 3: Trapezoidal move with S ramping applied

Moves are based on a trapezoid, which have separately configurable acceleration and deceleration parameters. MINT also supports 'S' curve profiles by use of the RAMP keyword.

At any time during motion, the position of the motor can be printed using:

```
PRINT POS
```

To print the position of axis 1, the following would be used. Note the use of the dot to signify the axis number:

```
PRINT POS.1
```

The use of *units* may have been noted in the example above. This is because engineering units can be applied to an axis. In a linear table for example, there may be 1000 encoder counts per mm. The keywords **SCALE** (or **SF** for short) can be used to set the new scale factor of the axis:

```
SCALE = 1000
```

will enable positions to be specified in mm and speeds in mm/s.

The following shows a more complete MINT example:

```
REM Program: XY example
REM

REM Define 10 XY positions
DIM xpos(10) = 10,10,10,20,30,40,40,40,30,20
DIM ypos(10) = 10,20,30,30,30,30,20,10,10,10

GOSUB init
GOSUB main
END

#init
  HOME = 0,0       : REM Home both axes
  PAUSE IDLE[0,1] : REM Wait for axes to stop
RETURN


#main
  REM Repeat forever
  LOOP
    REM Move to the 10 points
    FOR a = 1 to 10
      REM Move to the absolute position
      MOVEA = xpos(a),ypos(a) : GO
      PAUSE IDLE[0,1]: REM Wait for axes to stop
      OUT0 = 1        : REM Set an output (head down)
      PAUSE IN0       : REM Wait for head to be down
      OUT0 = 0        : REM Move the head up
    NEXT
    GOSUB init     : REM Home the axes again
    PAUSE IN1      : REM Wait for input to start again
  ENDL
RETURN
```

Although this is a simple example, MINT is equally adapted to more complex applications with keywords for print registration, product infeed, flying shears and cam profiling. MINT structure and examples are described in more detail later in this manual.

The example given would be loaded into the *Program buffer* for execution. The *Program buffer* is used to store the application program for the controller. In addition to the *Program buffer*, there is a *Configuration buffer*. This stores a file usually containing the configuration parameters of the system, such as system gains, speed and accelerations. *Program* and *Configuration buffers* are explained in more detail in later sections.

Using the fast interrupt capability on the controller, the position of all axes can be latched within 30µs. This latched position can be used as a product reference for product infeed or position verification. In addition to latching the position, a MINT interrupt routine can be called to perform a function associated with the input, immediately after it happens.

```
#FASTPOS
  ax1 = FASTENC.0
  ax2 = FASTENC.1
  OFFSET.0 = ax1 - x2 : REM make up the difference
RETURN
```

MINT supports a number of interrupt sources in addition to the fast interrupt. These are eight interrupts for the general purpose inputs and one from the stop input. Interrupts are placed in the program by simply adding a pre-defined subroutine to the program:

```
#IN0
  PAUSE IDLE
  MOVER = 10 : GO
  PAUSE IDLE
RETURN
```

This attaches an interrupt to input 0 and will result in a move relative of 10 units in response to a falling edge in the input.

Interrupts have the advantage of working in the background without affecting program flow. Within MINT there are few occasions where program execution is halted. A move can be set-up to operate in the background, at the same time, the program can be checking inputs, writing to outputs and checking position:

```
MOVEA = 100 : GO  : REM Absolute move of 100
PAUSE POS > 50    : REM Wait for position of 50
OUT1 = 1          : REM Set an output
```

This can occur with something more complex; a cam profile for example:

```
DIM cam0(11) = 10,1,2,3,4,5,6,7,8,9,10
MASTERINC = 100
CAMA = 1             : REM Start the cam
PAUSE CAMINDEX > 4  : REM Wait for segment 4 to execute
OUT1 = 1             : REM and fire an output
```

Cams are a special move type and take the place of mechanical cams. In MINT, a series of positions on the slave (the cam) are referenced against a series of positions on the master. This is given in the form of a table which can then be executed as a one shot or a continuous cam. The cam can also be triggered from the fast interrupt, allowing the cam to be re-referenced each turn. Cams once executed, operate in the background and give full control back to the program.

In addition to the motion control features, MINT provides sophisticated keywords for controlling terminal output, whether this is the *SmartMove* operator panel or a VT100 compatible terminal at the end of an RS232 cable. Keywords are available for locating the cursor on the screen, for printing messages and for formatting input and output. With the CAN Operator Panel, *Keypad*Node, MINT also has the ability to detect when a key is pressed. This is beneficial for operator jog control.

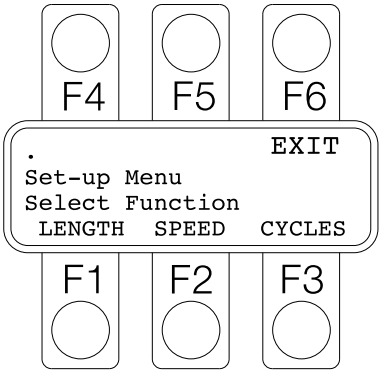


Figure 4

4.1 Systems with more than Three Axes

SmartMove supports a maximum of 3 axes on one controller. However, using a host computer based system it is possible to connect up to 16 controllers on to a multi-drop RS485 serial link and to issue instructions to these controllers using specially constructed data packets.

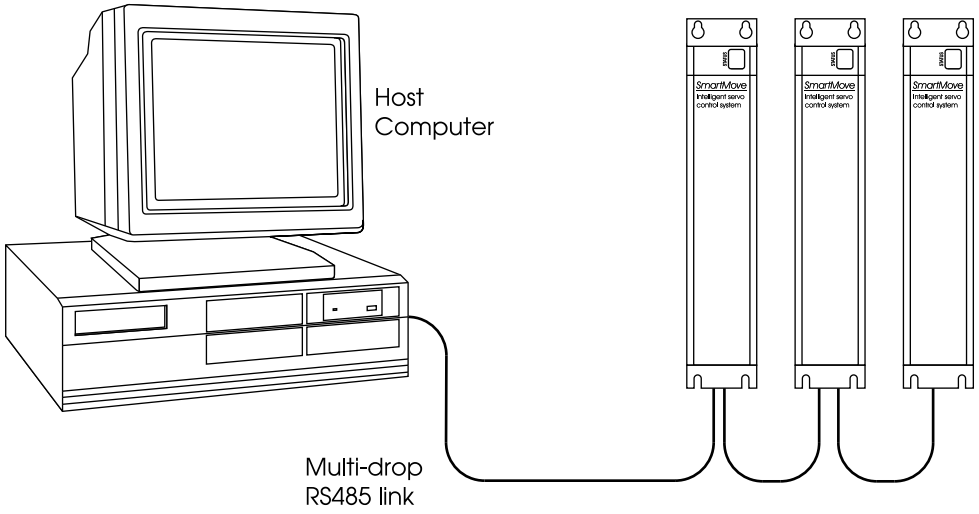


Figure 5: Multi-drop system

Using standard MINT, it is possible to switch from controller to controller over a multi-drop link by sending $ followed by the card address. The card address is set by the 5 pole DIP switch.

Example:

```
$3
```

This will connect the host to box #3. Having gained access it is now possible to upload and download programs over the multi-drop link.

It is also possible to pass data to an executing program or to issue commands for immediate execution using one of two protocols.

a). MINT Host Computer Communications (Comms Protocol)

Using MINT, data can be passed to and from the controller using a protected data packet placed into a 99 element array on the controller. This is analogous to a dual port RAM with 99 addresses (or a post-box with 99 pigeon holes). The host computer can place data into an address (pigeon hole) for the controller to read and vice versa which occurs during program execution. A typical example is shown below:

```
COMMSON       : REM Turn on COMMS protocol
REM comms addresses are split as follows
REM COMMS(1)  - command 1: Move absolute
REM                        2: Move relative
REM                        3: Home axis
REM COMMS(2)  - speed of move
REM COMMS(3)  - move distance
REM COMMS(4)  - home command

COMMS(1) = 0  : REM Clear the command
LOOP
 command = COMMS(1)
 IF command <> 0 DO
  IF command=1 THEN SP.1=COMMS(2):MA.1=COMMS(3):GO.1:PAUSE ID.1
  IF command=2 THEN SP.1=COMMS(2):MR.1=COMMS(3):GO.1:PAUSE ID.1
  IF command=3 THEN HM.1=COMMS(4):PAUSE IDLE.1
 ENDIF
 COMMS(1) = 0 : REM Send Ack back to host
ENDL
```

b). MINT/3.28

MINT/3.28 is intended specifically for systems where a host computer sends all motion control commands to the controller. These commands are for immediate execution, with the sequencing being performed on the host computer rather than the controller. MINT/3.28 is a data-packet based system used for communications over the RS232 or

RS485 serial communications link. The physical constraints of RS232 allow only point-to-point communication; meaning, one host computer can talk to one controller. RS485 provides longer transmission distances and allows a single host computer to communicate with up to 16 controllers on one multi-drop link. MINT/3.28 is not standard software and requires a firmware change. Contact a local distributor for details.

Both protocols are defined by the ANSI standard 3.28, which uses a simple ASCII based protocol. This is defined in the MINT Programming Manual and can be, for example, programmed on a PLC, with the PLC acting as the system host.

4.2 Keyword Summary

The following is a complete list (in alphabetical order) of all the keywords supported on SmartMove. Abbreviations are given following /.

#	subroutine label
#FASTPOS	fast position interrupt
#IN0..IN7	interrupt subroutines on digital inputs
#ONERROR	onerror subroutine
#STOP	stop subroutine on stop input
:	statement separator
ABORT/AB	abort motion - crash stop all axes
ABS	return absolute value of number
ACCEL/AC	set acceleration
ACCELTIME/AT	time to reach SPEED in ms
ANALOGUEx/Ax	read 10 bit analog inputs (1, 2 or 3)
AND/&	logical (bitwise) AND
AUTO	auto-execute program on power-up
AUX	write velocity or following error to DAC
AUXDAC	write direct to 4th analog output
AXISCON	configure axis attributes
AXES	set default axis numbers
BACKOFF/BA	set home speed backoff factor
BAUD	set the serial baud rate
BEEP	issue a beep at the user terminal
BEEPON	issue a beep during INPUT
BEEPOFF	do not issue a beep during INPUT
BINARY	print number as an 8 bit binary string
BOL	place cursor at beginning of line
CAM	start move on CAM table
CAMA	start move on absolute CAM table
CANBAUD/CB	set the CAN baud rate
CANSTATUS/CNS	read the cause of the CAN event or error
CANCEL/CN	clear error/cancel current move
CAPTURE/CP	begin data capture
CARD/CD	return card address number
CIRCLEA/CA	circular interpolation absolute (2 axes)
CIRCLER/CR	circular interpolation relative (2 axes)
CHR	output non-ASCII characters
CLS	clear the screen
COMMSON	enable protected communications mode
CONFIG/CF	configure axis for off/servo/inverter
CONTOFF/CO	contouring off
CONTON/CT	contouring on
COS	return cosine of angle
CURRLIMIT/CL	set current limit of DAC
DAC/DC	write direct to 12 bit DAC output
DATATIME/DT	data capture time
DECEL/DE	set deceleration
DEFAULT/DF	return all motion parameters to default values
DEMAND/DM	return instantaneous motor demand
DIM	reserve space for array data
DINT	disable interrupts
DISLIMIT/DL	disable limit switches
DISPLAY	list all defined variables and array values
ECHO/EO	control echoing of command line and error messages
EINT	enable interrupts
ENABLE/EB	enable/disable drives
ENCODER/EN	return the encoder position
ENCWRAP/EW	set a wrap around value on ENCODER value
END	end program execution
ENLIMIT/EL	enable limit switches
ERROR/ER	return axis error condition
ERRORIN/EI	set error input state
EXIT	terminate current loop structure
FASTENC/FC	read latched encoder position
FASTPOS/FP	read latched position from fast interrupt
FOLERR/FE	return instantaneous following error

FOLLOW/FL	enable encoder following (software gearbox)
FOLLOWAXIS/FA	set which axis to follow for encoder following
FOR {STEP} NEXT	Basic for next loop
FLY/FY	setup flying shear to a master position
FREE	display free memory
GAIN/GN	servo loop proportional gain
GO	begin synchronized motion
GOSUB	branch to subroutine
GOTO	continue execution from label
HMSPEED/HS	set speed axis seeks home switch
HOME/HM	seek home position/read home input value
IDLE/ID	return true when idle/set idle following error
IF DO {ELSE} ENDIF	block IF structure
IF THEN	standard IF THEN statement
IMASK/IM	interrupt mask on user inputs (#INx)
IN	read 8 bit value of digital inputs
INx/Ix	read digital inputs 0 to 7
INCA/IA	set new end position: absolute
INCR/IR	set new end position: relative
INKEY/IK	return ASCII character from serial/keypad buffer
INPUT	accept numeric input with optional formatting
INT	return integer value of number
IPEND/IP	user interrupts pending
JOG/JG	speed control
KEYS	set keypad key values and enable keypad
KINT/KI	servo loop integral gain
KINTRANGE/KR	servo loop integration range
KVEL/KV	servo loop velocity gain
KVELFF/KF	servo loop velocity feed forward gain
LASTERR	display last interpreter error message
LIMIT/LM	read limit switch input values
LINE	print a string to a specified line
LOAD	load a file
LOCATE	locate cursor on terminal
LOOP .. ENDL	loop forever
LOOPTIME/LT	set servo loop time
MASTERINC/MI	master link position for CAM profiling
MFOLERR/MF	set maximum following error
MOVECAM/MC	move one cam segment
MOD/%	modula arithmetic operator
MODE/MD	return current mode of motion
MOVEA/MA	positional move absolute
MOVER/MR	positional move relative
NODELIVE/NL	determine if a node is live or dead
NOT/!	logical not
OFFSET/OF	perform offset during pulse/encoder following
OR/|	logical (bitwise) or
OUT/OT	write 8 bit value to digital (user) outputs
OUTx/Ox	write to user outputs bits 0 to 7
PAUSE	pause program execution until condition is true
POFF	turn prompt off
PON	turn prompt on
POS/PS	read/set axis position
PRESCALE/PR	scales down the encoder input
PRINT/?	prints a string or number with optional formatting
PROTECT	password protection on program
PULSE/PU	enable pulse following (software gearbox)
PULSEVEL/PV	read pulse input velocity
RAMP/RP	velocity profile smooth factor
READKEY/RK	read current key pressed on keypad
RELEASE	clear all user variables from memory
REM	comment
REMOTEAL/RL	set active state of remote I/O
REMOTEBAUD/RB	set the CAN bit rate on the remote device
REMOTEERROR/RR	read the error status from the remote device
REMOTEIN/RI	read the inputs on remote input device
REMOTENODE	set the node ID on the remote device
REMOTEOUT/RO	set the outputs on the remote output device
REPEAT .. UNTIL	loop until condition is true
RESET/RE	re-initialize motion variables
RETURN	return execution from subroutine
RUN	execute a program

SAVE	save a file
SCALE/SF	scale encoder counts to user u nits
SERVOC/SC	turn servo on to current position
SERVOFF/SO	turn servo power off
SERVON/SV	turn servo power on (in previous position)
SIN	return sine of angle
SPEED/SP	set speed for positional moves
STATUSNODE/SN	read the node ID of the CAN device which caused an error
STOP/ST	bring axis to a controlled stop
STOPMODE/SM	set the action taken on the stop input
STOPSW/SS	read stop switch status
TAN	return tangent of angle
TERM/TM	set terminal output to LCD display/serial
TIME/TE	user timer
TIMER/TI	read pulse follower counter value
TORQUE/TQ	set torque control
TRIGGER/TG	trigger move off an input
TROFF	program trace off
TRON	program trace on
VECTORA/VA	vector move absolute (2/3 axes)
VECTORR/VR	vector move relative (2/3 axes)
VEL/VL	return instantaneous axis velocity
VER	display MINT version number
WAIT	wait in milliseconds
WHILE .. ENDW	loop while condition is true
WRAP/WR	set reset value of pulse follower timer
XENCODER/XE	read/set external encoder position
XIO/XI	read/set expanded I/O
XIO0..XIO7	read/set expanded I/O bits
XOUT	read last expanded output value
ZERO/ZR	defines present axis position as zero

Line Editor Commands:

CON	switch to Configuration buffer
DEL	delete lines from a program
EDIT	go to a program line
INS	insert lines into program
LIST	list a program
NEW	clear program from memory
PROG	switch to Program buffer

Operators:

+ - / * < > <= >= < > = MOD/%

ABS INT AND/& OR/| NOT/! COS SIN TAN

4.3 MINT Options

In order to provide extended functionality, SmartMove has different versions of MINT supplied on EPROM. This is commonly known as firmware. The three versions are described:

Process MINT	Process MINT, supplied as standard with SmartMove, incorporates motion control features such as software gearboxes, cam profiles and flying shears. Process MINT supports up to 3 axes of linear interpolation but does not support circular interpolation. Process MINT supports cam profiles and flying shears on the first 2 axes only (axes 0 and 1).
Interpolation MINT	Interpolation MINT provides full linear and circular interpolation within the MINT environment. Interpolation MINT does not support the cam profiling and flying shear capabilities of Process MINT. A subset of Process MINT's software gearboxes are supported.
MINT/3.28	MINT/3.28 is intended for when sequencing data is programmed on a host computer rather than on the controller. It allows the immediate execution of commands. MINT/3.28 does not run a MINT program. MINT/3.28 only supports a subset of the MINT command set, namely the motion control commands. The Process MINT command set is not supported. (Refer to Section 4.1 b).

5. Support Tools, Software and Documentation

The MINT Programmer's Toolkit (supplied on the disks attached with this manual) provides a number of tools and example programs for use with SmartMove. The main application is cTERM for Windows which provides a pre-configured terminal emulator and editor for MINT programs.

5.1 cTERM for Windows

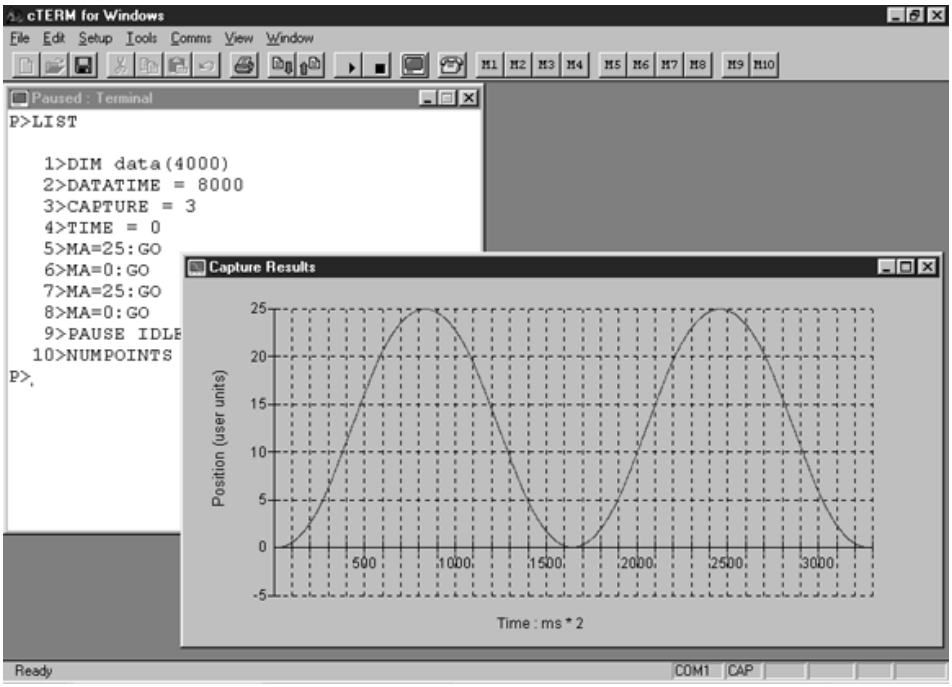


Figure 6: *cTERM* for Windows

cTERM for Windows provides an integrated Windows environment for *SmartMove*. Features include:

- VT100/VT52 terminal emulator.
- Editor for *Program* and *Configuration buffer* with file upload/download.
- Data capture. Using the data capture routines of *SmartMove*, velocity or following error profiles can be captured on the controller and viewed by *cTERM*. Refer to Figure 6.
- Protected Comms Protocol watch window.
- *Squash* utility which allows larger programs to be stored in memory by squashing the original source code.

5.2 cTERM for DOS

Figure 7: cTERM for DOS

cTERM is a pre-configured DOS based terminal emulator for use with *SmartMove*. As well as providing a standard terminal screen with file upload and download facilities, *cTERM* also facilitates the testing of multi-drop systems using the MINT communications protocol and MINT/3.28.

cTERM for DOS will be installed during installing of the MINT Programmer's Toolkit.

5.3 MINT Interface Library

The MINT Interface Library is a CD-ROM containing a set of libraries for host applications on the PC. The MINT Interface Library supports all *Baldor Optimised Control* products. The following features are supported in *SmartMove*:

- Ability to upload and download MINT programs and configurations over the RS232 or RS485 serial port.
- The MINT Comms Protocol, allowing a host application to send data to an executing MINT program. RS485 allows up to 16 *SmartMoves* to be connected to the same link.
- MINT/3.28 commands can be sent to *SmartMove* with simple instructions.

Support is provided for all versions of Windows (Windows 3.11, Windows 95 and Windows NT) allowing 16 and 32 bit applications to be written. With a Dynamic Link Library (DLL) interface, any programming language supporting DLLs can also support the MINT Interface Library. Interfaces have been supplied for the following languages:

- Borland Delphi
- Microsoft Visual Basic
- Microsoft Visual C++
- Borland C++
- C

Contact a local sales office for details.

6. Options

SmartMove supports a number of options from CAN bus I/O expansion boards to program development tools using the *MINT Interface Library*.

Order codes are given in the following table, with a more detailed description of some of the options given in later sections.

Ordering Information:

Item	Order Code	Description
Options:		
CAN Operator Panel (KeypadNode)	KPD002-501	CAN bus based 27 key keypad and 4 line LCD display
InputNode 8	ION001-501	CAN bus 8 digital inputs
OutputNode 8	ION003-501	CAN bus 8 digital outputs
RelayNode 8	ION002-501	CAN bus 8 relay outputs
64K RAM Card	OPT004-501	For program storage or memory expansion
128K RAM Card	OPT005-501	For program storage or memory expansion
Encoder splitter	OPT008-501	Provides two buffered outputs from one encoder input.
Firmware Options:		
Process MINT		Supplied as standard
Interpolation MINT		Supports circular interpolation
MINT/3.28		Host controlled communications
Memory expansion		Refer to section 6.3
Miscellaneous:		
RS232 Serial Cable 3m (10ft)	CBL001-501	9 pin to 9 pin serial cable
SmartMove manual	MN1250	Includes software
MINT Interface Library	SW1259	Libraries for Windows developers

6.1 CAN Operator Panel (*Keypad*Node)

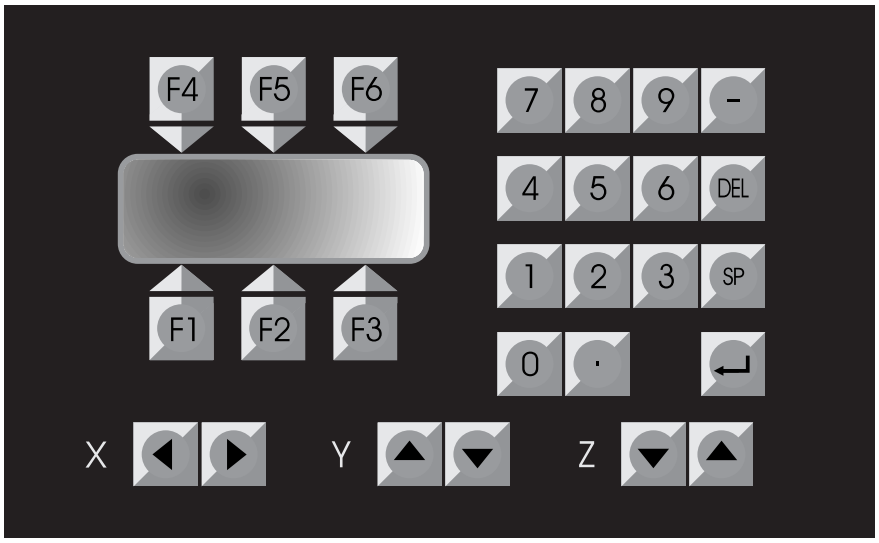


Figure 8: KeypadNode

The CAN Operator Panel (*Keypad*Node) provides a general purpose operator panel suitable for stand-alone machines of all types. *Keypad*Node is cost effective for simple functions, such as replacing thumb wheel switches and providing simple diagnostics. It may also be used as a fully interactive programming panel for special purpose machine control.

*Keypad*Node operates over the CAN bus, providing noise immunity for distances typically up to 500m (approx. 1640ft).

A range of keywords are available in MINT which allow the easy manipulation of display and keypad. Formatted numeric input and output can easily be achieved using the **`PRINT..USING and INPUT..USING`** keywords.

6.2 CAN I/O Nodes

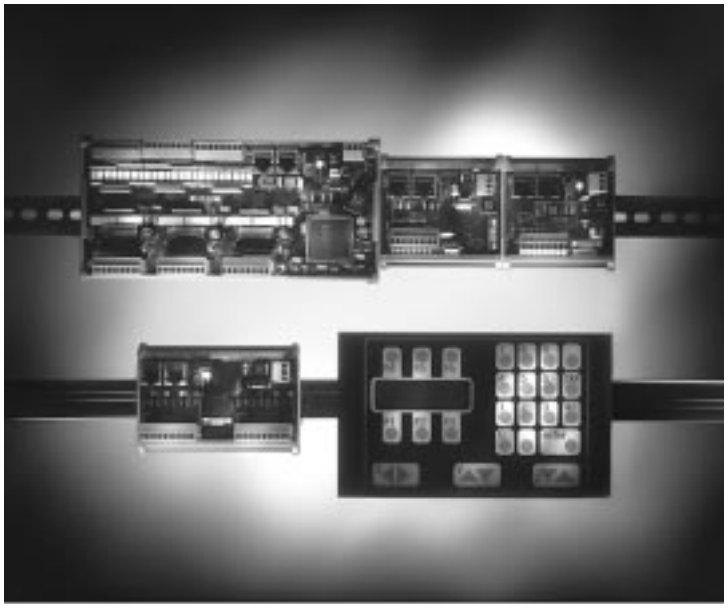


Figure 9: *ioNODE* Product Family

Digital I/O can be expanded easily on *SmartMove* using the *CAN bus interface*. This provides a high speed and secure serial bus interface to a range of I/O devices as described:

- *Input*Node 8: 8 opto isolated digital inputs.
- *Output*Node 8: 8 opto isolated digital outputs with short circuit and over current protection.
- *Relay*Node 8: 8 relay outputs.

These I/O devices, know generically as ioNODE, operate on the same bus as *Keypad*Node, the CAN bus operator panel.

6.3 Memory Card

A memory card interface comes as standard with *SmartMove* and supports either 64K or 128K credit card sized memory cards. The MINT software gives the following standard support for the memory card:

- A *Program and Configuration file* can be stored on the memory card. If the controller detects the presence of a memory card with either valid file during power-up, the *Configuration and Program files* are copied from the memory card to controller memory for execution.
- A copy of the array data within the controllers memory can be stored on the memory card.
- Array data can be stored off-line on the memory card. Array data is defined in the usual way, with the addition of the **`OFFLINE`** keyword. All references to the array data will then occur over the memory card. This allows the memory card to be used as array memory expansion. With a 128K memory card, some 32,000 array elements can be defined.

A change of firmware allows the memory card to be used to increase the size of the *Program buffer*. Memory expansion (denoted by /MX on the version number) stores the *Program and Configuration buffers* on the memory card at all times. During program execution, the code is compiled. It is this compiled code which is stored within the controller for execution.

Memory expansion allows for a theoretical program size of approximately 63K, although this must compile down to under 26K (not including arrays). Off-line array storage is supported with memory expansion as with the normal memory card usage.

6.4 The Encoder Splitter/Buffer Board

This is not an option that fits onto the controllers option bus but is a stand alone PCB that takes an encoder signal, either single ended or differential and gives 2 differential outputs. This is useful for 'daisy-chaining' an encoder signal from a master across a number of controllers.

Refer to Figure 10.

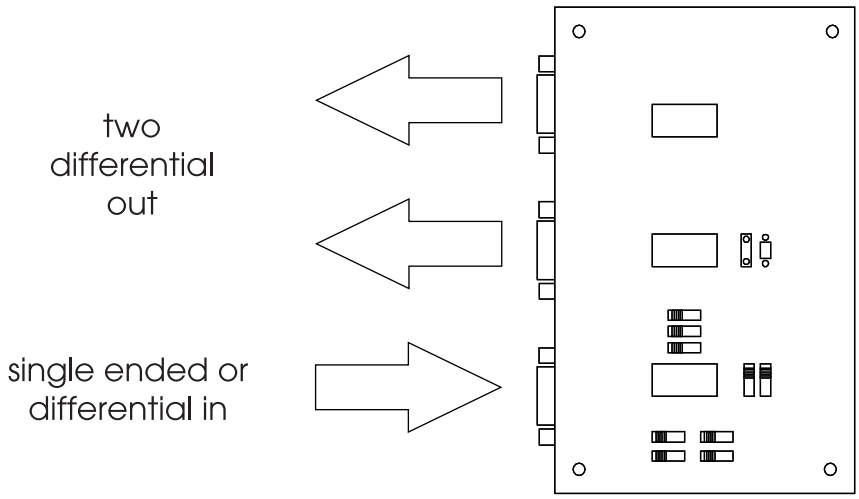


Figure 10: Encoder Splitter/Buffer Board

7. Getting Started

This section covers getting started with the SmartMove servo controller. It is essential to read Section 8, 'Setting Up' before powering up the controller. This section assumes very basic familiarity with a PC. It does however, assume that SmartMove is connected to the appropriate drives and motors and that Windows is installed on the PC. If the configuration is not exactly as described, it is still possible to proceed since much of this guide remains relevant.

7.1 What you need to get Started

Before setting up the controller, the following items required for getting started should be made available:

1. This *SmartMove* Installation Manual.
2. A PC running Windows 3.1 or 95 with at least one free serial port.
3. *cTERM for Windows* installed on your PC. *cTERM* is found on the 2 disks attached with this manual named *MINT Programmer's Toolkit: Serial*. Refer to section 13 for details on installing *cTERM for Windows*.
4. The *SmartMove* controller.
5. The servomotor/drive combination that is intended for use.
6. A small screwdriver (less than 3.5mm (1/8") blade), soldering equipment and some electrical cable.
7. An RS232 cable, or the components necessary to build one. A suitable cable is available from *Baldor Optimised Control* (order code: CBL001-501). If *SmartMove* is being used with an RS485 interface, then an RS232 to RS485 converter for the PC will be required. This allows the signals from the RS232 port to be converted to the signals necessary for RS485 communications.
8. The appropriate controller, amplifier cables, or the connectors in order for them to be constructed. (At least 9 core/conductor screened/shielded cable and 9 pin 'D' type male plugs will be required).

7.2 Introduction to Servo Positioning Systems

A typical closed loop positioning system can be broken down into three elements:

1. *Motor/actuator* – translates electrical power from the servo amplifier into rotary or linear movement. The motor is fitted with a device which feeds the output position of the motor back to the controller.
2. *Servo amplifier* – takes the demand/command signal from the position controller to control the torque or speed of the motor/actuator.
3. *Position controller* – performs real-time positional control of the motor(s), stores the application program and communicates with the user and other control equipment.

The controller works by sampling the position of the motor position at regular intervals and comparing this position with its target position. It then instructs the amplifier to drive the motor to correct any positional error. This process is typically repeated 500 times per second to ensure that the motor is always in the correct position. A typical closed loop control system is illustrated below:

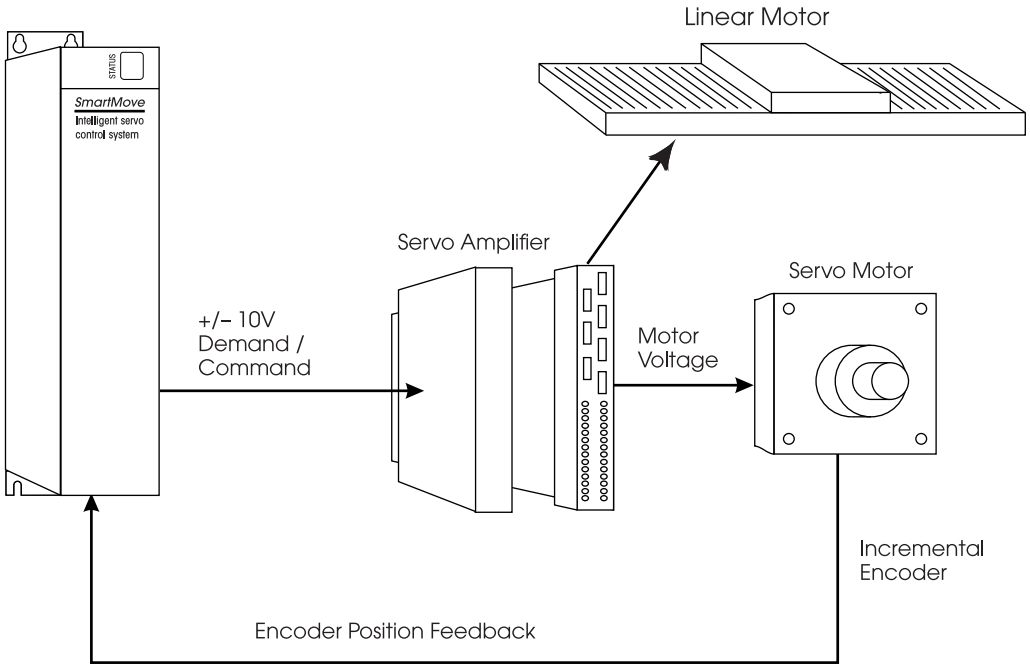


Figure 11: Typical closed loop positioning system

8. Setting Up

The following sections provide a step by step guide to setting up the servo system.

It is worthwhile but not essential to be familiar with the MINT programming language and editor before starting the set-up procedure. A summary of the use of MINT is given in Section 9 of this manual and covered in more detail in the MINT Programming Manual.

8.1 Minimum System Wiring

The controller will work with many servo systems, both electrical and hydraulic, as long as they accept a +/-10V input[1] as a velocity or torque command and provide feedback in the form of an incremental encoder. Resolver based systems can also be controlled when the amplifier provides a resolver to digital or pseudo encoder output which can be input to *SmartMove*.

A minimum system is one where the controller and drive are configured to work with as little external wiring as possible. The following step-by-step example covers setting-up systems with two axes of motion (servo drives and motors). Note that on the controller, there can be between 1 and 3 axes of motion available (called axes **`0,1`** and **`2`** when programming in MINT).

[1]SmartMove will also operate with drives which accept 0-10V plus direction. This is not covered in this Getting Started section. Refer to MINT Programming Manual.

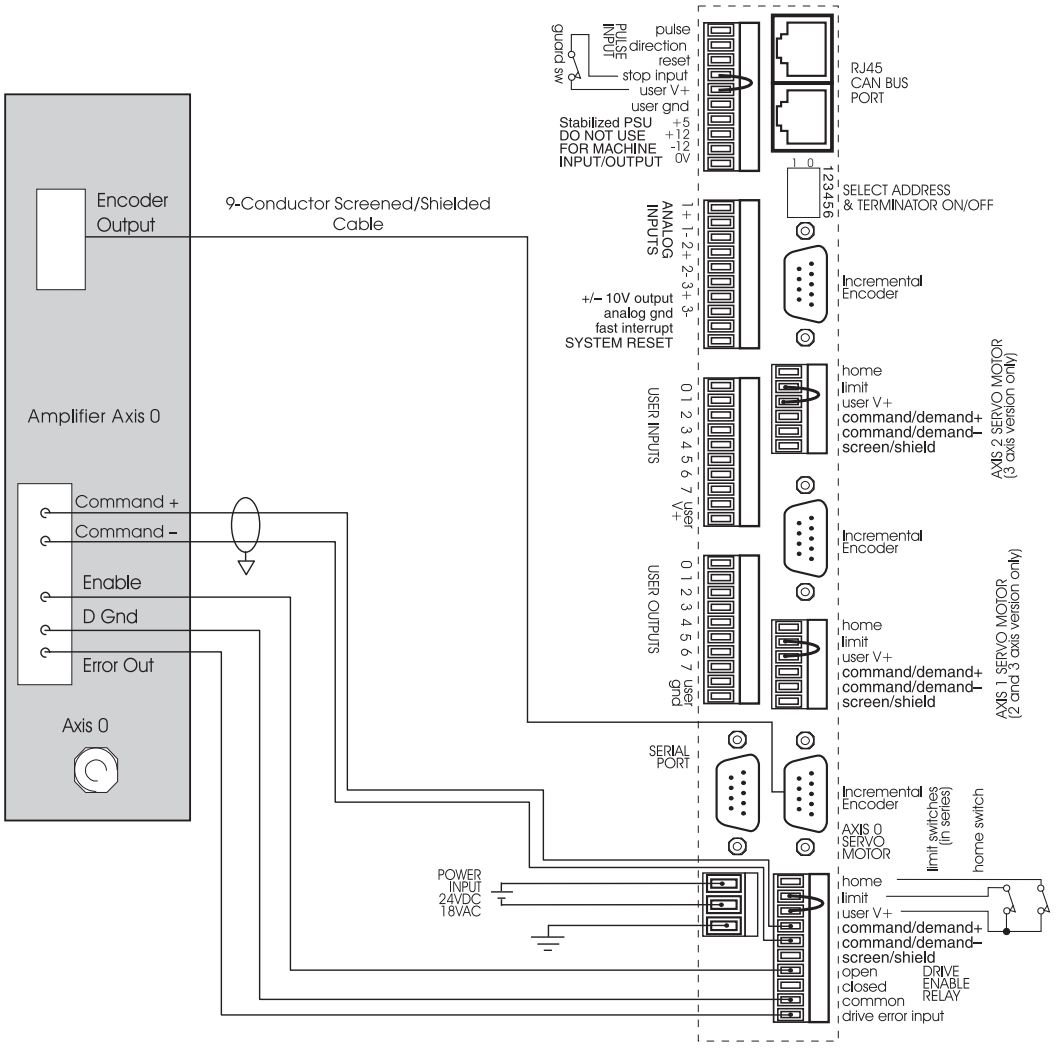


Figure 12: Minimum configuration for a 1 axis system

8.1.1 Power Supply Connections, J6

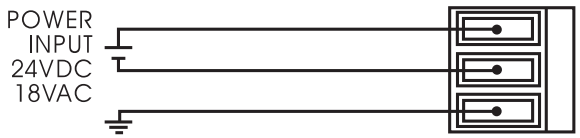


Figure 13: Power Input Connector, J6

The controller has opto-isolation between the microprocessor controller and the inputs and outputs to provide noise immunity required for industrial environments. The power supply unit inputs (24VDC, 18VAC) include isolated grounds.

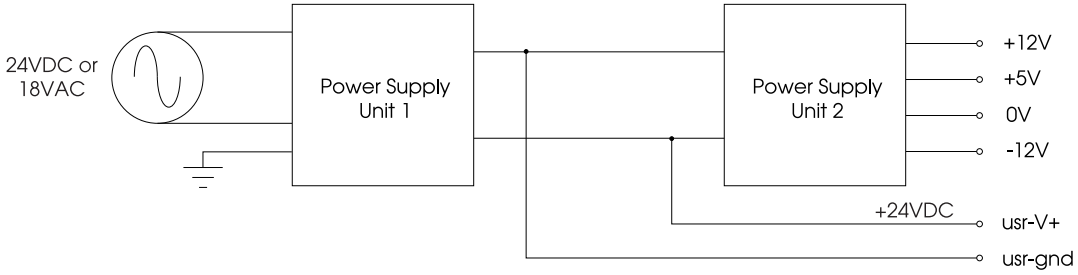


Figure 14: *SmartMove* Power Supply Unit

The unit has an on board power supply to provide the following:

- 24V output for machine I/O
- 5V regulated output for incremental encoders
- +/-12V output, 5V output for general use. Note that these must not be used for machine I/O. Do not attempt to draw more than 200mA from the 12V rails and no more than 500mA from the 5V.

There are two possible input configurations:

- A 24VDC input at 2A.
- An 18VAC input at 150VA.

These are sufficient to drive all the outputs at 50mA and with all the inputs on.

— WARNING —

Applying mains voltages to *SmartMove* (110V/220V) will damage the unit. Ensure that the power input voltages comply.

8.1.2 Limit and Home Switches, J7, J9 and J11

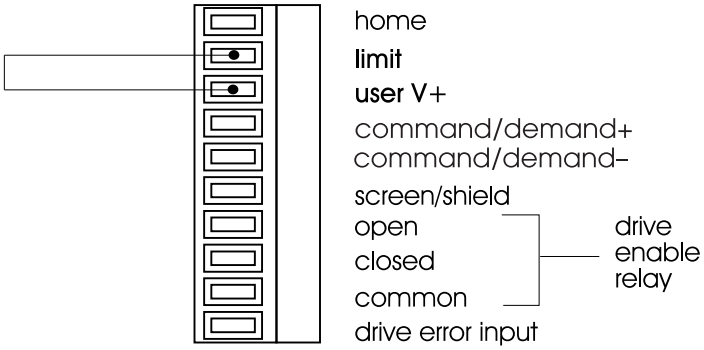


Figure 15: Connections for Axis 0, J7

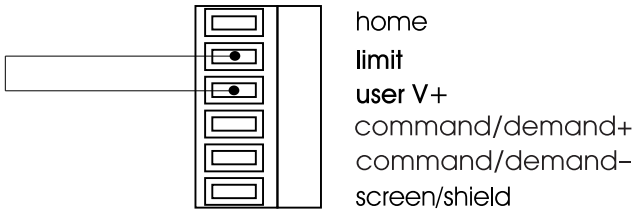


Figure 16: Connections for Axis 1 and 2

The **limit** switch inputs are usually connected through normally closed over-travel switches on the end of the axes of motion and when open or floating, cause motion to stop. To allow motion they must have power supplied to them. If two **limit** switches are connected to one axis, they must be wired in series. Refer to Figure 17. This arrangement means that the controller is fail safe, if a wire breaks, the controller stops the motion.

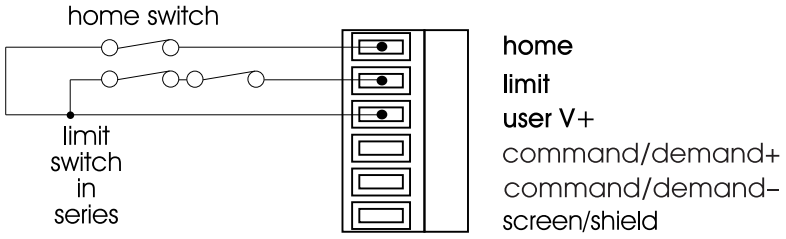


Figure 17: Example home and limit switch wiring in series, J9/J11

If the **limit** input is not properly connected, the controller will display an L on the Status Display on the front of *SmartMove* during power-up. Refer to Section 14.

The home switch (where required) is used as a mechanical datum (homing) reference point, to tie the physical position of the axes to a known position through the switch and in some cases through the "Z" or Index channel of the encoder. For further details on homing refer to 8.4.

8.1.3 Servo Drive Demand/Command, J7, J9 and J11

The controller uses +/- 10V outputs as a **demand/command** voltage for the servo drive. This **demand/command** signal relates to a speed or torque command for the motor. The speed or torque range is determined by the servo drive set-up.

Before continuing with the set-up procedure, please make sure the servo drives have been correctly configured for the motor in accordance with the instructions in the drive manual.

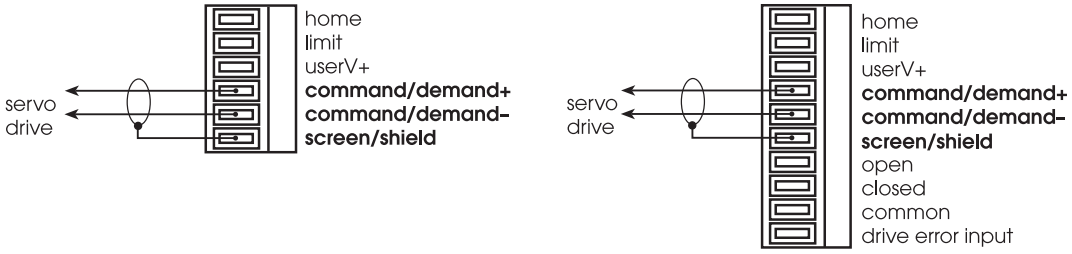


Figure 18: Servo drive connections for all axes

8.1.4 Servo Drive Enable, J7

The **enable** input on the servo drive should be connected to the **enable** output from the controller. The **enable** output from the controller is a relay, where **open** is normally open and **closed** is normally closed and **common** is the common.

It is essential that the servo drives are disabled during power-up of the controller or when a controller error occurs. This ensures that the drives are only able to drive the motors when the controller is performing servo-loop closure.

The actual connections required will differ for the particular configuration of the **enable** input of the drive. Two common examples are given shown in Figure 19 and Figure 20.

If the servo drive **enable** input has to be grounded to enable the drive, the connections should be made as shown in Figure 19:

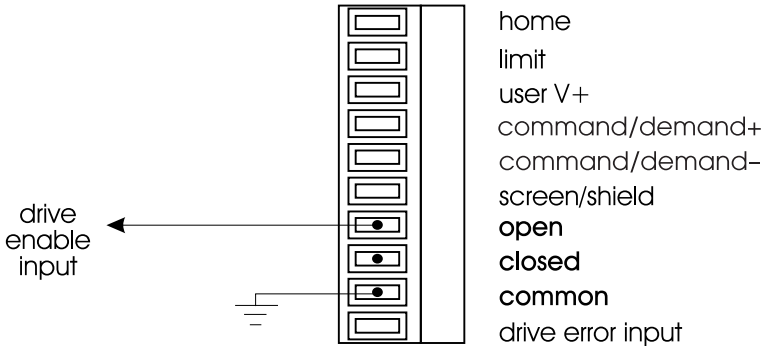


Figure 19: drive enable Example, J7

If the **drive enable** input must be tied to 24V to enable the drive and grounded to disable the drive, the connections should be made as shown in Figure 20:

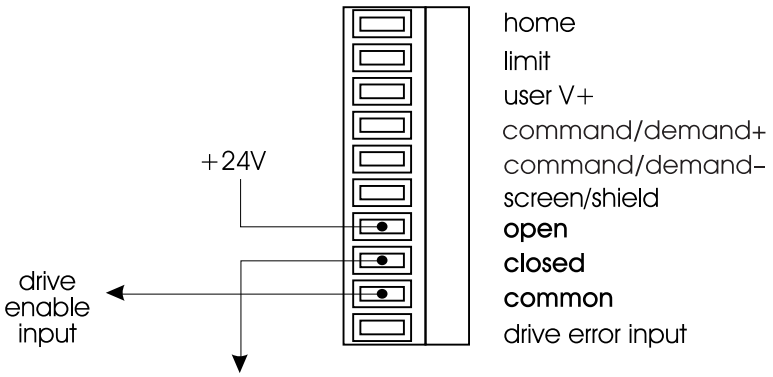


Figure 20: drive enable Example, J7

On power-up, the servo drive should be disabled. The **RESET** command is used to enable the drives. It is not necessary to fully understand the use of the **RESET** command at this stage. Section 8.2.2 covers checking of the enable signal.

8.1.5 Stop switch, J1

NOTE: The stop input to this equipment should not be used as the single means of achieving a safety critical stop. Drive disable, motor disconnect, motor brake and other means should be used as appropriate.

The **stop** switch input causes all axes to decelerate to a halt and is used for connections to machine guards[2]. This input is also normally closed and must have power supplied to the switch in order to allow motion.

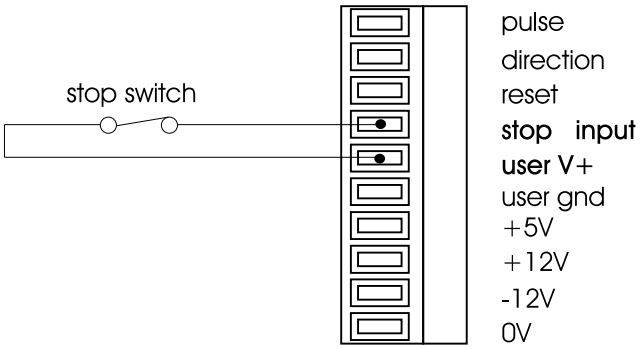


Figure 21: Example stop switch wiring, J1

If the **stop** input is not properly connected, the controller will display an S on the Status Display on the front of *SmartMove* during power-up. Refer to Section 14.

8.1.6 Encoder Connections, J8, J10 and J12

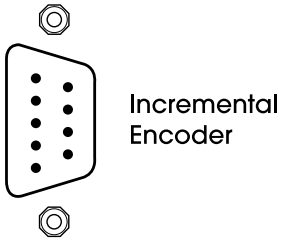


Figure 22: Encoder Connector, J8

The connectors used are ITT Cannon 9 pin D type; serial number: DE-121073-54, order code 105-533.

[2]Refer to the MNT Programming Manual for more details on the stop input. Items to refer to are: **#STOP, STOPMODE.**

The encoders are the position sensors used by the controller to measure axis position and consist of two pulse trains, 90 degrees out of phase. The controller uses the phase difference to determine direction of motion and counts the encoder edges to determine position. The frequency of these counted edges reflects the motor velocity.

The controller will work with three channel incremental encoders **(chA**, **chB** and **index**) and with both single ended TTL or differential line driver TTL output types. It is recommended that line driver outputs be used in all applications, since this gives increased noise immunity. It is important that each encoder cable is screened/shielded independently and that the screen/shield is connected at least at the *controller end*. Maximum cable length is dependent on the encoder specification, but should be kept as short as possible.

In many brushless servo drives, the motor is fitted with a resolver and the encoder signal is synthesized by the drive. In these instances, the encoder connections are wired to the drive rather than the motor.

When making connections to encoder outputs from a brushless drive, do not connect the +5V supplies on the controller and servo drive together since this may cause noise problems.

The encoder must be wired to a 9 pin 'D' male plug, using good quality multi-core/conductor screened/shielded cable, according to the diagram shown in Figure 23. If the encoder is a single-ended type (that is, no complimentary outputs) leave the **!chA**, **!chB** and **!index** pins unconnected. If the encoder does not have an index (Z) output, leave the **index** and **!index** unconnected.

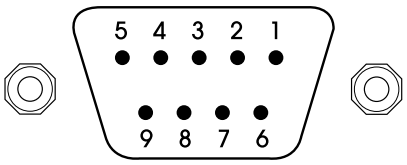


Figure 23: Encoder Pin-out

Pin No.	Signal Name	Function	Type
1	**+5V**	Power to Encoder	Output
2	**index**	Index Mark	Input
3	**!chB**	Channel B Compliment	Input
4	**shld / scrn**	Cable Screen/Shield	Input
5	**chA**	Channel A	Input
6	**!index**	Index Complement	Input
7	**gnd**	Signal Ground	
8	**chB**	Channel B	Input
9	**!chA**	Channel A Compliment	Input

8.1.7 Ground Connections

A good ground connection to the controller is essential for noise immunity in industrial environments. Bad ground connection can be the cause of problems, for instance, loss of motor position.

Connect the earth/chassis ground on *SmartMove* and the servo drives to a common ground point (or busbar) using heavy duty cable. Ensure that the connection is in a 'star' configuration.

8.1.8 Serial Cable

The RS232 or RS485 cable is used to connect the controller to a computer for programming and system commissioning. A computer is not essential for operation of the controller. A standard serial cable should be used or built according to the wiring diagram in Section 10: Hardware. An RS232 cable is available (order code CBL001-501).

Please note that the RS232 specification is a 'standard' that varies from manufacturer to manufacturer and therefore not all RS232 cables will work with the controller.

If an RS485 version of the controller is being used, please ensure you have a working RS485 to RS232 converter for your PC.

8.2 Testing System Wiring

In order to check that the system is correctly wired, it is recommended that testing is carried out on the bench and not within the machine as this may result in damage to the machine on which the motors are connected.

In order to verify that the system is wired up correctly, the following steps should be performed:

1. Check that error output (**enable**) is the correct polarity.

2. Check that the encoder and motor are connected correctly.

3. Check that the encoders are working.

4. Check that the servo drive is working in local mode if applicable.

5. Set-up system gains for satisfactory closed loop control.

Before checking the wiring, it is important to be familiar with the software tools. These are covered in the next section.

8.2.1 Communicating with *SmartMove*

Before starting, it is important to understand some of the basics in the communication with **MINT** on *SmartMove*. It is assumed that the reader has a working knowledge of Windows applications.

There must be a terminal emulator on the PC. This accepts characters from the PC's keyboard and transmits them over a port. In this case, the port will be a serial port (either **COM1** or **COM2** on the PC). The receiving device (*SmartMove*) will accept the characters and perform some action on them. It may respond with more characters which the terminal emulator receives and displays on the computer screen.

SmartMove will operate with almost any terminal emulator. A terminal emulator has been provided (called *cTERM for Windows*) which has been specially configured for use with *SmartMove*. Full details of *cTERM for Windows* and how to use it can be found in Section 13.

Before starting, *cTERM* must first be installed on the PC:

If running Windows 95:

1. Insert disk 1 into the floppy drive.

2. Select *Start* and then *Run*

3. Type **`A:\setup`** and follow the on screen instructions.

4. From the *MINT Programmer's Toolkit* group, select *cTERM*.

If running Windows 3.1:

1. Insert disk 1 into the floppy drive.

2. From the *Program Manager* Select *Run*

3. Type **`A:\setup`** and follow the on screen instructions.

4. From the *MINT Programmer's Toolkit* group, select *cTERM*.

When running *cTERM*, it must be configured for the correct controller type, serial port (**COM** port) and baud rate. Details of how this is achieved can be found in section 13.1. SmartMove is a member of the *EuroSystem* Product Family. The required baud rate is 9600.

Before powering up *SmartMove*, the serial cable from the PC serial port should be connected, (taking note of which **COM** port it was connected it to; **COM1** or **COM2**). The other end should be connected to the *SmartMove* 9 pin male connector.

From *cTERM* select the terminal icon:  . Applying power to *SmartMove* should display a sign on message, see figure 24.

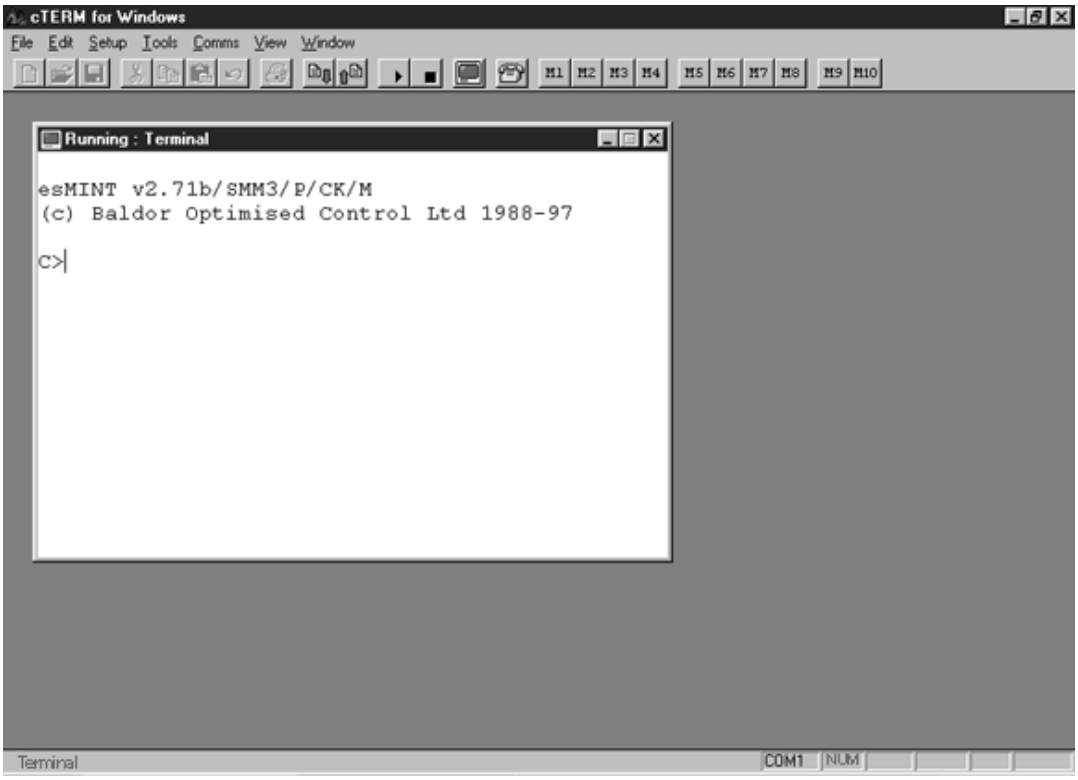


Figure 24: *SmartMove* Sign on Message

This shows the version number of MINT followed by either a **P>** or a **C>** prompt.

Pressing the Enter/Return ↵ key should produce another **P>** or **C>** prompt. This shows that there is communication with the MINT command line. The MINT command line is important in that it allows communication with MINT and the ability to issue commands.

By typing the following:

? “Hello” ↵

The ↵ symbol shows that the *Enter/Return* key must now be pressed.

Hello is displayed on the next line.

The command line provides immediate execution for the majority of MINT commands. Multiple lines of commands construct a program. A simple example would be:

```
FOR a = 1 TO 10
  ? “Hello”
NEXT
```

Rather than entering this at the command line as is, it must first be loaded into a file buffer and executed. MINT supports 2 buffers:

- *Configuration buffer*: This typically stores parameters for the drives and each axis. It is executed on power-up.
- *Program buffer*: This stores the application program and is executed after the Configuration buffer.

MINT supports a simple editor for editing *program* and *configuration files*. *cTERM* provides a more user friendly way to edit files and download them to the controller.

Program and *configuration files* are discussed in this section. The disk icon, 🖫, is used to identify example files which have been installed on the PC. After the disk icon, the file name is given. This file can be found in the installed *cTERM* directory, **\MINT\ESD\START**. These programs can be downloaded to the controller. Some files names have "**.MNT**" after the name which indicates that they are MINT *program files* and will be downloaded to the program storage space on the controller. Others, have "**.CFG**" after the filename, which indicates that they are *configuration files* and will be downloaded to the controller configuration storage space. Details of how to download these are given Section 13. More information on *configuration* and *program files* is given in the MINT Programming Manual.

Typing **RUN** (or pressing ■ on the toolbar) will compile and execute the configuration/ program. Pressing **[Ctrl]+[E]** (or the ■ icon) will terminate program execution. The following message will be returned:

```
Break at line XXX
```

It should be remembered that the **C>** or **P>** prompt symbol indicates commands that must be entered directly onto the command line.

All of the following examples assume the use of a two axis controller. Sample programs for 1 and 3 axis versions of the controller can be found in the directories **\MINT\ESD\START\1** and **\MINT\ESD\START\3** respectively.

8.2.2 Checking the Drive Enable

The drive enable relay allows SmartMove to shut down the drive in the event of an error. On power-up, the drive enable output will be disabled until a command such as RESET or CANCEL is encountered within the *program*. In order to check that the drive enable relay is correctly wired up, the drive should first be enabled with:

```
> RESET
```

The drive should now be enabled.

An error can be simulated and the drive disabled with:

```
> ABORT
```

The drive should now be disabled. If this is not the case, then the wiring and drive set-up should be checked.

8.2.3 Checking the Encoder

The encoder records the position of the motor in a positive and negative direction. The following program can be used to check that the encoder is functioning correctly (assuming a 2 axis servo system):

ENCODER.MNT

```
AXES[0,1]
RESET[0,1,2]
GN = 0; : REM Set all the gains to zero
KV = 0;
KI = 0;
KF = 0;
SF = 1;
ABORT   : REM Disable the servo drive
LOOP
  PRINT POS[0];POS[1]; : BOL
ENDL
```

The program should now be downloaded to the controller. The controller will indicate that the program has been downloaded successfully.

Type **RUN** at the **C>** or **P>** prompt.

Note that to stop the program running [Ctrl]+[E] must be typed at the terminal window

The program works by setting all system gains to zero and disabling the servo drives so that the motor shaft can be moved by hand if you are working with a torque amplifier. If the drive is configured in velocity mode, jogging it in local mode can be substituted for turning the motor shaft by hand. If the servo drive is not disabled, the connection of the enable output from the controller should be checked.

The program will print the position (encoder value) of axes 0 and 1 to the terminal. By manually turning the motor shaft clockwise and counterclockwise or by jogging motor with keypad, it is possible to see the position change. It should be noted that the controller uses quadrature decoding which gives 4 counts for each line of the encoder disk. With a 250 line encoder, the position should change by ±1000 for every revolution moved clockwise or counterclockwise. If the position does not change then the following should be checked:

- The axis 0 encoder cable is connected to encoder input 0 and axis 1 encoder cable is connected to encoder input 1.
- The encoders have power.
- The encoders are correctly wired up.

A note of which direction gives an increase in position should be made. The process should be repeated for other axes.

To check the index (marker) pulse, refer to section 8.4.

8.2.4 Checking Motor Polarity

To confirm the polarity of the motor connections, the **TORQUE** command should be used as shown in the following example. Refer to the MINT Programming Manual for further details on the **TORQUE** command:

```
> TQ.0 = 1
```

Starting with a value of **1**, the torque value should be increased until the motor starts to move. The motor should move in a positive direction, that is, the position (encoder) should increase. Download and **RUN** the following program to check this[3]:

MOTOR.MNT

```
RESET[0,1,2]
SERVOFF.1
TQ.0 = 1  : REM Increase until motor moves
LOOP
  ? POS.0; : BOL
ENDL
```

Tip: it is possible to use the controller's MINT editor to change line 3 of this program. First terminate any executing program by typing **[Ctrl]+[E]**. From the **P>** prompt type:

```
EDIT 3
```

After making changes to the line, press **return** to finish.

If the position decreases, the **demand/command** cables could be reversed. Ideally the **A** and $\bar{A}$ signals on the encoder should be reversed, having first removed power from the servo drive. The test should then be repeated.

[3]If a 1 or 3 axis controller is being commissioned, the program file can be found in the 1 and 3 sub-directory.

The operation can be repeated with a negative value of torque, for example:

```
TQ.0 = -1
```

The position should now decrease. Again increase the negative value until the motor starts rotating. This operation should be repeated for the other axes. The previous example will change to look like:

```
RESET[0,1,2]
SERVOFF.0
TQ.1 = 1  : REM Increase until motor moves
LOOP
  ? POS.1; : BOL
ENDL
```

If the motor does not move with a torque value of 100 (100% demand output) the following should be checked:

- The servo drive is enabled.
- The motor is connected.
- The servo drive is configured correctly.
- The controller is connected correctly to the servo drive
- Stop input is not active

More suggestions are given in Section 14.2 at the back of this manual.

8.3 Setting System Gains

At the lowest level of control software, instantaneous axis position demands produced by the controller software must be translated into motor demands. This is achieved by *closed loop control* of the motor. The motor is controlled to minimize the error between demand and actual position measured with an incremental encoder. Every 2ms (or optionally 1ms using the **LOOPTIME** keyword) the controller compares desired and actual positions and calculates the correct demand for the motor. The torque is calculated by a *Proportional, Integral, Velocity Feedback* and *Velocity Feed forward* (PIVF) algorithm.

It is possible that control could be achieved by applying a torque proportional to the error alone, but this is a simplistic approach. If it is imagined that there is a small error between demanded and actual position, a proportional controller will simply multiply the error by some constant (the *proportional gain*) and apply the result to the motor via the servo drive. If the gain is too high this may cause overshoot, which will result in the motor vibrating back and forth around the desired position. As the gain is increased, the controller will present more resistance to *positional error*, but oscillations will increase in magnitude until the system becomes completely unstable.

To reduce the instability a damping term is incorporated in the servo loop algorithm, called *Velocity Feedback*. This is analogous to putting the motor output shaft in syrup. Velocity feedback acts to resist rapid movement of the motor and hence allows the *proportional gain* to be set higher before vibration sets in. (In some applications, the velocity feedback is handled by the servo drive, called a *Velocity Servo*). The effect of too high proportional gain, or too low velocity feedback gain is illustrated by the '*Underdamped*' line in Figure 25:

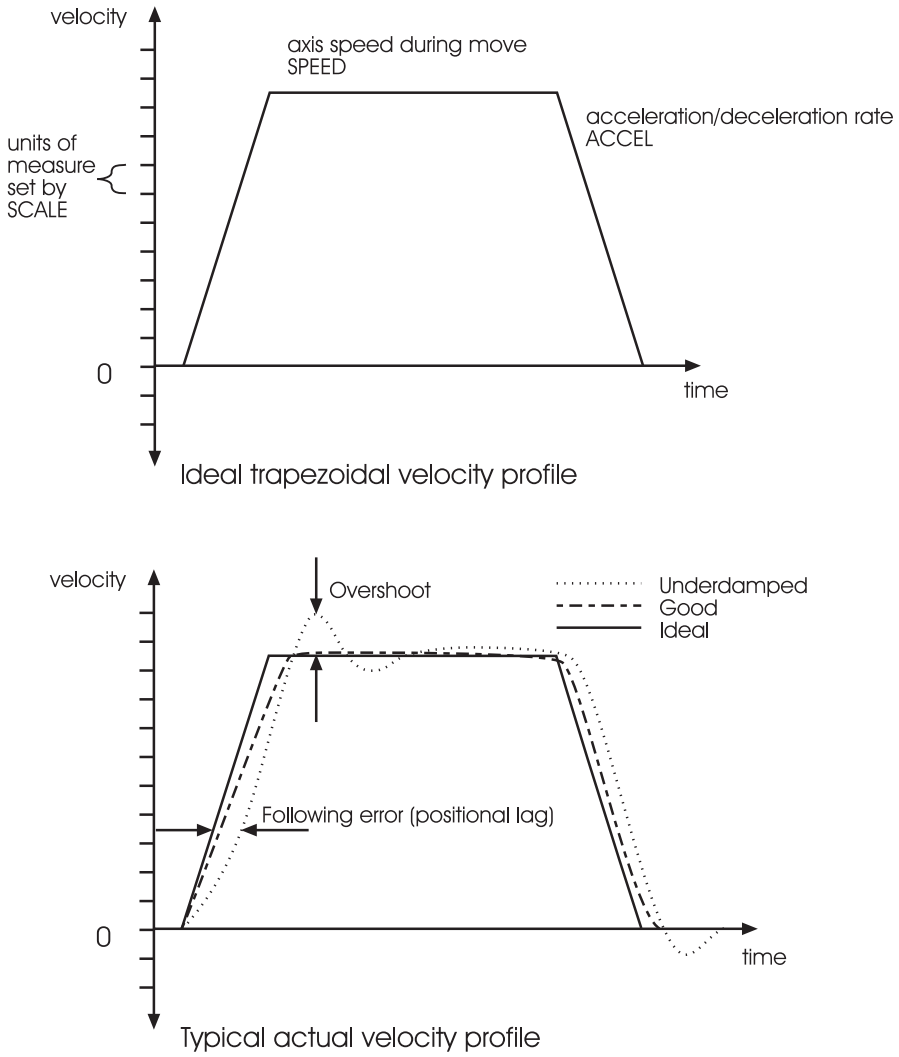


Figure 25: Servo response

When the motor is stationary at a set point there may be a small *positional error*. The controller multiplies the error by the *proportional term* to produce an applied corrective torque (in *current control*). But for very small errors, the torque may not be large enough to overcome static friction. Therefore integral action is also incorporated in the loop calculations, this involves summing the error over time so that the torque may be gradually increased until the *positional error* falls to zero. The rate at which integral action works is controlled by the Integral Gain. Integral action is useful to eliminate steady state *positional errors*, but will result in reduced dynamic response for the system.

The final term in the control loop is *Velocity Feed Forward*. This is useful for increasing the response and reducing the *following error*.

Two types of servo drives may be used with the controller:

- *Current or torque servo drives:* These use the **demand/command** signal to control the current flowing in the motor armature and hence the torque of the motor.
- *Velocity controlled servo drives (velocity servo):* These use the **demand/command** signal as a servo speed reference.

For general purpose applications, the *torque servo* drive is cheaper and simpler to set-up, but the *velocity servo* gives better control, especially in high performance applications. For *torque servo* drives, *velocity feedback* must be used to stabilize the system. However, this is not normally required for a *velocity servo* since it incorporates its own internal *velocity feedback*.

A block diagram of the complete control loop, showing controller, drive, motor and gearbox is shown in Figure 26. The *servo drive* may be a simple current drive, or may incorporate internal *velocity feedback* via a tachometer:

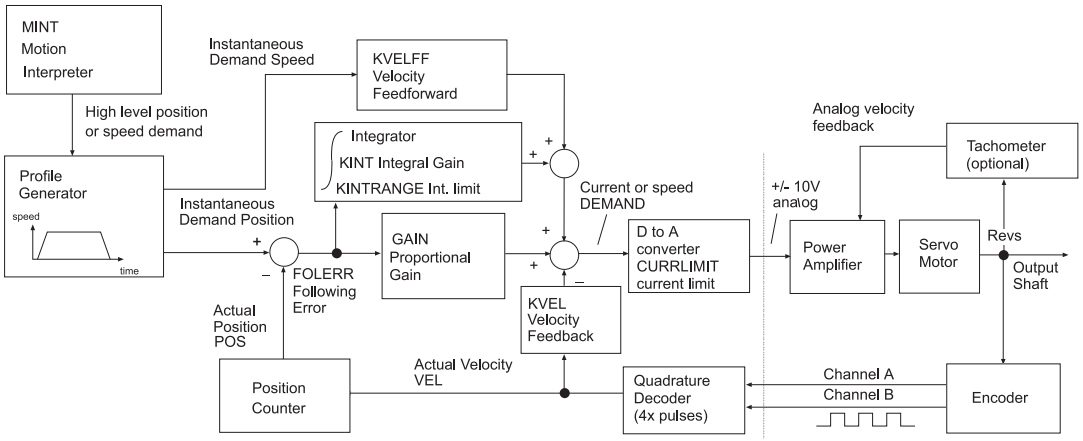


Figure 26: Servo loop block diagram showing relevant MINT keywords in capitals

It can be observed that there is a four term controller incorporating *proportional, velocity feedback/feed forward* and *integral* gains.

The equation of the loop closure algorithm is as follows:

```
Demand = GN.e - KV.v + KF.V + KI.e
```

Where:

e = *following error* (quad counts)

v = *actual axis velocity* (quad counts/sample time)

V = *demand axis velocity* (quad counts/sample time)

These equate to the following MINT keywords:

Keyword	Abbreviation	Description
GAIN	GN	Proportional servo loop gain
KVEL	KV	Velocity feedback gain
KVELFF	KF	Velocity feedforward gain
KINT	KI	Integral feedback

Tuning the drive involves changing the four servo loop gains, **`GN, KI, KV`** and **`KF`** to provide the best performance for the particular motor/encoder combination and load inertia. In view of the diversity of application, these values all default to zero and should be set-up in the *Configuration buffer*.

Two other keywords, **KINTRANGE** and **CURRLIMIT**, are used to control the **demand/command** output. **KINTRANGE**, the integration limit, determines the maximum value of the effect of integral action, KI.Σe. **KINTRANGE** is specified as a percentage (%) of the full scale **demand/command** output in the range of ±10V. Therefore if **KINTRANGE** = 25, the maximum effect of integral action is ±2.5V.

CURRLIMIT, the current limit, so called for its use with current drives, determines the maximum value of the **demand/command** output as a percentage of the full scale demand. Therefore if **CURRLIMIT** = 50, the maximum **demand/command** output will be ±5V.

The encoder gain (measured in pulses/rev) is one factor that is hardware dependent but has a direct effect on the overall loop gain. The other parameters are software controlled in the range of 0.0-255.0, the resolution of the decimal part is one part in 256 as with normal MINT variables. Refer to the MINT Programming Manual for an explanation of scaled integers.

Since all servo loop parameters default to zero, the motor will have no power applied to it on power-up. Most servo drives can be set-up in either *Current* (torque) *Control* mode or *Velocity Control* mode. The procedure for setting system gains differs slightly for each.

8.3.1 Setting System Gains for Current Control

Having confirmed that the encoder and motor are correctly wired up, some feedback gain KV should be applied. Starting with a value of 1 it should be increased until some resistance in the motor is felt.

For example:

```
> KV = 1
```

For some motors, it may be necessary to apply fractional gains (**KV = 0.5**, for example).

Once the feedback gain has been set, some proportional gain, **GN**, should be applied. Start off with a value which is a quarter of the feedback gain, that is **GN = KV/4**. If the motor starts to vibrate, the velocity feedback gain (damping), **KV**, should be increased or the proportional gain, **GN** decreased. The proportional gain, **GN**, should be increased until the motor shaft becomes stiff.

Finally, the velocity feed forward gain, **KF**, should be set to the same value as the velocity feedback gain, **KV**. This has the effect of reducing the position lag (following error) during motion.

```
> KF = KV
```

It should now be possible to move the motor under closed loop control. The JOG command can be used to move the motor at constant velocity. For example:

```
> JOG = 1000
```

will jog the motor at 1 revolution per second in a positive direction (assuming a 250 line encoder).

```
> JOG = -1000
```

will jog the motor at 1 revolution per second in a negative direction.

By using the statement:

```
> PRINT FE
```

the *following error* (position error) can be read. By continuously reading the *following error*, the value should be constant to within a few counts. If the value increases, it may be necessary to increase the gains.

By experimentation and increasing the **JOG** speed to 5000, 10000, 15000 and 20000 and the following error (**FE**), can be checked. The program below can be found in the directory **\MINT\ESD\START** and may be used to check the *following error* and *demand/command* output from the controller. It should be noted that the program assumes a 250 line encoder. It may therefore be necessary to modify the **JOG** speed to suit the system being used:

FOLERR.MNT

```
JOG.0 = 10000 : REM 10 revs/sec for 250 line encoder
LOOP
  PRINT FOLERR.0;DEMAND.0 : REM Print the following error
  WAIT = 100              : REM Wait 100 milliseconds
ENDL
```

This can also be used to determine the maximum speed of the motor without a load. An error will be generated if the motor cannot keep pace because the jog speed is too high. This is represented by an F on the 7 segment status display. It may be necessary to increase or decrease the proportional gain, **GN**, or damping, **KV**, to achieve satisfactory following errors.

An alternative method of finding the maximum speed of the motor is to use **TORQUE** control and read back the velocity, the torque output can be increased until a constant maximum velocity is obtained. For example:

```
> TQ = 20
> ? VEL
```

The maximum speed may be reduced once the motor shaft is loaded.

If the motor is running at a continuous speed for any length of time, it must be made certain that the servo drive output is within its continuous rating to avoid over current. For many servo drives, this is 50% of the peak output, therefore a controller *demand/command* of 50 will keep the servo drive within its continuous rating. Refer to the **DEMAND** keyword in the MINT Programming Manual.

The program can be changed to test the other axes in the system. The same rules should be followed for setting the servo gains, but applied to the other axis. For example:

```
> KV.1 = 1
```

sets the velocity *feedback gain* on axis 1.

8.3.2 Fine Tuning System Gains

The above procedure for setting system gains, although adequate to get the system moving, will not provide the optimum response without further fine tuning of the system gains. The aim is to set the *proportional gain* as high as possible without getting overshoot, instability or hunting (buzzing) on an encoder edge when stationary.

This is best achieved by attempting some short positional moves (perhaps one revolution of the motor) with high accelerations and speeds and observing the response on an oscilloscope. The oscilloscope is normally connected to a tachometer on the motor or an output on the drive. However, the controller has the capability to output the instantaneous velocity from a +/-10V tuning output. Refer to the **AUX** keyword in the MINT Programming Manual for further details.

If an oscilloscope is not available, *cTERM* for Windows allows a move to be set-up and plotted in order to assist in servo tuning.

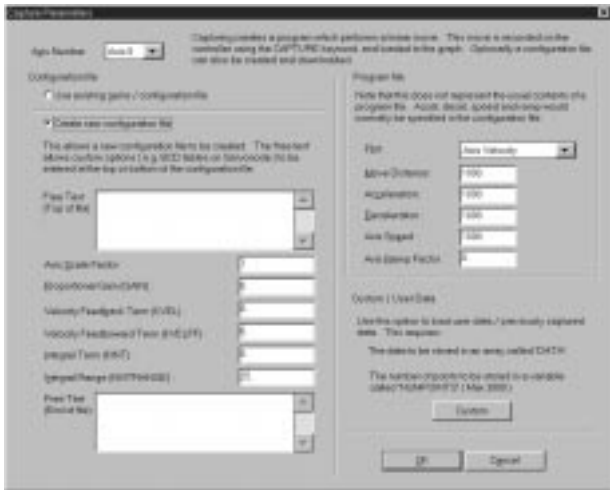


Figure 27: Capture facility in cTERM for motor tuning

If an oscilloscope is not available, the following program can be found in the directory **\MINT\ESD\START** and may be used to print the axis speed and following error to the terminal screen during a position (relative move). Note that the program assumes a 1000 line encoder and should be modified if the system used differs from this:

MOVE1.MNT

```
REM Program buffer for trial moves

RESET[0,1,2]
AXES[0]          REM Only axis 0 used
SCALE = 4000     REM set this to 4x the encoder line count
SPEED = 50       REM 50 revolutions per second
ACCEL = 10000    REM 1000 rev/sec^2

REM move relative four revolutions
MOVER = 20
GO

REM print following error and speed to screen during move
REPEAT
  PRINT FE; SP;
UNTIL IDLE
END
```

When this program is run, it will print the *following error* and *speed* of the axis in quick succession on the screen. Generally, the *following error* will be greatest during acceleration. The system, if well behaved, should have a relatively low error during the constant speed part of the move.

Altering the gains, the acceleration and speed should be tried in order to identify the maximum achievable acceleration for the system and the maximum speed.

Once the motors are connected to the system (or loaded), it may be necessary to increase the gains slightly and to reduce the speeds.

The program can simply be changed to test the other axes, by altering the **AXES[0]** command to **AXES[1]** and **AXES[2]**.

8.3.3 Eliminating Steady-State Errors

In systems where precise positioning accuracy is required, it is often necessary to position to within one encoder count. *Proportional gain*, **GN**, is not normally able to achieve this because a very small following error will only produce a small demand for the servo drive. This may not be sufficient to overcome mechanical friction. This is particularly so for current controlled systems. This error can be overcome by applying some *integral gain*.

The integral gain, **KI**, works by accumulating *following error* over time to produce a demand sufficient to move the motor into the *zero following error* position.

Particular care is required when setting **KI** since a high value can cause instability during moves. The effect of **KI** should be limited by setting the maximum range of the integration (using the **KR** keyword) to the minimum value sufficient to overcome friction or static loads. Typical values are:

```
KR = 25
KI = 0.1
```

where **KR** limits the integral term to 25% of the full DAC output range. **KI** is usually a factor of 10 less than proportional gain, **GN**. In most systems, it is better to avoid using integral action by choosing an encoder which has a line count significantly better than the highest positioning accuracy required.

8.3.4 System Gains for Velocity Drives

Velocity controlled drives incorporate the *velocity feedback* term in the servo drive and therefore it is usually sufficient to have **KV = 0** on the controller.

Usually, the value of the *proportional gain*, **GN**, will be less than with an equivalent current controlled system. Often a fractional value of *proportional gain* gives the best response.

For example:

```
> GN = 0.25
```

Correct setting of the *velocity feed forward gain*, **KF**, is important to get maximum response from the system. This is best performed using a storage oscilloscope to record the tachometer output for fast point-to-point moves. This enables the velocity/time profile for the motor to be observed in order to monitor actual acceleration and overshoot.

If an oscilloscope is not available *cTERM* for Windows allows a move to be set-up and plotted to assist in servo tuning.

Referring to the servo loop block diagram in section 8.3 figure 26, the *velocity feed forward* term is a block which takes the instantaneous speed demand from the profile generator and adds this to the output block. Because **KF** is a feed forward term, an important difference between this and the other terms exist. **KF** is outside the closed loop and therefore does not have an effect on system stability. This means that the term can be increased to maximum without causing the motor to oscillate; provided that the other terms are set-up correctly.

In practice however, a very high value of **KF** is of no benefit to system performance. Instead, it must be set such that a demand/command of X rpm from the profile generator results in a **demand/command** output to the velocity drive which gives X rpm on the motor shaft (a 1:1 relationship).

When set-up correctly, **KF** will cause the motor to move at the demand speed from the profile generator. This is true without the **PID** terms (**GAIN, KVEL, KINT**) in the closed loop providing anything except compensation for small errors in the position of the motor due to analog drift. This gives faster response to changes in demand speed, with lower following errors.

Example calculation of **KVELFF**:

In order to calculate the correct value for **KF**, the workings of the servo loop closure algorithms must be considered. In the servo loop, speeds are expressed in quadrature counts/servo loop closure time. For instance, a speed of 100 is 100 counts every 2ms in the standard controller.

It is possible to set the controller to a 1ms sample time using the **LOOPTIME** keyword. Refer to the MINT Programming Manual for further information.

In the following example, the velocity of the servo is 3000rpm with a +10V input, and the encoder has 1024 counts per revolution.

At 3000rpm an analog voltage of +10V is required. This relates to:

$$\frac{3000}{60} = 50 \text{ revs per second}$$

The number of quadrature counts per loop closure time can be calculated by using the expression:

$$\frac{\text{speed\_in\_revs} * \text{encoder\_line\_count} * 4}{\text{number\_of\_loop\_closures\_per\_second}}$$

The factor of 4 is included since the controller counts every edge of the pulse train coming from the encoder **A** and **B** channels, which gives four times better resolution than the number of lines.

$$\frac{50 * 1024 * 4}{500} = 409.6 \text{ quadrature counts per servo loop closure time}$$

(Note: for 1ms loop closure time, this expression becomes: 50*1024*4/1000).

The DAC output has a resolution of 12 bits over the range -10V to +10V, therefore +10V = 2048 counts. The feed forward term is therefore given by:

$$KVELFF = \frac{2048}{409.6} = 5.0$$

Increasing **KF** above the calculated value will cause the controller to have a *following error* ahead of the desired position. Decreasing **KF** below this value will cause the controller to have a more normal *following error* behind the desired position. The calculated value above should give zero *following error* in normal running.

The effect of *velocity feed forward gain* can be investigated by jogging the motor at constant speed and printing out the *following error*, **FE**, for different values of **KF**. When attempting this, ensure that there is zero *integral gain*, since this will cause the *following error* to tend to zero under steady state conditions, thereby negating the effect of changes in **KF**.

Once all of the servo parameters have been determined, they can be entered into the MINT *Configuration file*. An example and explanation of a MINT *Configuration file* can be found in Section 9.1.

8.4 Encoder Marker Pulse

When the servo drives are configured with the correct gains, the marker pulse on the encoder can now be checked. This is achieved by using the MINT HOME command as follows:

```
> HOME = 6
```

This will seek the index pulse in a positive direction. If the motor does not stop after more than 1 revolution, the wiring on the encoder should be checked.

Using the command:

```
> HOME = 4
```

The motor should rotate in a negative direction and stop at the marker pulse. The process should be repeated for other axes.

9. Introduction to MINT™ Programming Language

MINT is the programming language used to program the controller in order to meet the requirements of specific applications. MINT provides control of the I/O and motion control aspects of the controller using Basic-like programming structures and keywords. There may be an application where a thumbwheel switch (fed into the digital inputs of the controller) sets a distance to move and a potentiometer (fed into one analog input on the controller), changes the slew speed. A simple MINT program can be written to achieve this.

MINT programs consist of two buffers. The *Configuration buffer* stores information relating to the machine set-up, for instance the servo loop gains. The *Program buffer* stores the actual motion control program. In fact the two buffers are the same and can contain the same instructions, except that the *Configuration buffer* is only 1K maximum size, while the *Program buffer* can be up to 26K. Additionally, there is a third method of storing information in the controller - in the form of array data - which can be used as variables within a program. More information is provided in the MINT Programming Manual.

Note that a file is data held on the computer's storage device (floppy or hard disk) which is downloaded to the controller. A buffer is the area in the controller's memory that holds a duplicate of the file.

9.1 The Configuration File

The *Configuration file* is used to store appropriate defaults for a particular system. Once gains and speeds have been found, these should be incorporated into the *Configuration file* for the system.

Whenever the program is **RUN**, the *Configuration buffer* is executed first. The following provides a list of parameters necessary for set-up:

- **Servo loop gains** – to tune the system response.
- **Scale factor** – to set the units of measure of the application. Refer to the MINT Programming Manual for details of using SCALE.
- **Maximum following error** (**MFOLERR**) – to set a safe maximum difference between actual and desired positions.
- **Default speeds (SPEED), accelerations (ACCEL) and decelerations (DECEL) for the system** – to determine the shape of the trapezoidal velocity profile **(RAMP)**.

These parameters are generally to be set-up only once for an application, but can at any time be altered in the Program.

A typical *Configuration file* for a three axis system is:

CONFIG.CFG

```
AUTO : REM automatic execution on power-up

REM Config file for XYZ system

RESET[0,1,2] : REM to ensure previous setting cleared
AXES[0,1,2]: : REM 3 axis servo system

REM system gains
GAIN   = 10;
KVEL   = 40;
KVELFF = KVEL; REM assuming a torque servo drive
KINT   = 0;

REM position/SPEED parameters
SCALE = 2000; : REM units revs (500 line encoder with 4x multiplication)
SPEED = 60;   : REM max SPEED 60 revs/sec
ACCEL = 150;  : REM max accel 150 revs/sec^2
RAMP = 0;     : REM Trapezoidal motion
MFOLERR = 1;  : REM 1 rev maximum following error
```

In the above example, the default axis list is **0,1,2**. The semi-colon is used to apply all the parameters to all three axes. Refer to the MINT Programming Manual for further details on program language syntax.

Please note that the values for gains, speeds etc. are given only as an example. It is up to the user to determine the best values for a system. Failure to do so may result in damage to the machine.

9.2 The First MINT Program

This first MINT program example consists of a simple Configuration and Program which is used to index a motor by a set distance entered via a computer, or via the operator keypad. The Configuration and Program files can be found in the **\MINT\ESD\START** directory.

The Program file is called **FIRST.MNT** and the Configuration file is called **FIRST.CFG**. The example has been written for a single axis of motion, connected to axis 0 of the controller.

FIRST.CFG

```
REM File name: first.cfg
REM Configuration file for first MINT program

AXES[0]  REM This program only uses axis 0 (the first axis)
RESET[0,1,2]

SCALE = 2000 REM Scale to revs  Assume 500 line encoder (500*4)
GAIN = 1     REM Servo gains - these should be changed to the values..
KVEL = 5     REM .. found during servo set-up
KINT = 0
KF = KVEL

SPEED = 50   REM Default speed during positional moves 50 rev/s
ACCEL = 200  REM 200 rev/s^2

END
```

Any characters after a **REM** (remark) statement are ignored, which allows comments to be inserted into the program to make it more intuitive. It is important that back-up copies of *Program* and *Configuration files* are kept a on disk. The first line of this program indicates the name given to the disk file.

It must be remembered that the gains and scale factor may have to be changed to suit a particular motor.

Tip: When downloading a *Configuration file* from *cTERM* the message "Cannot Download" may be printed on the computer screen. A common reason for this is that there is already a program running on the controller. This can be checked (and the program aborted) by going into the terminal screen and pressing the **E** key while holding down the **CTRL** key.

FIRST.MNT

```
REM File name: first.mnt
REM Program to demonstrate use of the keypad to cause
REM the motor to repeat an index distance ten times

REM initialise variables used in program
index_length = 0

REM enable keyboard with default layout
KEYS ""

LOOP
 CLS REM clear screen

 REM Print up screen requesting user to enter move distance
 LINE 1,"Enter index distance"

 REM input statement uses formatted input xx.x
 LOCATE 5,2 REM put cursor at column five line 2
 INPUT index_length USING 2,1

 REM Perform move ten times printing cycle number on screen

 LINE 1,"Indexing"
 FOR cycle = 1 TO 10
  MOVER = index_length  REM move relative command
  GO                    REM start motion
  LINE 3,"Cycle no: ",cycle;
 NEXT

ENDL REM go back to start of loop for next motion
```

The *Program file*, **FIRST.MNT** should be downloaded to the controller and **RUN** typed at the **P>** prompt in the terminal screen.

Tip: If during program execution the controller aborts and prints the error message:

```
ERROR: Program: Following error at line xx on axis 0
```

this probably signifies that the motor is not correctly set-up. It should be verified that the configuration is in accordance with the instructions in the Setting-Up or Troubleshooting sections of this manual.

9.2.1 Program Narrative

The first MINT statement in the file (which is not a comment) is the line:

```
index_length = 0
```

this defines a variable called **index_length** and initializes it to the value 0. **index_length** is used later in the program to store the length of move entered by the operator.

This example incorporates the use of an *Keypad*Node which functions in the same manner as a standard serial terminal. Pressing any of the keys on the panel causes a character to be placed in the serial port buffer. It can then be read by the program by using the **INPUT, INKEY** keywords etc. Similarly, the keyword **PRINT** can be used to output data to the LCD screen (20 character x 4 line) on the operator keypad.

The **LOOP** statement in the program signifies the start of a loop from which the program never exits. It simply marks the point to which the program jumps when it encounters the **ENDL** statement.

After clearing the terminal screen of any erroneous information printed by previous programs (CLS) the statement:

```
LINE 1,"Enter Index Distance"
```

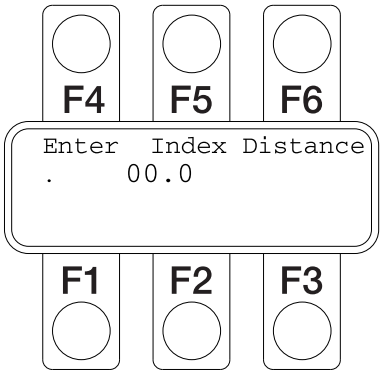
prints a message on line one of the terminal. The **LINE** keyword was written to ease printing information on the LCD display. Note that it is equally possible to use the **LOCATE** and **PRINT** statements to do this:

```
LOCATE 1,1
PRINT "Enter index distance"
```

The **LINE** keyword however,clears all characters to the end of the line that may be left on that line from previous **PRINT** statements, even if the text string printed is less than 20 characters - the width of the screen.

The next statement requests an input from the operator, being the number of revolutions of the motor wanted to index. The **USING** statement is used here to print the number in a set format. In this case two integer characters followed by a single decimal character. (Note that the **SCALE** keyword in the *Configuration file* has been set-up so that all distances and speeds are in revolutions of the motor.) On power-up the operator keypad should look as follows:

The desired length can be entered by typing in the number of revolutions at the keypad. Pressing return will cause the program to move the motor the set distance ten times.



Tip: An audible 'Beep' at the keypad whenever a key is pressed, gives the operator feedback that the key has been entered correctly.

The motor movement is achieved by the statement:

```
MOVER = index_length
GO
```

MOVER is a relative positional move. It causes the motor to move the number of revolutions specified, by using the accelerations and speed set-up previously in the *Configuration file*. The **GO** command is required to actually start the motion. This is beneficial if synchronizing both relative and absolute moves on two or more motors when necessary. **GO** is not required for certain types of moves, notably continuous speed control using the **JOG** command.

It can be seen that the **MOVER** statement is surrounded by a **FOR .. NEXT** loop statement. This causes the statements inside the loop to be executed 10 times with the number being stored in the variable cycle each time. This is printed to *Keypad*Node each time the loop is repeated.

Finally, the program arrives at the **ENDL** statement. This causes it to jump back to the **LOOP** statement so that the next index length can be entered. **FOR .. NEXT** and **LOOP .. ENDL** are two examples of the four different loop statements. These statements are useful for writing machine control programs.

To end program execution, the keys **[Ctrl]+[E]** (**Ctrl** and **E** together).

9.3 A Simple Cut to Length Feeder

In many applications, MINT can be used to program the system as a stand-alone machine controller. This program illustrates such a program for a simple cut-to-length machine. The program allows the operator to enter a product length, feed speed and number of feed cycles. It will then record the total length of product that has been cut.

If the controller has been set-up according to the instructions in the first part of this manual and the standard controller with keypad is being used, then it should be possible to get this program operating within a few minutes, by using one or two motors with their shafts in free air.

The *Program file* **FEEDER.MNT** and the *Configuration file* **FEEDER.CFG** will have been installed in the appropriate *cTERM* directory. *cTERM* should be started and these files downloaded to the controller as before. **RUN** should be typed at the **P>** prompt.

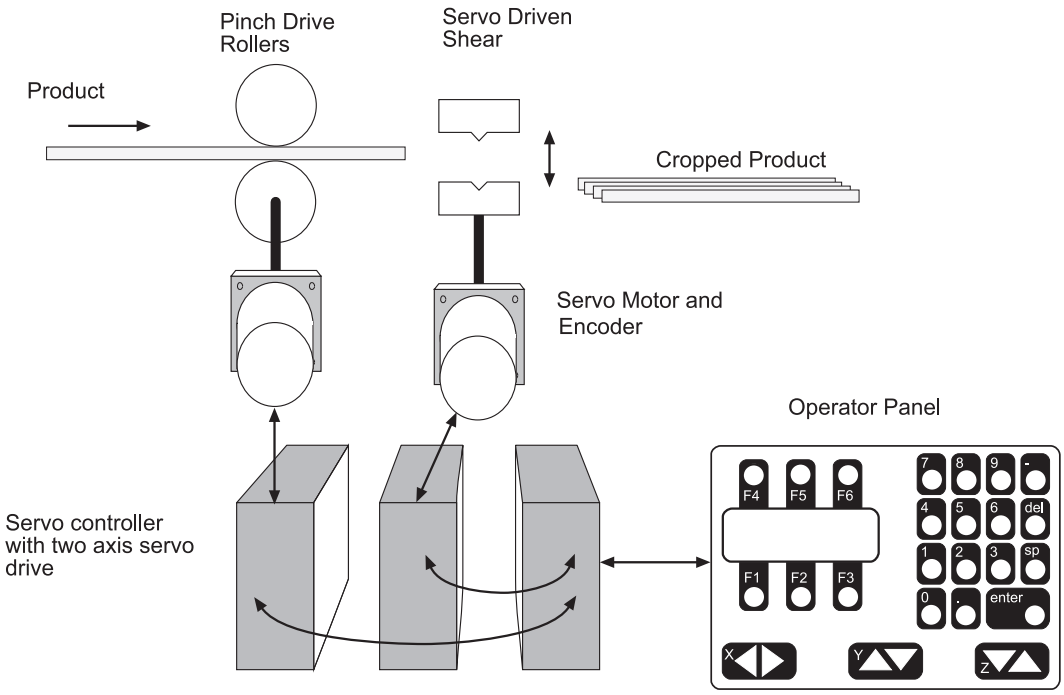


Figure 28: Schematic Diagram of the Cut to Length Feeder

9.3.1 Configuration File FEEDER.CFG

```
AUTO REM Automatic program execution on power-up

REM File name: feeder.cfg
REM Configuration buffer for simple cut to length machine

CLS  REM clear screen
PRINT "Please wait..."

AXES[0,1]
RESET[0,1,2]

SCALE = 50;    REM 50 encoder counts = 1 mm on this application
GAIN = 10;     REM Servo loop gains
KVEL = 40;
KF = KVEL;
KINT = 0;
SPEED = 4000;  REM 4000 mm/s
ACCEL = 40000; REM 40000 mm/s^2
RAMP = 0;      REM no 's' ramping
PAUSE NODELIVE.14  REM wait for keypad communications to be established

END
```

The *Configuration file* contains information specific to the servo system set-up. Detailed information of the MINT command syntax is given in the MINT Programming Manual.

The beginning of the file contains the statement **AUTO**. This command signifies that the *Configuration buffer*, then the *Program buffer* should be run automatically on power-up. If automatic execution is required, **AUTO** must always be placed at the very start of the *Configuration buffer*. When turning the controller off, the program will be retained in non-volatile memory and on power-up the message:

```
"Please wait ..."
```

will be displayed on the screen (printed on line 3) during which time the controller compiles and executes the Program buffer. **AUTO** must always be on the first line of the *Configuration buffer*.

The statement **AXES[0,1]**, indicates that the controller is configured for two axes of motion and that all commands thereafter will relate to these two axes unless explicitly indicated by enclosing the axis number in brackets. For example, the command:

```
SPEED[1] = 10
```

sets the speed of axis 1 to 10, but

```
SPEED = 10;
```

sets the speed of both **0** and **1** to **10**. Note that the use of the semi-colon to set both axes to **10**, otherwise it would be necessary to type **SPEED = 10,10**.

Also note the inclusion of the **RESET** command. Although not strictly necessary, this guarantees that the controller starts in a known state with all error flags reset to their default values and position set to zero.

The code **SCALE = 50**; sets the system units in relation to the number of encoder quadrature counts. If there were 50 counts per mm of linear movement, and it was desired to program speeds and distances in mm, **SCALE = 50** would be set .

The remainder of the file contains system gains and configuration information.

These values may have to be changed to achieve a stable system according to the particular motor/drive being using - refer to section 8.3 of this guide for further information.

9.3.2 Program FEEDER.MNT

```
REM Program to demonstrate use of the keypad in a cut to length machine

RESET[0,1,2] REM reset all motion parameters to default values
GOSUB initialize REM call initialize subroutine
GOSUB main_loop REM call main subroutine
END

#non_volatile
REM dummy definitions of variables stored in non-volatile RAM
REM this routine is not actually called and therefore the variables
REM are defined but not initialized to zero so that their programmed
REM values are retained
 cycles = 0 REM number of material feed cycles
 slew_speed = 0 REM speed of material feed
 length = 0 REM amount of material feed
RETURN

#initialize
 REM This subroutine sets up various parameters when program starts
  KEYS ""  REM enable keyboard with default layout
  BEEPOFF  REM turn off automatic keyboard beep
  SPEED = slew_speed REM restore speed stored in non-volatile memory
  count = 0           REM total length of material fed
  jog_sp = 2000       REM default jog speed in manual mode
RETURN
```

```
#main_loop
REM This subroutine is the main program loop, it prints the start-up
REM screen and handles operator selections by calling further subroutines
 LOOP
  REM Print up menu screen on four line operator display
  REM Line 4 is the 'soft keys' for operator selection
  LINE 1,""
  LINE 2,"XYZ Widget Company"
  LINE 3,"Press Feed Control"
  LINE 4,"START  SETUP MANUAL",

  REM Read the key pressed into the variable 'key' and test to see if
  REM the function keys (which return A B and C) are pressed
  key=INKEY
  IF key = 'a' THEN BEEP:GOSUB start REM call start subroutine
  IF key = 'b' THEN BEEP:GOSUB setup
  IF key = 'c' THEN BEEP:GOSUB manual
 ENDL REM go back to top of loop
RETURN

REM This subroutine is manual mode allowing movement of roll back & forth
#manual
  REM Print new menu where soft keys are FAST motion, SLOW motion and
  REM EXIT to main menu
  LINE 1,""
  LINE 2,"Manual mode - press"
  LINE 3,"'X' key to Jog roll"
  LINE 4,"FAST   SLOW    EXIT",
  jog_sp = 2000
  LOOP
   REM read softkey presses and set fast or slow motion or exit
   key=INKEY
   IF key = 'a' THEN BEEP : jog_sp = 2000 REM fast motion
   IF key = 'b' THEN BEEP : jog_sp = 100  REM slow motion
   IF key = 'c' THEN BEEP : EXIT          REM program jumps to ENDL if true

   REM This moves the motor back and forth using the arrow keys < and >
   REM marked 'X'. READKEY is used to return the value of a key that
   REM is pressed and held < returns the character 'x' and > returns 'u'
   IF READKEY = 'u' DO
    JOG = -jog_sp
   ELSE IF READKEY = 'x' DO
    JOG = jog_sp
   ELSE
    STOP   REM button released - stop motor
   ENDIF
   ENDIF
  ENDL
RETURN
```

```
REM Subroutine to allow operator to set-up product length, feed speed and
REM number of repetitions
#setup
 REPEAT
  LINE 1,"                EXIT"
  LINE 2,"Setup menu"
  LINE 3,"Select function"
  LINE 4,"LENGTH SPEED CYCLES",
  LOCATE 1,3
  key=INKEY
  IF key = 'a' THEN BEEP : GOSUB get_len
  IF key = 'b' THEN BEEP : GOSUB get_sp
  IF key = 'c' THEN BEEP : GOSUB get_cy
 UNTIL key = 'f' REM end of REPEAT..UNTIL loop, terminates if 'f' pressed
 BEEP : CLS      REM beep and clear screen
RETURN

REM subroutine to get length of material
#get_len
 CLS
 LINE 1,"Enter index"
 LINE 2,"distance:"
 LINE 3,"              mm"
 BEEPON REM automatic keyboard beep on
 REM formatted input XXX.X
 LOCATE 9,3 : INPUT length USING 3,1
 BEEPOFF
RETURN

REM get number of feed cycles, up to 99 repetitions
#get_cy
 CLS
 LINE 1,"Enter number of"
 LINE 2,"indexes required:"
 BEEPON
 LOCATE 9,3 : INPUT cycles USING 3
 BEEPOFF
RETURN

REM subroutine to get speed, this example validates entered number to
REM make sure that it is less than 4000 and greater than 100
#get_sp
 REPEAT
  CLS
  LINE 1,"Enter maximum"
  LINE 2,"slew speed:"
  LINE 3,"              mm/s"
  LINE 4,"Range 100-4000mm/s",
```

```
  number = slew_speed REM store before validating
  BEEPON
  LOCATE 9,3 : INPUT number USING 4
  BEEPOFF
 UNTIL number >= 100 AND number <= 4000
 slew_speed = number REM if valid update variable and SPEED command
 SPEED = slew_speed
RETURN

REM Run automatic cycle
#start
 LOCATE 1,1
 LINE 1,"               STOP"
 LINE 2,"Machine Running"
 LINE 3,"Cycle no:"
 LINE 4,"Material:",
 REM FOR .. NEXT loop executes n times where n = cycles
 FOR index = 1 TO cycles
  REM index material - move axis 0 distance given by length
  MOVER[0] = length : GO[0]
  PAUSE IDLE  REM wait for previous move to finish
  REM perform punch operation by moving axis 1 up and down
  MOVEA[1] = 10 : GO[1]
  MOVEA[1] = 0 : GO[1]
  BEEP
  count = count + length REM accumulate material fed
  REM print status information
  LOCATE 11,3 : PRINT index USING 2;
  LOCATE 11,4 : PRINT count USING 5,1;
  IF INKEY = 'f' THEN STOP : EXIT
 NEXT  REM end of FOR .. NEXT loop
 REM End of programmed number of cycles
 LINE 1,"               EXIT"
 LINE 2,"Machine Stopped"
 PAUSE INKEY = 'f'
 BEEP
RETURN

#ONERROR
IF ERR = 70 THEN status = CANSTATUS
REM Error handling subroutine called by system in the event of excessive
REM following error (machine jam) or limit switch error
 LINE 1,"****** Error ******"
 LINE 2,"*** Machine Jam ***"
 LINE 3,"*** Press Reset ***"
 LINE 4,"RESET",
 REPEAT
  BEEP  REM sound alarm buzzer continuously
 UNTIL INKEY = 'a'
```

```
 CANCEL[0,1,2]  REM cancel error and re-run program
 RUN
RETURN

#stop
REM Error handling routine called by system when STOP input (guard switch)
REM is asserted.  Subroutine prints message on operator screen, then
REM returns to main program
 LINE 2, "GUARD OPEN"
 PAUSE STOP = 0 REM wait for guard to be closed
 LINE 2, "Machine running"
RETURN
```

9.3.3 Cut To Length Program Narrative

The program has been written in a structured method for ease of maintenance, according to the following prototype:

```
REM program starts here
GOSUB init
GOSUB main
#init
...
RETURN

#main
...
RETURN
```

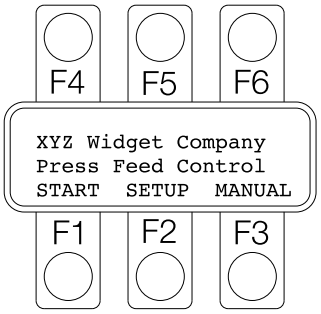
The program is well commented and by making suitable adjustments to the **SCALE** factor and gains in the *Configuration file* it should be possible for the program to operate on any system equipped with a keypad and at least one axis of motion.

The **#non_volatile** subroutine is defined but never called. This is used to define some values which are used in the program to store the product length entered by the user, for example. Because the subroutine is never called they are never actually set to zero on power-up and therefore the last values entered by the user are retained.

The initialization sub-routine, **#init** is called first. Initialization sets up some defaults for the system. The operator keyboard works in exactly the same way as a serial terminal. That is, the pressing of a key causes the appropriate character to be sent to the controller serial port buffer. The function keys labeled **F1** to **F6** return respectively the characters 'A' to 'F'. A legend can be printed to the LCD screen for each function key and the corresponding character pressed and checked in the program, giving a context sensitive 'soft key' type operator interface.

The **RETURN** statement causes the program to jump back to the statement directly after **GOSUB init** (the main loop) is called.

The main loop simply prints up the following message on the LCD display:

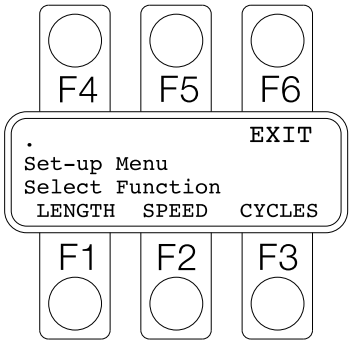


Note the use of **START, SETUP** and **MANUAL** to create user definable legends for the function keys **F1, F2** or **F3** on the keypad. The routine then loops, checking to see if a key is pressed at the keyboard. **F1, F2** and **F3** return the characters "A", "B" and "C" when pressed. For instance pressing **F3** causes a "C" character to be sent to the controller which then causes the set-up subroutine to be called. Note the use of **INKEY** to read the value of a key pressed.

Pressing **F1** will start the operation. **F2** sets up the parameters for speed, length and number of cycles. **F3** enters manual mode, allowing the user to move the material backwards and forwards.

Manual mode shows the use of **READKEY** to only move while a key is pressed (refer to subroutine **#manual**). This feature is not normally available with serial terminals. The **X Y** and **Z** cursor keys on the operator keypad return the values: X ('x' and 'u'), Y ('y' and 'v'), Z ('z' and 'w'). **READKEY** returns the value of the key that is currently *pressed*. This is different to **INKEY** which returns the value of a character in the serial port buffer. The motor is actually moved using the **JOG** command (continuous speed control).

Pressing **F2** will enter set-up mode, and the following display shown:



Pressing **F1, F2** or **F3** calls the **#get_len, #get_cy** and **#get_sp** subroutines which allow the operator to enter data. The **INPUT USING** command is used to provide formatted input. In this case the number is displayed as three integers.

The **#start** routine contains the code to start movement. In this example, a relative move (distance from start) is executed on the material feed axis; followed by an absolute (relative to a fixed zero position) up-down movement on the press axis. Note that in both the case of **MOVER** (move relative) and **MOVEA** (move absolute) the **GO** command is used to start motion. This command is required for positional moves, but not for continuous moves such as **FOLLOW** (following an external encoder) or **JOG** (constant speed control).

The move commands are surrounded by a **FOR - NEXT** loop which causes them to be repeated a number of times specified by the variable **cycles**, which is entered by the user during the set-up routine.

The **#ONERROR** routine near the end of the program is a routine which is called in the event of a controller error, such as when the motor jams or a limit switch is hit. In this case, the routine prints an error message and waits for a key to be pressed before re-running the program from the start.

The program can be tried and some of the parameters at the operator keypad changed. Program execution can be aborted by pressing **[CTRL]+[E]** at the terminal screen.

9.3.4 Using Batch Numbers

A common requirement in a cut-to-length machine is to store the parameters (speed, length and number of cycles etc.) that have been entered for different products and restore these quickly using a batch code. MINT allows this by using array variables and making the index to the array a batch code. More information on array variables is given in the MINT Programming Manual.

The program **FEEDER2.CFG** is an example of this type of program. The variables: **slew_speed, length** and **cycles** are redefined to be arrays each of length 99 by using the **DIM** statement:

```
DIM slew_speed(99)
DIM length(99)
DIM cycles(99)
```

A fourth option is added to the main menu allowing the entry of a batch number into the variable batch which is then used whenever a reference is made to the above variables. Therefore the move statement becomes:

```
MOVER = length(batch)
```

where **batch** is the index to the array called **length**.

Another advantage of using arrays to store operator input information is that they can be uploaded into a computer and stored to disk in a similar manner to *Program* and *Configuration files*. Similarly a file can be created which contains the data for each batch number and this can be downloaded to the controller. This is all achieved, while the program on the control system is running by using the **LOAD** and **SAVE** commands.

9.4 X-Y Teach and Replay Program

This example program illustrates the use of array variables for storing and replaying positional data programmed by the operator using the X and Y joystick keys on the operator panel. This program records ten x,y data points and then replays them printing the X Y position on the LCD panel as it does so. The *Configuration file* **CONFIG.CFG** is used for the application. In order to run this program, it is not necessary to have an XY table - two motors will work just as well.

This example contains considerable terminal I/O and therefore formatting of the information printed on the screen is important. Note the use of the **LINES** keyword to print information on the screen, and the , (comma) after any statement on line 4. This suppresses the carriage return that follows any normal print statement. This is important, since the LCD screen has only four lines of display. A carriage return on line four will cause the entire screen to clear.

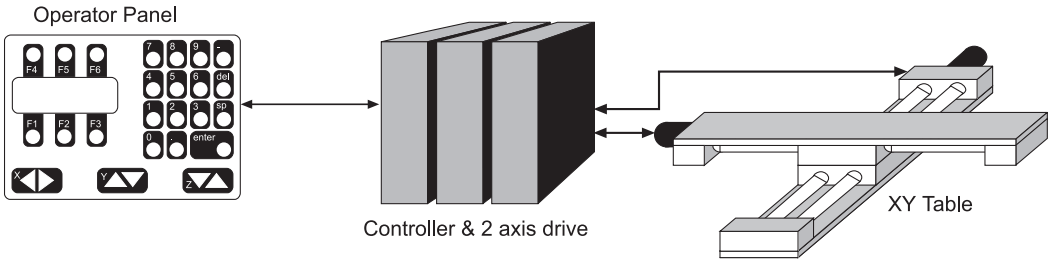


Figure 29: XY Table Control System

TEACH.MNT

```
REM Filename: teach.mnt
REM XY Table Example: teach and replay for an insertion application
DIM x_position(10) : REM 10 points of data
DIM y_position(10)

KEYS "" REM enable keyboard
CLS     REM clear screen
```

```
LOOP
  LINE 1,"XYZ WIDGET COMPANY"
  LINE 2,"Press Teach/Replay"
  LINE 3,""
  LINE 4,"TEACH  REPLAY",
  REM Note comma above to stop line feed which would cause LCD to flicker

  REM Wait for a key to be pressed
  key = INKEY  REM Read keyboard
  IF key = 'a' THEN BEEP:GOSUB teach
  IF key = 'b' THEN BEEP:GOSUB replay
ENDL

REM Subroutine to teach points and record in arrays
#teach
LINE 1,"Teach mode"
LINE 4,"RECORD",
REM record 10 points of data
FOR point = 1 to 10

  LINE 2,"Move to posn: ",point,
  REPEAT  REM repeat jog motors until record button F1 is pressed

   REM motor X
   IF READKEY = 'x' DO
    JOG[0] = 30 REM 30 mm/s
   ELSE IF READKEY = 'u' DO
    JOG[0] = -30
   ELSE
    STOP[0] REM key released - stop motor
   ENDIF
   ENDIF

   REM motor Y
   IF READKEY = 'y' DO
    JOG[1] = 30 REM 30 mm/s
   ELSE IF READKEY = 'v' DO
    JOG[1] = -30
   ELSE
    STOP[1] REM key released - stop motor
   ENDIF
   ENDIF

   REM print X,Y positions on line 3
   LINE 3,"X",POS[0] USING 4;"Y",POS[1] USING 4

  UNTIL INKEY = 'a'
```

```
    BEEP
    x_position(point) = POS[0]  REM Read position of X axis
    y_position(point) = POS[1]  REM Read position of Y axis

  NEXT  REM get next point

  RETURN

  REM Replay learnt points
  #replay
    LINE 1,"Replay mode"
    LINE 4,"STOP",
    FOR point = 1 TO 10
      VECTORA = x_position(point), y_position(point)
      GO
      REM If stop key pressed then exit
      IF INKEY = 'a' THEN STOP[0,1]:EXIT

      REM print X,Y positions on line 3
      LINE 3,"X",POS[0] USING 4;"Y",POS[1] USING 4
    NEXT
  RETURN
```

This program uses the **READKEY** function to jog the motors X and Y back and forth in response to the pressing or activation of the *Keypad*Node **X** and **Y** keys. It's possible however, to adapt the program to work with a standard potentiometer joystick connected to the analog inputs. The analog inputs return a 10 bit value 0 - 1023 between -10V and +10V (or 0-5V). The optimum way to connect the joystick is to connect +/-12V across the potentiometer and connect the wiper to the analog inputs 1+ and 2+. The differential inputs **1-** and **2-** should be connected to analog ground.

The analog inputs, read by the keywords **ANALOGUE1** and **ANALOGUE2** (abbreviated to **A1** and **A2**), will return 0 when the joystick is in the fully negative position and 1023 when in the fully positive position. A value of 512 will be returned when the joystick is in the middle position. Most joysticks are equipped with trimmers to adjust the zero position, but electrical noise can cause some jitter about 0V, so it is advisable to employ a deadband of 10 counts around the zero point.

The following code fragment illustrates the implementation of analog joystick control, which would replace the **REPEAT ..UNTIL** loop in **TEACH.MNT**:

```
  REPEAT  REM repeat until record button F1 is pressed
    REM jog at X,Y speed given by analog inputs, range ..
    REM is 0-1024, subtract 512 to give bi-directional control
    jog_x = A1-512
    jog_y = A2-512
    REM deadband of 10 points either side of zero
```

```
  IF ABS(jog_x) < 10 THEN jog_x = 0
  IF ABS(jog_y) < 10 THEN jog_y = 0
  JOG = jog_x, jog_y
UNTIL INKEY = 'a'
STOP[0,1] REM stop jog motion
```

Note that this example provides variable speed movement on the table depending on the position of the joystick, and also allows simultaneous movement on X and Y, which is not possible using the operator panel.

Another worthwhile feature would be to home the axes during power-up of the system. This can be achieved by using limit switches on the table and the **HOME** keyword. Further details of this are given in the MINT Programming Manual.

9.5 Software Gearbox Example - Coil Winding Machine

The controller has a *software gearbox* function making it ideal for controlling machines that have to follow a master encoder or pulse train. A typical application is to wind wire onto a drum rotating at a speed dictated by a mechanical drive from a wire drawing machine. When the spool is empty, the spool rotates at the highest speed, slowing down as layers of wire are wound onto the spool. The thickness of wire, width of the spool, and spacing between each wire must be changed to cope with different thickness of product.

An encoder on the spool rotation provides a speed signal. The controller uses this signal as a reference for a servo motor which drives the spool back and forth via a lead screw.

An operator panel on the front of the machine allows the operator to change *the following ratio* (spacing between wires) and the position of the left and right end of travel.

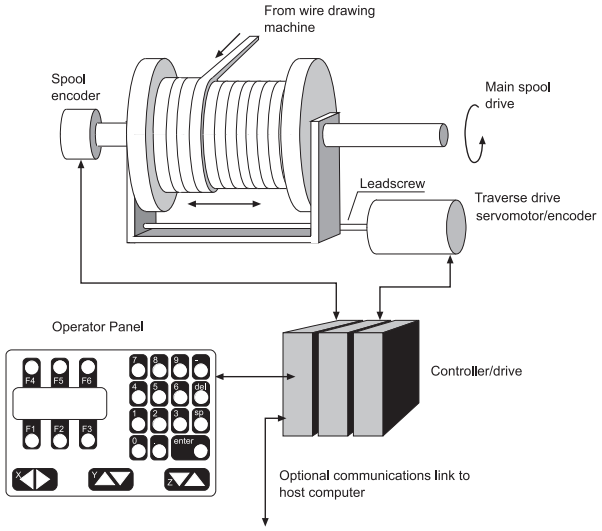


Figure 30: General Layout of Coil Winding Machine

The example MINT program uses the standard *Keypad*Node with six function keys returning **a,b,c,d,e** and **f**, plus numeric and enter keys. This example however does not take advantage of the 'soft-key' approach used in previous examples.

SPOOLER.MNT

```
REM Filename SPOOLER.MNT
REM Example program for stand alone spooler

REM defaults
AXES[0]
ratio = 1  REM 1:1 gearbox ratio
SCALE = 1000  REM 1000 counts on servo encoder = 1mm of travel
left_limit = 10
right_limit = 210
REM perform zero home on limit switch (negative direction)
HOME = _neg

REM infinite loop to provide operator display
LOOP
  CLS : REM clear KeypadNode display
  PRINT "XYZ Ltd Spooler"
  PRINT "Press:"
  PRINT "F1 to start"
  PRINT "F2 enter parameters"
  REPEAT
    key = INKEY : REM read keyboard buffer
  UNTIL key = 'a' OR key = 'b'
  IF key = 'a' THEN GOSUB spool
  IF key = 'b' THEN GOSUB enter_parameters
ENDL

#spool : REM subroutine to perform spooling
  CLS
  PRINT "XYZ Ltd Spooler"
  PRINT "Spooler running"
  PRINT "Press F3 to stop"
  REPEAT
    REM forward direction
    FOLLOW = ratio
    PAUSE POS > right_limit
    REM reverse direction
    FOLLOW = -ratio
    PAUSE POS < left_limit
  UNTIL INKEY = 'c'  : REM stop function key pressed
  STOP
RETURN
```

```
#enter_parameters : REM subroutine to change left, right limits and ratio
  CLS
  INPUT "Left limit",left_limit
  INPUT "Right limit",right_limit
  INPUT "Ratio",ratio
RETURN
```

9.5.1 Program Narrative

The variable **ratio** is used to store the 'software gearbox' ratio. That is, the number of servo motor encoder pulses moved per pulse from the spool encoder. This ratio is variable between -127 and +127, in increments of 0.003. **left_limit** and **right_limit** store the limits of linear travel. In this example, there are 4000 quadrature encoder counts per revolution of the servo motor encoder and a 4mm pitch lead screw. Therefore, setting the keyword **SCALE** to 1000 means that the limit positions may be expressed in mm.

This program is an example of a spooler with operator panel. A typical enhancement on practical machines is a 'dwell on reversal' to allow the material to ride up onto the new layer.

9.5.2 Remote Operation Using the COMMS array

In many applications the spooler system may be required to be adjusted remotely, under instruction of a host computer and perhaps while the spooling operation is in progress. The controller's protected communications protocol allows data to be exchanged between a host computer and the controller by means of a 99 location array called **COMMS**. **COMMS** operates like a dual port RAM; the host computer can write to and read the value of specific locations, over the serial port, for use by the controller program. This happens automatically, without any overhead in the MINT program itself.

For remote operation, the variables **ratio, left_limit** and **right_limit** would be replaced by three **COMMS** letterbox addresses:

```
REM routine to perform spooling
  LOOP
    REM forward direction
    FOLLOW = COMMS(1)  : REM COMMS(1) = ratio
    PAUSE POS > COMMS(2) : REM COMMS(2) = right limit
    REM reverse direction
    FOLLOW = -COMMS(1)
    PAUSE POS < COMMS(3) : REM COMMS(3) = left limit
  ENDL
```

If a value is changed by the host computer, this automatically appears in the program while it is running. A protected protocol is used to make the system fault tolerant. This protocol is ANSI 3.28 based, compatible with many other manufacturers of drive systems. The structure and implementation of protected communications are discussed in detail in the MINT Programming Manual. These cover the use of *cTERM* for reading and writing to the controller using protected communications.

9.6 Infeed Packaging Machine

MINT offers a number of features to facilitate the control of product that is continuously moving on a production line. A popular application in this area is Infeed machines, where the requirement is to put irregularly spaced product into pockets on a conveyor, so that they may be fed into a downstream machine.

The creation of 'order from disorder' is a taxing application of the controller, but has a large application base in the food, domestic product and pharmaceutical industry.

The following explains a simple Infeed system and the use of the **OFFSET** keyword.

The Infeed machine accepts irregularly spaced products on an input conveyor and feeds the product accurately into flights (pockets on a conveyor) on a *Parent machine*, typically a packaging machine that inserts product into boxes.

Product is fed onto the *Stripping Conveyor*, which runs slightly faster than the Input conveyor to separate product. The *Product* is next transferred to the *Synchronizing Conveyor*, which runs at the same base speed as the *Parent machine*, determined by following a pulse train generated by an Encoder (the Parent Machine Encoder). This Encoder is geared to the *Parent machine* such that one turn (1000 pulses) corresponds to one flight spacing. The encoder output is fed into the pulse input on the controller. The number of pulses per revolution is assigned to the MINT keyword, **WRAP**.

When a product passes the *Photocell*, the controller reads the position of the flight and calculates a correction on the belt to feed product accurately into a flight. The correction is expressed as a positional error in the product with respect to the flight position. The MINT **OFFSET** command is used to perform the correction. **OFFSET** works much like a normal relative positional move, except that the acceleration-deceleration profile is imposed on the base speed of the *Parent machine*.

In practice it is often possible to have two products on the synchronizing conveyor at one time, therefore the program has been written so that it can simultaneously sense and correct product position. The program also ensures that the second correction is not implemented until the first package is fed into the *Parent machine*.

A further function of the controller is to provide machine status information to the operator, such as number of products per hour etc. This information can be displayed on the operator panel, which is mounted on a convenient part of the machine. The *Keypad*Node is also used to enter set-up information, for instance product length and the speed of the metering conveyor.

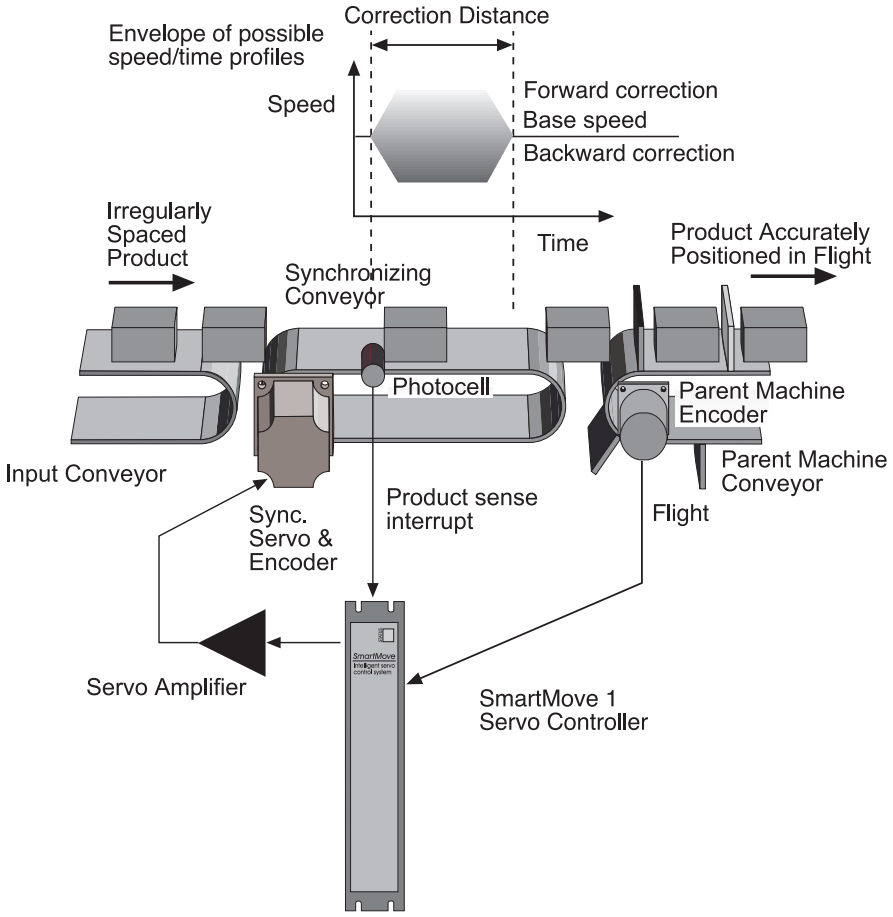


Figure 31: Diagram of Product Infeed Machine

The program is a simple example of the controller in an Infeed machine application. In more complicated applications, it is possible to collate a number of products and feed them into a single flight. The flexibility of the MINT programming language is such that these changes can be made quickly. The control program is stored in non-volatile RAM in the controller. As far as the operator is concerned, the system is a dedicated Automatic Product Infeed machine.

Infeed.MNT

```
REM  Infeed System
REM  Axis   0 Stripping Conveyor Drive
REM  Axis   1 Synchronising Conveyor Drive
REM  Input  1 Photo Electric Product Detector
REM  Output 1 Start Input Conveyor
REM  Master encoder produces 1000 counts per flight (one revolution)

REM Program starts here
GOSUB set_up
GOSUB main_program
END

#set_up         : REM subroutine to set-up system
 AXES [0,1]
 OUT1 = _on     : REM start input conveyor
 WRAP = 1000    : REM timer input is reset every flight (1000 counts)
 PULSE = 1;     : REM follow master encoder at 1/1 ratio
RETURN

#main_program
REM main loop to perform control over synchronising conveyor
 AXES[1]  : REM commands refer synchronising motor only
 LOOP     : REM main loop
  REM Wait for product edge (low/high transition)
  PAUSE NOT IN1     : REM low
  PAUSE IN1         : REM high
  REM product sensed - read master encoder
  product_position = TIMER
  REM Wait for previous correction to finish
  PAUSE MODE = _pulse
  REM perform new correction,  subtract 500 from product ..
  REM position to feed product into centre of flight
  OFFSET = product_position - 500
ENDL : REM back to start of loop
```

The program can be further enhanced to correct a product on the Synchronizing Conveyor while a product is already being corrected. This is achieved by reading back the **OFFSET** keyword and assigning the value to the new **OFFSET**.

10. Hardware Guide

This chapter describes in detail the hardware interface to the *SmartMove* controller. Each connector is labeled 'J' or 'SW'. The section headings identify the connectors being described in the following manner: *Connector Name; J Number*. For example *User Outputs: J4*, refers to the digital outputs as shown in10.1, Figure 32.

Under each section, a table defines the pin type (either input or output or both) and the associated MINT keyword if applicable.

10.1 Operating Environment

The safe operation of this equipment depends upon its use in the appropriate environment:

- At an altitude of ≤ 2000m (6560ft) above sea level
- In a locally ambient temperature of 0°C to 40°C (32°F to 104°F)
- In relative humidity levels of 80% for temperatures up to 31°C (87°F) decreasing linearly to 50% relative humidity at 40°C (104°F), non-condensing
- The pollution degree according to IEC664 shall not exceed 2
- The 18V ac, 24V dc supplied to the unit to power the control circuit shall be isolated from the mains using double or reinforced insulation so as to constitute a safety extra low voltage supply. The inputs and outputs of control circuit shall also be confined to SELV circuits
- The atmosphere shall not contain flammable gases or vapors
- There shall not be abnormal levels of nuclear radiation or X-rays
- The product shall be secured by the slots in the flange, the protective earth/ground stud shall be bonded to a safety earth/ground by a 25A conductor.

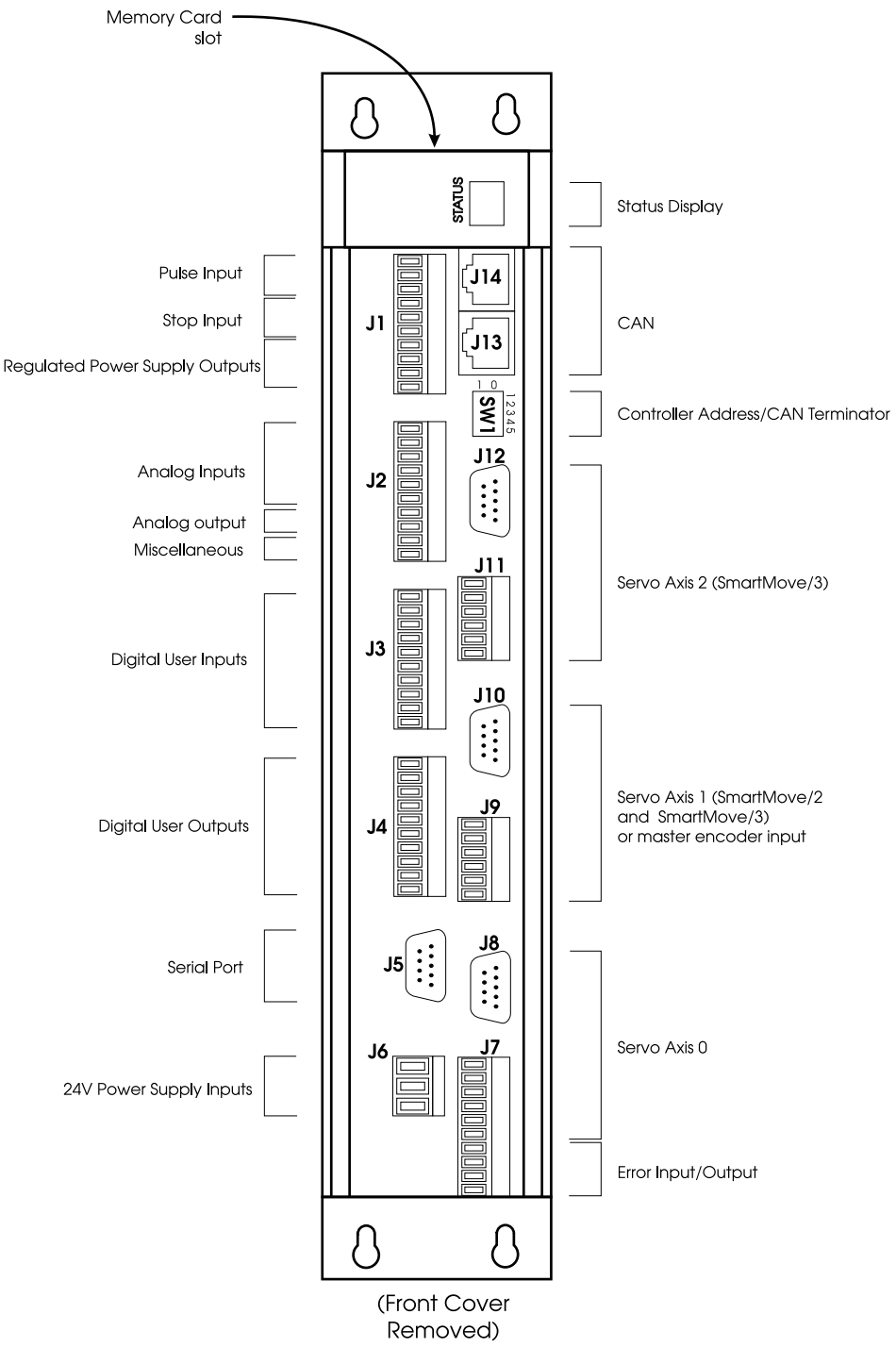


Figure 32: Front panel connections

10.2 User Outputs: J4

Label	Type	Description/Notes	MINT Keyword
0	Output	24V PNP Output bit 0	**OUT0**
1	Output	bit 1	**OUT1**
2	Output	bit 2	**OUT2**
3	Output	bit 3	**OUT3**
4	Output	bit 4	**OUT4**
5	Output	bit 5	**OUT5**
6	Output	bit 6	**OUT6**
7	Output	bit 7	**OUT7**
usr gnd	Input	Return for outputs	

The controller provides 8 uncommitted, software controlled, digital outputs.

The uncommitted digital outputs are driven by an octal Darlington array (Allegro UDN2987 device). Each output is capable of sourcing 50mA (nominal) on all channels continuously. A single channel can source up to 350mA. However the total output for all channels cannot exceed 500mA. These outputs include an over-current and over-temperature protection. In this situation an error condition will be flagged to the processor and the seven segment display will show 0. This will also cause an **Error 13** for which a user defined error handler can be created using the MINT **ONERROR** routine. Refer to the MINT Programming Manual for further information. The circuit is shown in Figure 33:

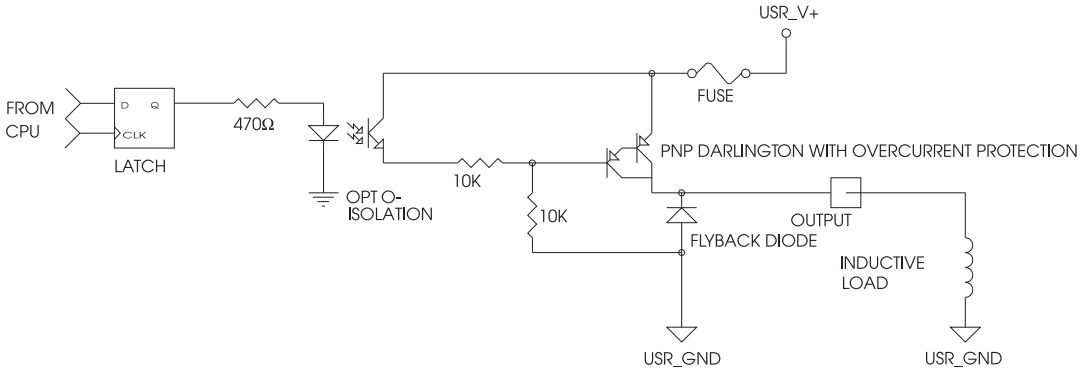


Figure 33: Output driver circuit (simplified schematic)

Some loads, such as tungsten filament lamps, may draw only 50mA in the steady state. However, they draw an inrush current when turned on which is large enough for the over current protection to trip out. In such cases the use of LED lamps should be considered. It is also possible to connect more than one output in parallel, although there is no guarantee of the outputs share the load equally.

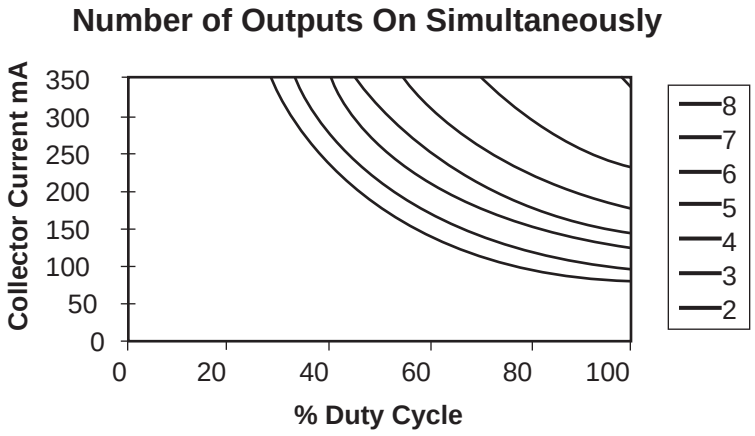


Figure 34: Output duty cycle

The above diagram shown in Figure 34 gives information about the current sourcing capabilities of the eight outputs.

Note that although these devices feature over-current and over-temperature protection they are not protected against an output pin rising above **usr-V+**. If this is likely then an external protection diode (e.g. 1N4001) should be fitted nearest to the controller whose anode is connected to **OUTx** and whose cathode is connected to **usr-V+**. Products built to E202-4A upwards incorporate on-card protection. Refer to Section 15.

The output cables need not be screened/shielded.

Opto-isolation is provided on these outputs to offer the user some flexibility in the earthing/grounding scheme, but not to provide level translation. usr-gnd must be connected to system ground - typically at a star point

10.3 User Inputs: J3

Label	Type	Description/Notes	MINT Keyword
0	Input	24V PNP input. 24V will activate the input	**IN0**
1	Input	bit 1	**IN1**
2	Input	bit 2	**IN2**
3	Input	bit 3	**IN3**
4	Input	bit 4	**IN4**
5	Input	bit 5	**IN5**
6	Input	bit 6	**IN6**
7	Input	bit 7	**IN7**
usr-V+	Input	24V source for inputs	

The controller provides 8 uncommitted digital inputs which can be configured as interrupts within MINT. Refer to the MINT Programming Manual. (see **#INx**).

The uncommitted inputs are buffered and interpreted in the same way as the **home** and **limit** inputs. A floating or low input is read as a 1 and an input connected to **usr-V+** is read as a 0. The state of the inputs is read using the MINT IN keyword.

All digital inputs include a reverse bias protection. This means that if the input is accidentally connected to a negative voltage (e.g. -24V) no damage will be caused.

The input buffer circuit is shown in Figure 35:

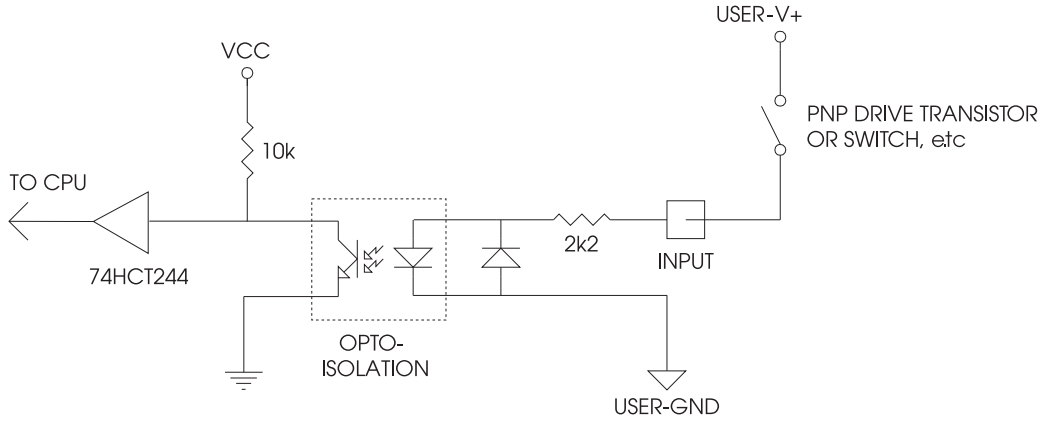


Figure 35: User inputs buffer circuit

Figure 36 shows how to connect the user inputs in the PNP configuration using either the external power or the internal power source:

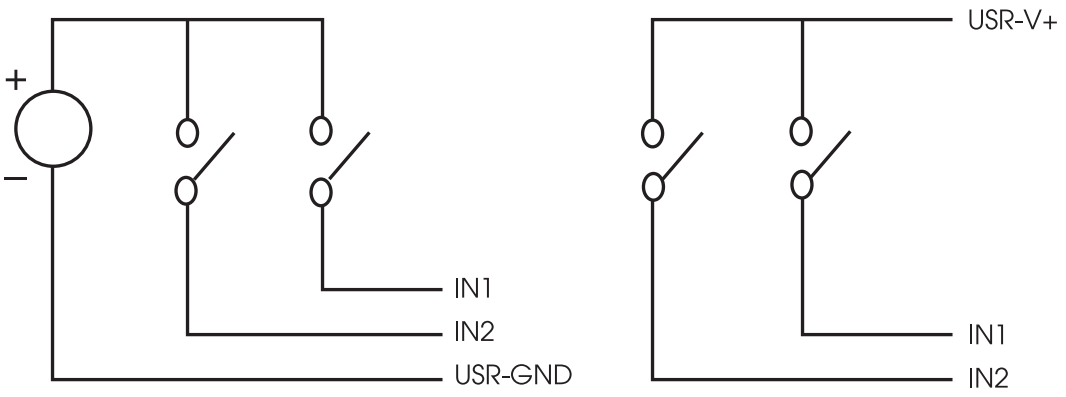


Figure 36: Example connection

The input cables need not be screened/shielded.

Opto-isolation is provided on these inputs to offer the user some flexibility in the grounding scheme but not to provide level translation, usr-gnd must ultimately be connected to system ground - typically at a star point

10.4 Analog Inputs: J2

Label	Type	Description/Notes	MINT Keyword
1+	Input	Positive input if +/-10V input used. Single ended input if 0 to 5V input used.	**ANALOGUE1 abbr: A1**
1-	Input	Negative input. Connect to 0V for single ended operation	
2+	Input	Positive input if +/-10V input used. Single ended input if 0 to 5V input used.	**ANALOGUE2 abbr: A2**
2-	Input	Negative input. Connect to 0V for single ended operation	
3+	Input	Positive input if +/-10V input used. Single ended input if 0 to 5V input used.	**ANALOGUE3 abbr: A3**
3-	Input	Negative input. Connect to 0V for single ended operation	

Three independent analog inputs are provided, each with 10 bit resolution in the range of ±10V or 0-5V. These may be used for analog sensor input or to provide a low cost joy-stick interface.

Each input is buffered and has a low-pass filter to reject noise, (-3dB @ 2kHz). For ±10V operation, the inputs are differential, therefore helping reduce the problems associated with differing ground potentials. This may be by-passed using jumpers JP1, JP2 and JP3 to allow 0 to 5V single ended operation. The jumpers are accessed by removing the front cover. Refer to section 11, figure 55.

Jumper Connection	Voltage Range
Pins 1 and 2	±10V - default
Pins 2 and 3	0 to 5V

On no account must the input voltage exceed the maximum rating shown above.

In the 5V configuration, the inputs are single ended inputs via the positive input and are referenced to the controller analog ground. In ±10V configuration the inputs are differential and not referenced to the controller ground. In normal circumstances the negative input is connected to the analog ground of the external equipment. It is important that this connection is made to this external analog ground, and not to external **system** or **digita**l ground. Connection to the external system ground may result in erroneous input readings caused by large return currents associated with motor control.

Each analog input signal should be connected to the system using a screened/shielded twisted pair cable, and the cable screen/shield should be connected to the ground pin on the power input J2. No other connections should be made to the cable screen/shield. The screen/shield should be connected at one end only.

The analog inputs can be read in MINT as 10 bit values using the keywords **`ANALOGUE1 (A1), ANALOGUE2 (A2)`** and **`ANALOGUE3 (A3)`** (MINT abbreviated keywords are shown in brackets).

Figure 37 shows how the analog inputs can be connected in either bipolar or single ended configuration:

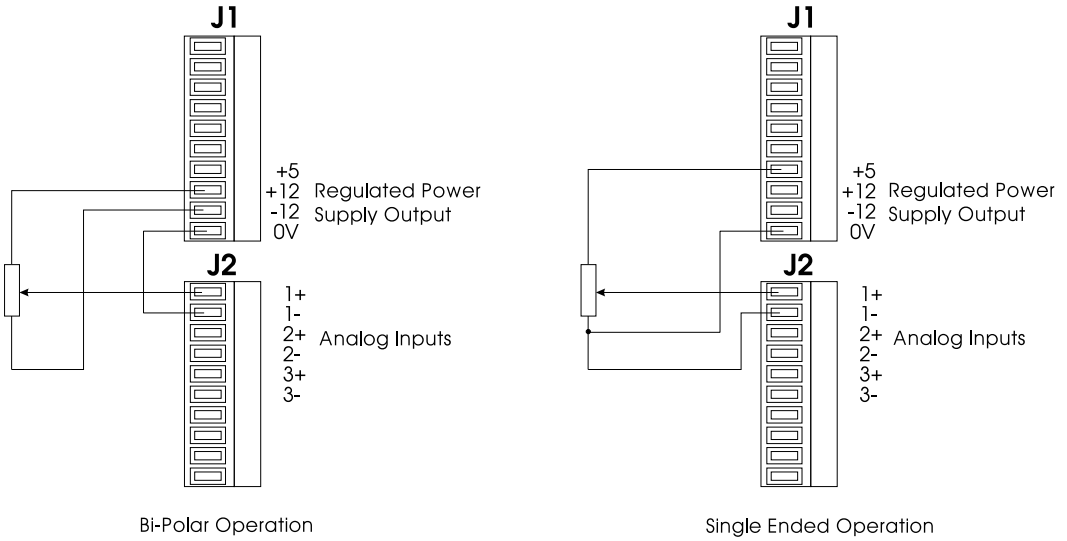


Figure 37: Analog inputs connected for bipolar (+\- 10v) operation and single ended (0 to 5v) operation

10.5 Analog Output: J2

Label	Type	Description/Notes	MINT Keyword
+/- **10V** +	Output	Positive signal	**AUXDAC** **AUX**
output -	Output	Negative signal	

The analog output can be used for system tuning by imposing the following error or instantaneous velocity on the output. Alternatively it can be used for a general purpose output with an output voltage in the range of -10V to +10V.

The use of the output for tuning is discussed in detail in the Section 8.

This output is robust enough not to require screening/shielding. However, it is advised in order to prevent contamination of the signal. The screen/shield ground pin should be connected on the power input J2.

10.6 Miscellaneous: J2

Label	Type	Description/Notes	MINT Keyword
fast interrupt	Input	Record axis positions within 30 microseconds	**FASTPOS**
system reset	Input	Resets the control when the line is pulled up to 24V.	

The fast interrupt is used to record the instantaneous position of all three axes. It is edge triggered and has a fast response time. It may capture spurious voltage spikes if care is not taken over cable connections. An individually screened/shielded cable should be used to connect to this signal, and the adjacent **usr-gnd** connection may be used to ground the respective screen/shield.

As with all isolated digital inputs, **fast interrupt** and **system reset** include reverse bias protection. This means that if the input is accidentally connected to a negative voltage (e.g. -24V) no damage will be caused.

The fast interrupt buffer circuit is shown in Figure 38:

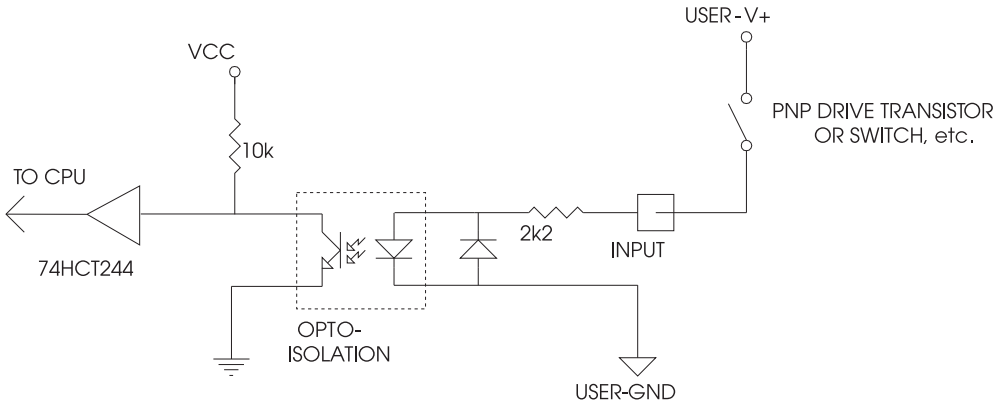


Figure 38: fast interrupt buffer circuit

The **system reset** input will cause a hardware reset when the line is pulled up to 24V. This can be useful to provide an external reset of the system when turning off the power to the controller would prove inconvenient.

10.7 Pulse Input: J1

Label	Type	Description/Notes	MINT Keyword
pulse	Input	Pulse input train for software gear box. 50kHz max. input frequency (24V pulse train)	**PULSE**
direction	Input	Direction of **pulse** input.	
reset	Input	Resets the pulse input counter on a rising edge.	**WRAP**

The pulse input provides the means for setting up a software gearbox following an external pulse train. Refer to section 9.6 for an example using the **pulse** input.

The controller provides a pulse following interface which consists of 3 signals:

- **pulse:** Pulse train input
- **direction:** Direction of pulse train
- **reset:** Reset pulse counter

The **pulse** and **direction** inputs feed directly into an up-down counter in the microprocessor (after isolation and buffering). This counter changes on *both* the rising and falling edges of the incoming pulse train.

Reset will reset the counter to 0 on a rising edge. The counter value can be read in MINT using the **TIMER** keyword. This would commonly be used where one channel of an encoder provides the pulse train, and the index pulse resets the counter on every revolution. The direction input, **direction,** determines whether the counter increments or decrements on each edge. If the **direction** signal is left unconnected or taken low then the counter will increment and if it is taken high (24V), the counter will decrement.

The pulse input has a maximum input frequency of 50kHz. An individually screened/shielded cable should be used to connect to this signal, and the adjacent **usr-gnd** connection may be used to ground the respective screen/shield.

By using an external conversion circuit, the counter input can be used to accept a quadrature encoder signal.

All digital inputs include a reverse bias protection. This means that if the input is accidentally connected to a negative voltage (-24V for example) no damage will be caused.

The buffer circuit for these inputs is shown in Figure 39:

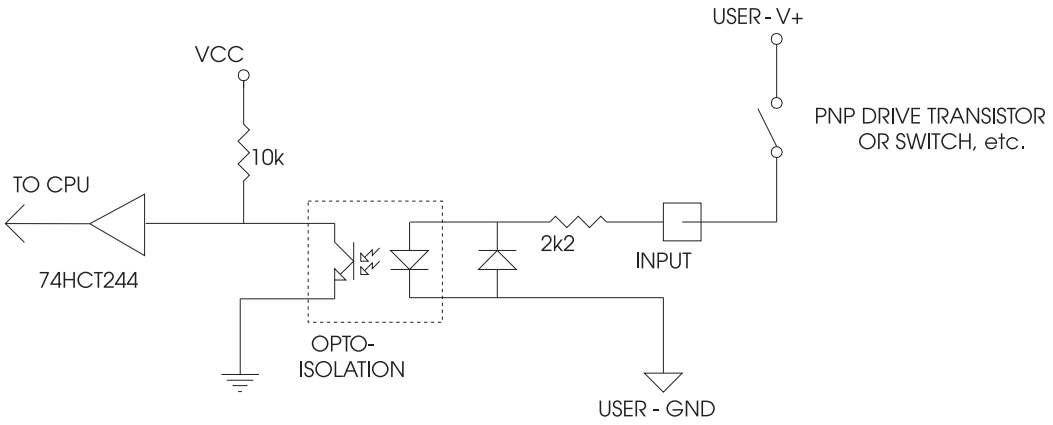


Figure 39: Input Circuit

10.8 Stop Input: J1

Label	Type	Description/Notes	MINT Keyword
stop input	Input	Brings all axes to a controlled stop if asserted. Connect to **user-V+** through a normally closed switch.	**STOPSW**
usr-V+	Input	24V source for **stop** input	
user-gnd	Input	0V for opto-isolated input-output	

The **stop** input, if asserted, will bring all axes to a controlled stop. The **stop** input is considered active when not connected and must be connected to **usr-V+** to allow motion. If it is not used it must be connected to **usr-V+** otherwise the controller will stay in the **stopped** condition, which is indicated by an S on the LED status display.

The **stop** input is useful where a controlled stop such as a machine guard is required.

The state of the stop input can be read in MINT using the **STOPSW** keyword. A subroutine, (**#STOP**) within MINT can be called in response to a rising edge on the stop input.

All digital inputs include a reverse bias protection. This means that if the input is accidentally connected to a negative voltage (-24V for example) no damage will be caused.

The **stop** input buffer circuit is shown Figure 40:

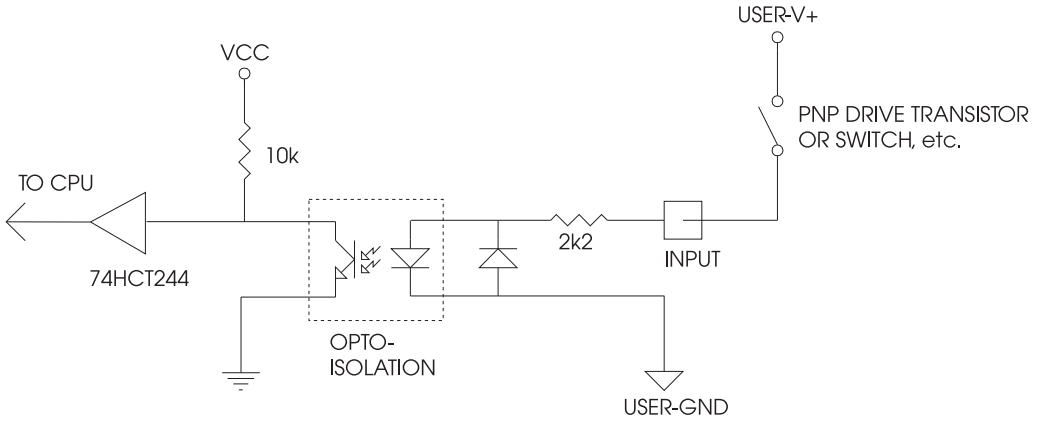


Figure 40: stop input buffer circuit

10.9 Regulated Power Supply Unit Outputs: J1

Label	Type	Description/Notes
+5	Output	Regulated 5V output
+12	Output	Stabilized +12V output
– 12	Output	Stabilized –12V output
0V	Output	0V output

Provides regulated **+5V** and **+/-12V** outputs for general use such as local potentiometers and instrumentation. **DO NOT USE FOR MACHINE INPUT/OUTPUT.** These voltages are provided for use with external equipment such as joysticks, analog input amps or pots located close to *SmartMove*

10.10 Power Supply: J6

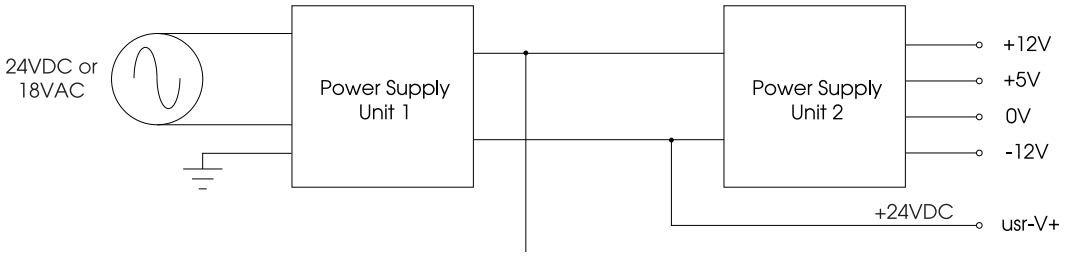


Figure 41: Power Supply.

The unit has an on board power supply to provide the following:

- 24V (**usr-V+**) output for machine I/O. Up to 500mA can be drawn from the **usr-V+** unregulated output. Note that **usr-gnd** is isolated from 0V. **usr-gnd** must be tied to the system star grounding point using external wiring.
- 5V regulated output for incremental encoders or potentiometers.
- +/-12V output, 5V output for general use. Note that these must not be used for machine I/O. It must not be attempted to draw more than 200mA from the 12V rails or more than 500mA from the 5V, including the current drawn by the encoders.

There are two possible input configurations:

- An *isolated* 24VDC power source at approximately 2.6A presented to the pins 1 and 2 of J6.
- An *isolated* 18VAC power source at approximately 4A presented to the pins 1 and 2 of J6.

These are sufficient to drive all the outputs at 50mA and with all the inputs on.

— WARNING —

Applying AC voltages to *SmartMove* (110V/220V) will damage the unit. Ensure that the power input voltages comply.

10.11 Battery Back-up

Program and data are retained in battery backed RAM while the controller is turned off. The battery, a rechargeable Nickel Cadmium (NiCad), is charged when the controller is powered up and will retain memory contents for at least 12 months if not recharged. The battery will take approximately 6 days to recharge after a complete discharge and has an expected life of four years.

10.12 Servo Drive Connections: J7, J8, J9, J10, J11, J12

Label	Type	Description/Notes	MINT Keyword
encoder	Input	Encoder connector for closed loop control (9 pin D-type)	**POS**
Home	Input	Home/Datum switch input. Normally closed to 24V	**HOME**
Limit	Input	Limit, end of travel limits. Normally closed to 24V	**LIMIT**
usr-V+	Input	24V source for **home** and **limit** inputs.	
demand+ /command	Output	±10V drive **demand/command** output	**DAC, TORQUE**
demand- /command	Output	±10V drive **demand/command** output	
screen	Input	Cable screen/shield	

10.12.1 Encoder Interface

The controller provides interfaces for three independent, three channel, incremental encoders (**chA, chB, index**) and operates with both single-ended (TTL or open collector) or differential (TTL or RS422) output types. It is recommended that line driver outputs (RS422) be used in all applications, since this gives increased noise immunity. It is important that each encoder cable is screened/shielded independently of other signals. The screen/shield must be connected at the controller end. It may also be connected at the encoder end provided the user is satisfied that the ground loop will not damage the cable. The maximum usable cable length is dependent on the encoder specification and the voltage drop in the encoder cable; thus the encoder cable should be kept as short as possible. Shielded twisted pair cabling must be used to achieve the quoted immunity performance of the controller. **chA** twisted with **!chA; chB** twisted with **!chB; index** twisted with **!index** and **5V** twisted with **0V**.

The maximum input frequency that the encoders can normally accept is 4.6 million quadrature counts/s. This equates to a maximum frequency for the **A** and **B** signals of 1.15MHz. This maximum however, is limited to short cables. The table below shows recommended length vs. frequency for differential signals. Single-ended TTL signals should be kept below 5m (approx. 16ft) and open collector signals are limited to 250kHz and 2m (approx. 61/2 ft).

Frequency	Max. Length
1.15MHz	2m (61/2 ft approx.)
500kHz	10m (33ft approx.)
250kHz	20m (651/2 ft approx.)
100kHz	50m (164ft approx.)
50kHz	100m (328ft approx.)
20kHz	300m (984ft approx.)
10kHz	700m (2296ft approx).
7kHz	1000m (3280ft approx.)

The input receiver circuit allows encoders with either single ended or differential line drivers to be used. However, single-ended encoders provide inferior noise immunity and should only be used on the shortest of cable runs (< 3m or approx. 91/2 ft) away from sources of interference. The input receiver circuit includes diode protection on the inputs to guard against noise spikes. The diodes prevent any input from exceeding 5.6V or below -0.6V. The circuit is shown in Figure 42:

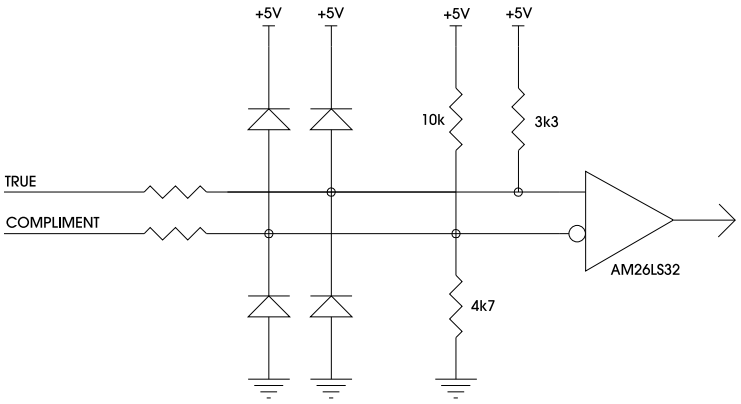
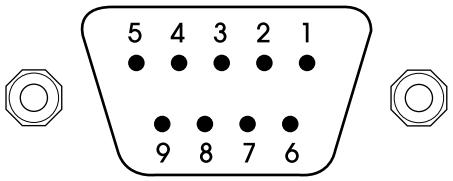


Figure 42: Encoder line receiver circuit

It should be noted that *SmartMove1* has 2 encoder inputs, with the additional encoder input being used for software gearboxes and cams, for example. *SmartMove 2* also has an additional encoder input, giving a total of three inputs.

The encoder inputs are brought out to 9 pin 'D' type female sockets. The encoder should be wired to a 9 pin 'D' male plug. Low profile hoods are available from ITT Canon (part no. DE-12107354)



Pin No.	Signal Name	Function	Type
1	**+5V**	Power to Encoder	Output
2	**index**	Index Mark	Input
3	**!chB**	Channel B Complement	Input
4	**scrn**	Cable Screen/Shield	Input
5	**chA**	Channel A	Input
6	**!index**	Index Complement	Input
7	**gnd**	Signal Ground	
8	**chB**	Channel B	Input
9	**!chA**	Channel A Complement	Input

10.12.2 Limit Inputs

In a typical application, the **limit** switch inputs would be connected to normally closed micro switches on the axis. Pressing the **limit** switch will cause the switch to become open circuit, the respective input then becomes active internally, resulting in a limit error. All axes will be disabled and the **demand/command** signal set to 0V. A limit error is indicated by an L on the LED status display. The **ERROR** keyword will return the value 3 when read. Refer to the MINT Programming Manual for more details on the **ERROR** keyword and error handling within MINT.

Because there is only one limit input per axis, if two end-of-travel limit switches are used then they must both be connected in series as shown in Figure 43.

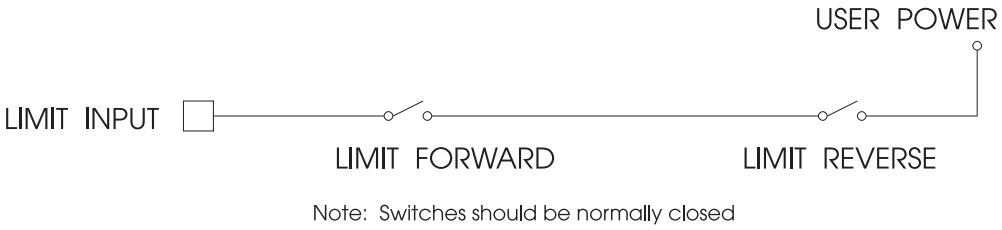


Figure 43: Dual Limit input switch connection

A possible problem with this arrangement is that it is not possible to determine which end-of-travel switch has been hit by simply reading the **limit** input with software. This can be overcome by introducing a double pole limit switch at one end of the axis and connecting this to the home input or a spare digital input. This means that when a limit switch is hit the program can determine whether this is the **forward** or **reverse** limit by reading the status of the home switch or relevant digital input.

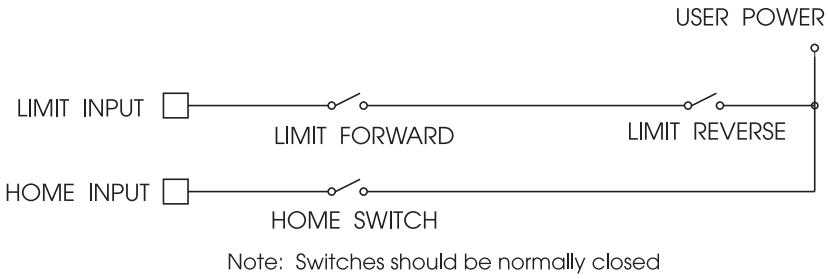


Figure 44: Dual limit and home Input connection

In many applications it is necessary to have separate **limit** and **home** switches. In these cases it is best to arrange the active areas of the switches to be set-up as shown in Figure 45.

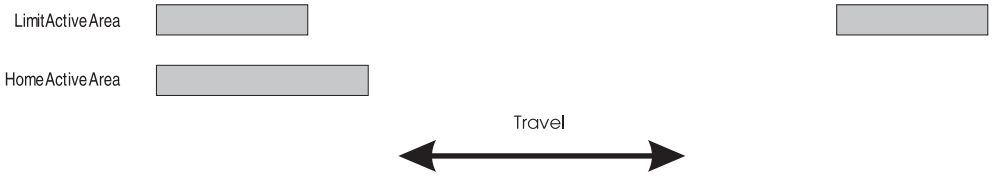


Figure 45: limit/home switch active areas

Hitting a limit switch causes an error which is handled in software by the ONERROR subroutine. Details of this are given in the MINT Programming Manual.

In an application where only some of the axes are being used, the redundant axes must have their respective limit switches tied to **usr-V+** for normal operation. Alternatively the MINT keyword **`DISLIMIT`** can be used to disable the detection of limit switches.

The state of the limit switches can be read using the MINT **`LIMIT`** keyword. A value of 1 (logically active) will be returned if the limit input is low or floating (switch open) and 0 (logically inactive) if connected to **`usr-V+`** (switch closed).

As with all digital inputs the limit switch input includes a reverse bias protection. This means that if the input is accidentally connected to a negative voltage (e.g. -24V) no damage will be caused.

Figure 46 shows the input buffer circuit and normal connections for all motion inputs.

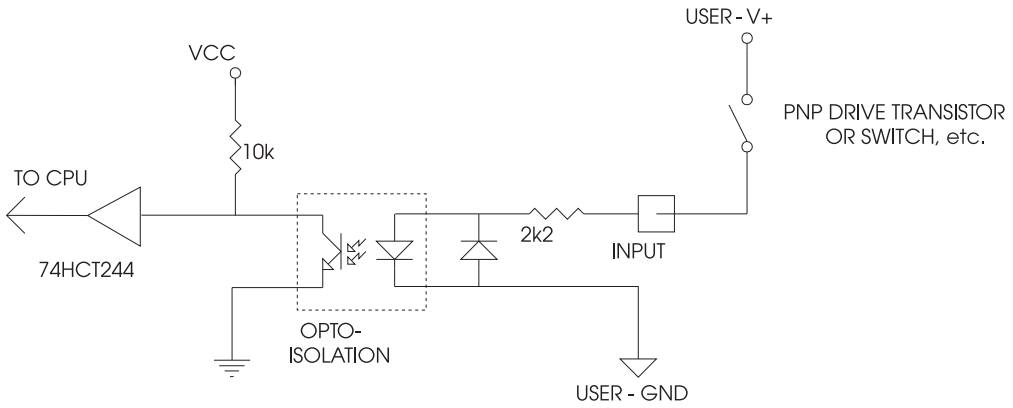


Figure 46: limit Input buffer circuit

10.12.3 Home Inputs

Up to three **home** inputs are provided; one for each axis. Like the limits these inputs require closed switches for normal operation. Unlike the limit switches, however, they do not have to be connected if not being used.

If the limit switch is used as a home point, both the limit input and the home input must be connected together. Noisy environments can cause problems on the limit switches which may result in a limit error during homing. A 100nF ceramic capacitor connected between the limit input and ground should eliminate this problem.

The state of the home switches can be read using the MINT **`HOME`** keyword. A value of 1 (active) will be returned if the **home** input is low or floating (switch open) and 0 (inactive) if connected to **usr-V+** (switch closed).

As with all digital inputs the **home** switch input includes a reverse bias protection. This means that if the input is accidentally connected to a negative voltage (e.g. -24V) no damage will be caused.

The input buffer circuit is shown in Figure 47:

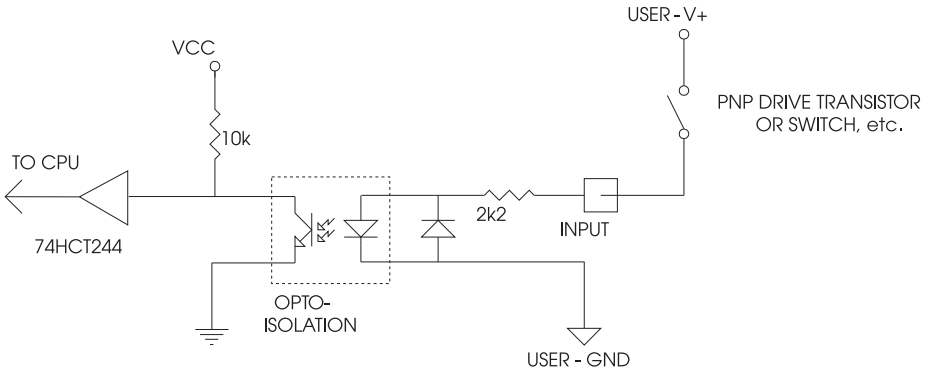


Figure 47: Input buffer circuit

10.12.4 Drive Demand/Command Outputs

The controller provides up to three ±10V (±0.1%) analog outputs for motor **demand/command** - one for each servo axis. A 12 bit DAC is used which gives a resolution of 4.9mV/bit. The optimum cabling arrangement is to use a separate shielded twisted pair cable, twist **demand+/command+** with **demand-/command-** and connect the screen/shield to screen/shield at the controller end only.

10.13 Error Output and Input: J7

Label	Type	Description/Notes	MINT Keyword
open	Output	Normally open connection of relay	**ENABLE**
closed	Output	Normally closed connection of relay	**ENABLE**
common	Output	Common connection of the relay	
drive error input	Input	Used to signal the controller that an error has occurred on a drive. The polarity of the input is controlled using the **ERRORIN** keyword.	**ERRORIN**

The **error** output provides a set of voltage free contacts for connection to motor drives and other equipment. The contacts are rated at 1A @ 24V when driving a *resistive* load. An *inductive* load such as a slave relay may be driven, but a freewheeling diode must be attached across the load. The **error** relay is energized when no faults are present. When the controller detects a fault such as:

- an end-of-travel limit switch is open
- maximum *following error* exceeded
- a programming **error** or the error input is active

the controller de-energizes the relay. In this way, if power to the controller is lost or the relay itself fails, all motor drives would be disabled, giving fail safe operation.

It is essential that the error output is connected to the drives and configured to the correct polarity with servo drives. The **error** output ensures that the drive is disabled during power-up. Once powered-up the **error** signal is maintained until deliberately removed by the software. This allows time for system gains to be set before the motors are enabled (relay energized).

The **error** input is used to detect error conditions in other parts of the system such as PLCs, or from the motor drives so that if a fault occurs on one motor the whole system is stopped.

The sense of this input is software-selectable for active high or active low, it may also be disabled in software.

The **error** input also includes a reverse bias protection. This means that if the input is accidentally connected to a negative voltage (e.g. -24V) no damage will be caused.

The **error** input buffer circuit is shown in Figure 48:

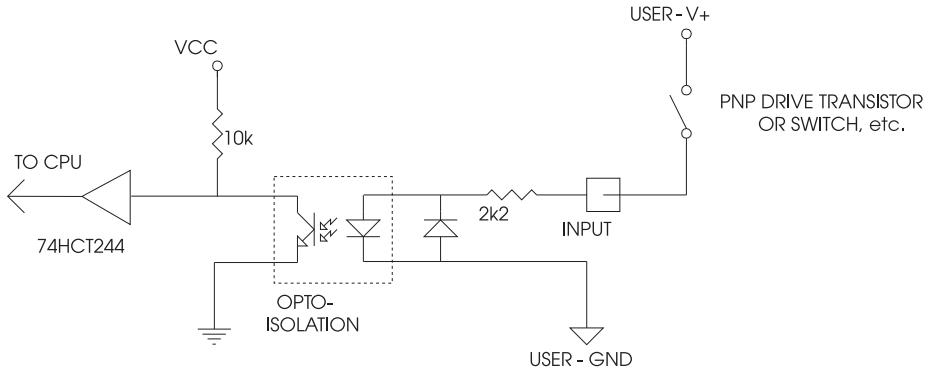


Figure 48: error input buffer circuit

10.14 Serial Port: J5

The controller has a full duplex serial port which can be specified to be either RS232 or RS485. It is possible to change from RS232 to RS485 or vice-versa in the field by moving jumpers. However, this is not recommended after the PCB has been assembled into the box as access is limited. Refer to Section 11.

The serial port is set-up for the following configuration:

- 9600 Baud default. Software selectable in MINT using the **BAUD** keyword.
- 1 start bit
- 8 data bits
- 1 stop bit
- No parity

MINT will transmit a line feed/carriage return (**<LF><CR>**) combination but only expects a carriage return (**<CR>**) from the host terminal.

cTERM, the terminal emulator program, is supplied on the enclosed diskette and is designed for use with the controller. It is available for both Windows and DOS.

10.14.1 RS232

The RS232 connections are brought out onto a male 9 pin D-type connector on the front of the controller. Screened/shielded cables should always be used.

The RS232 port is configured as a DTE (Data Terminal Equipment) unit so it is possible to operate the controller with any DCE (Data Communications Equipment) or DTE equipment. Both the output and input circuitry are single ended and operate between ±12V.

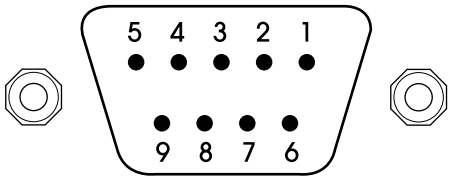


Figure 49: RS232 D-Type connector pin-out

Pin No.	Signal Name	Function	Type
1	**scrn**	Cable screen/shield	Input
2	**rxd**	Receive Data	Input
3	**txd**	Transmit Data	Output
4	**dtr**	Data Terminal Ready (Internal connection to pin 6)	Output
5	**gnd**	Signal Ground	
6	**dsr**	Data Set Ready (Internal Connection to pin 4)	Input
7	**rts**	Request to Send	Output
8	**cts**	Clear to Send	Input
9	**gnd**	Signal Ground	

The following table shows the wiring required for a standard IBM PC 25 pin or 9 pin connector:

Controller Pin No.	Signal Name	Function	Wire to: 25 Pin	Wire to: 9 Pin
1	**scrn**	Cable screen/shield	—	—
2	**rxd**	Receive Data	2	3
3	**txd**	Transmit Data	3	2
4	**dtr**	Data Terminal Ready (Internal connection to pin 6)	6	6
5	**gnd**	Signal Ground	7	5
6	**dsr**	Data Set Ready (Internal Connection to pin 4)	20	4
7	**rts**	Request to Send	5	8
8	**cts**	Clear to Send	4	7
9	**gnd**	Signal Ground	7	9

10.14.2 RS485

The RS485 connections are brought out onto the male 9 pin D-type connector on the front of the controller. The RS485 supports a full multi-drop protocol using a 4 wire system. Both the output and input signals are differential and operate between 0 and 5V. Screened/shielded twisted pair cabling should be used, with the true and complement signals twisted together to give maximum noise immunity.

Fail safe operation of the receiving line is achieved by inclusion of line biasing resistors. The values of these resistors are set for operation with a small number of cards. When a larger number of cards are required on the same line, the values may need to be altered.

The signals are also brought out onto a 9 pin D-type connector.

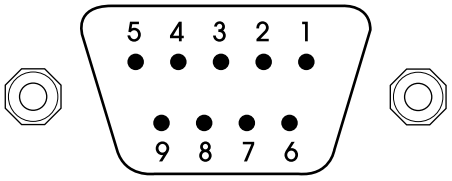


Figure 50: 9 Pin Connector

Pin No.	Signal	Function	Type
1	**scrn**	Cable screen/shield	Input
2	**rxd**	Receive Data	Input
3	**txd**	Transmit Data	Output
4	Not connected		
5	**gnd**	Signal Ground	
6	Not connected		
7	**!txd**	Transmit Data Complement	Output
8	**!rxd**	Receive Data Complement	Input
9	**gnd**	Signal Ground	

10.14.2.1 RS485 Multi-Drop

A multi-drop system can easily be configured using a ribbon cable and IDC D-Type connectors to the controller cards. A multi-drop layout is shown in Figure 51. The controller supports up to 16 cards on the serial line, where each card is distinguished by a unique address set by a 5 pole DIP switch. Refer to Section 10.15 for more details on the card address. Software details on multi-drop can be found in the MINT Programming Manual.

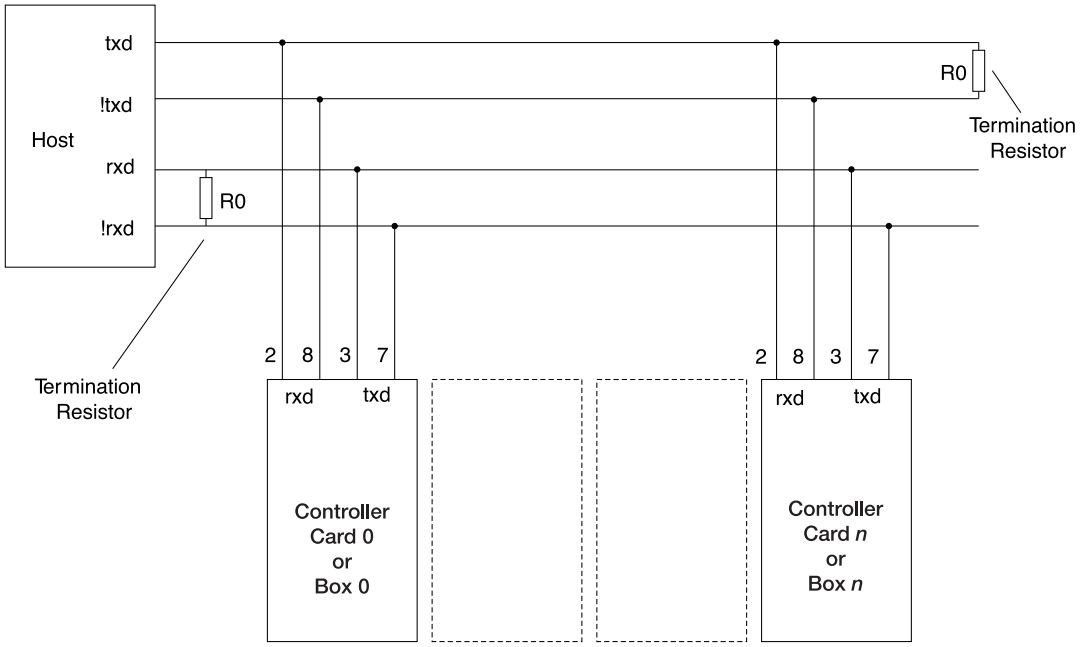


Figure 51: Multi-drop system

The RXD and TXD lines should be terminated with 120 ohm resistors at the receiving end as shown.

10.14.2.2 RS485 to RS232 Converter

Figure 52 shows a schematic for an RS485 to RS232 converter. This is useful for connecting a host PC to a multi-drop link.

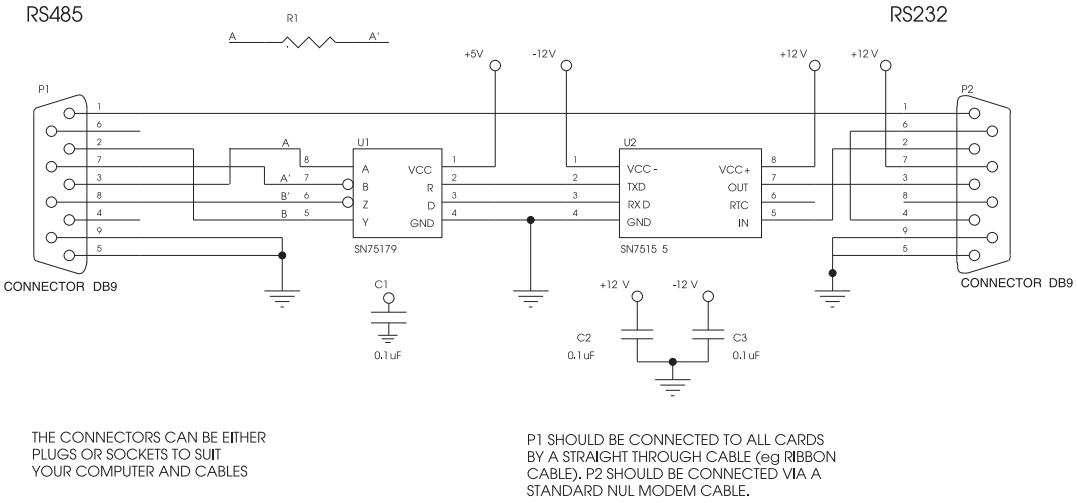


Figure 52: RS232 to RS485 Converter

10.15 DIP Switch (Card address): SW1

Up to 16 controllers can be connected together over a multi-drop RS485 link for host control. Each card on the link is distinguished by a unique address set by a 5 pole DIP switch beneath the CAN connectors on the front panel. Note that only the top 4 switches are used to set the card address. Switch 5 is used to switch in the CAN bus terminator described in section 10.16.

Switch Position 4 3 2 1	Address
0 0 0 0	**0**
0 0 0 1	**1**
0 0 1 0	**2**
0 0 1 1	**3**
0 1 0 0	**4**
0 1 0 1	**5**
0 1 1 0	**6**
0 1 1 1	**7**
1 0 0 0	**8**
1 0 0 1	**9**
1 0 1 0	**10**
1 0 1 1	**11**
1 1 0 0	**12**
1 1 0 1	**13**
1 1 1 0	**14**
1 1 1 1	**15**

On power-up, the controller will display the address number on the LED status for about 1 second. The DIP switch value can be read through MINT using the **CARD** keyword.

If the controller is used in a stand-alone system, the card address should be set to address **0**.

10.16 CAN Bus Port: J13, J14

SmartMove can communicate with I/O expansion modules via CAN. CAN offers 125kbit/s serial communications over a two wire twisted pair cable up to a maximum of 500m (approx. 1640ft) in length. CAN offers very high reliability of communications in an industrial environment, the probability of an undetected error rate is $4.7x10^{-11}$. CAN is optimized for the transmission of small data packets and therefore offers fast update of I/O devices (*io*NODES) on the bus. Several *io*NODES may be attached to the same controller via the CAN link using the CAL protocol. The maximum number of *io*NODES on the *SmartMove* CAN network is limited to three input nodes and three output nodes and one keypad.

The CAN is brought out to two RJ45 connectors (J13 and J14) located on the daughter board and these are wired in parallel as follows:

Pin	Signal
1	**can1+**
2	**can1-**
3	**n/c**
4	**gnd**
5	**n/c**
6	**n/c**
7	**can2+**
8	**can2-**

SmartMove has only one channel of CAN but the standard cabling system (which may be supplied with the unit) supports two channels. *SmartMove* may be connected to either channel by setting the jumpers located behind SK5 on the daughter board. The factory default is to connect *SmartMove* to CAN channel #1. Notice also that CAN nodes will have to be externally powered as there is no power connection on the RJ45s.

If *SmartMove* is at the end of the bus then position 5 of the DIP switch SW1 should be in the ON position. This connects the termination resistor.

SmartMove supports the following range of ioNODE devices:

- *inputNode 8*
- *outputNode 8*
- *relayNode 8*
- *keypadNode*

A very low error rate of CAN communication can only be achieved with a suitable wiring scheme. The following points should be observed:

1. CAN must be connected via twisted pair cabling. The connection arrangement is normally a simple multi-point drop. The CAN cables should have a characteristic impedance of 120Ω; and a delay of 5ns/m. Other characteristics depend upon the length of the cabling:

Cable length	Maximum bit rate	Specific resistance	Conductor area
0-40m (0-157ft)	1 Mbit/s	70mΩ	0.25-0.34mm^2
40m-300m (157ft-1180ft)	200 kbit/s	< 60mΩ	0.34-0.60mm^2
300m-600m (1180-2362ft)	100 kbit/s	< 40mΩ	0.50-0.60mm^2
600m-1000m (2362ft-3937)	50 kbit/s	< 26mΩ	0.75-0.80mm^2

2. Terminators should be fitted at both ends of the network only.

3. To reduce RF emissions and more importantly, to provide immunity to conducted interference, shielded twisted pair cabling should be used. If two CAN channels are bundled in a cable then each requires a twisted pair.

Cable screens/shields should not be connected to 0V on connector J13 and J14 as this will ground loop interference into the 0V plane on the processor board.

4. The 0V rails of all of the nodes on the network must be tied together through the CAN cabling. This ensures that the CAN signal levels transmitted by a *SmartMove* or *ioNODE* are within the common mode range of the receiver circuitry of other nodes on the network.

Cables may also be purchased from the factory.

10.17 Operator Panel (KeypadNode)

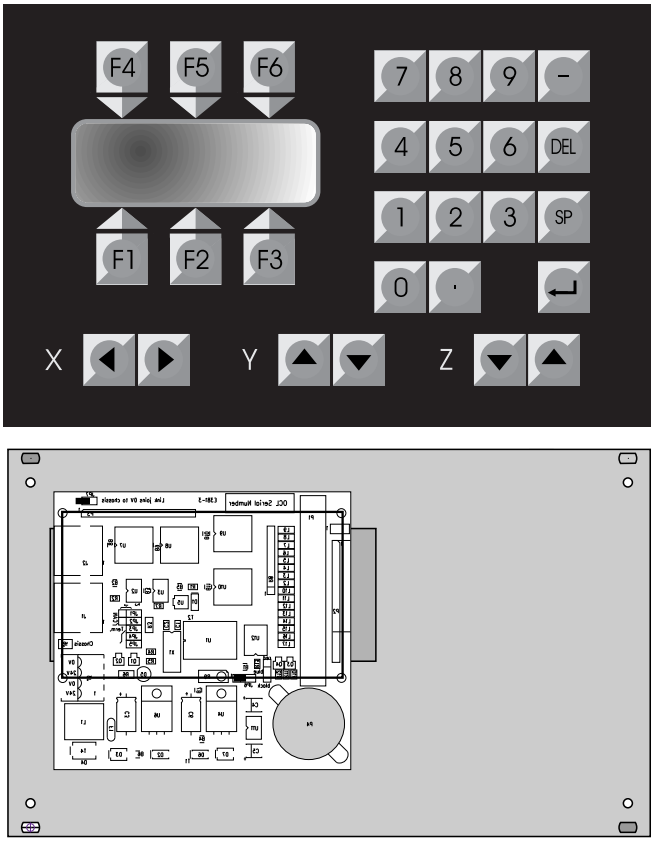


Figure 53: KeypadNode showing front and back view

The *Keypad*Node provides a general purpose interface suitable for stand-alone machines of all types. It is cost effective for simple functions, such as replacing thumb wheel switches and providing simple diagnostics, or it may be used as a fully interactive programming panel for special purpose machine control.

- High speed CAN bus connection.
- 20 character by 4 line LCD display.
- 27 keys, numeric keypad, function keys and XYZ control keys.
- Software controlled piezoelectric buzzer.
- Only one keypad can be used with *SmartMove* at any one time.

Full details of the MINT keywords can be found in the MINT Programming Manual. Full details of the *Keypad*Node can be found in CAN Peripherals.

10.18 Memory Card Interface

SmartMove supports an SRAM memory card for either program storage or memory expansion. 64K and 128K RAM cards are supported. Refer to the MINT Programming Manual for further details on how to use the memory card.

The memory card is used by placing it in the slot at the top of the unit. When inserting the card, ensure that the gold contacts are facing to the right (viewing the controller from the front).

Typing **PROG** or **CON** from the MINT command line will display the memory capacity size of the card.

When used for **OFFLINE** array storage, it must be ensured that the write protect switch on the card is set to off.

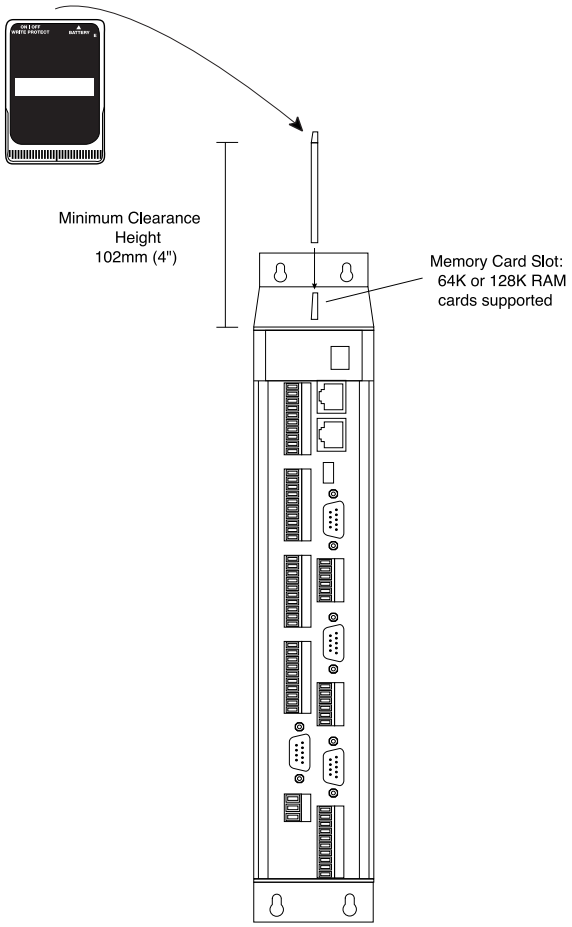


Figure 54: Memory Card Installation

11. SmartMove PCB Settings

Anti-static pre-cautions must be taken before handling the printed circuit board (PCB).

Access to jumpers, EPROM and other components can be accessed via a slide-down cover on the right hand side of the box. The cover is held in place by a set screw which should first be removed. A screwdriver may then be required to ease the cover free.

The jumpers, switches and trimmers are marked with an arrow and numbered. Refer to Figure 55. Details of each are given under the section headings “Name : Reference number”. For example “VR4 : 7”.

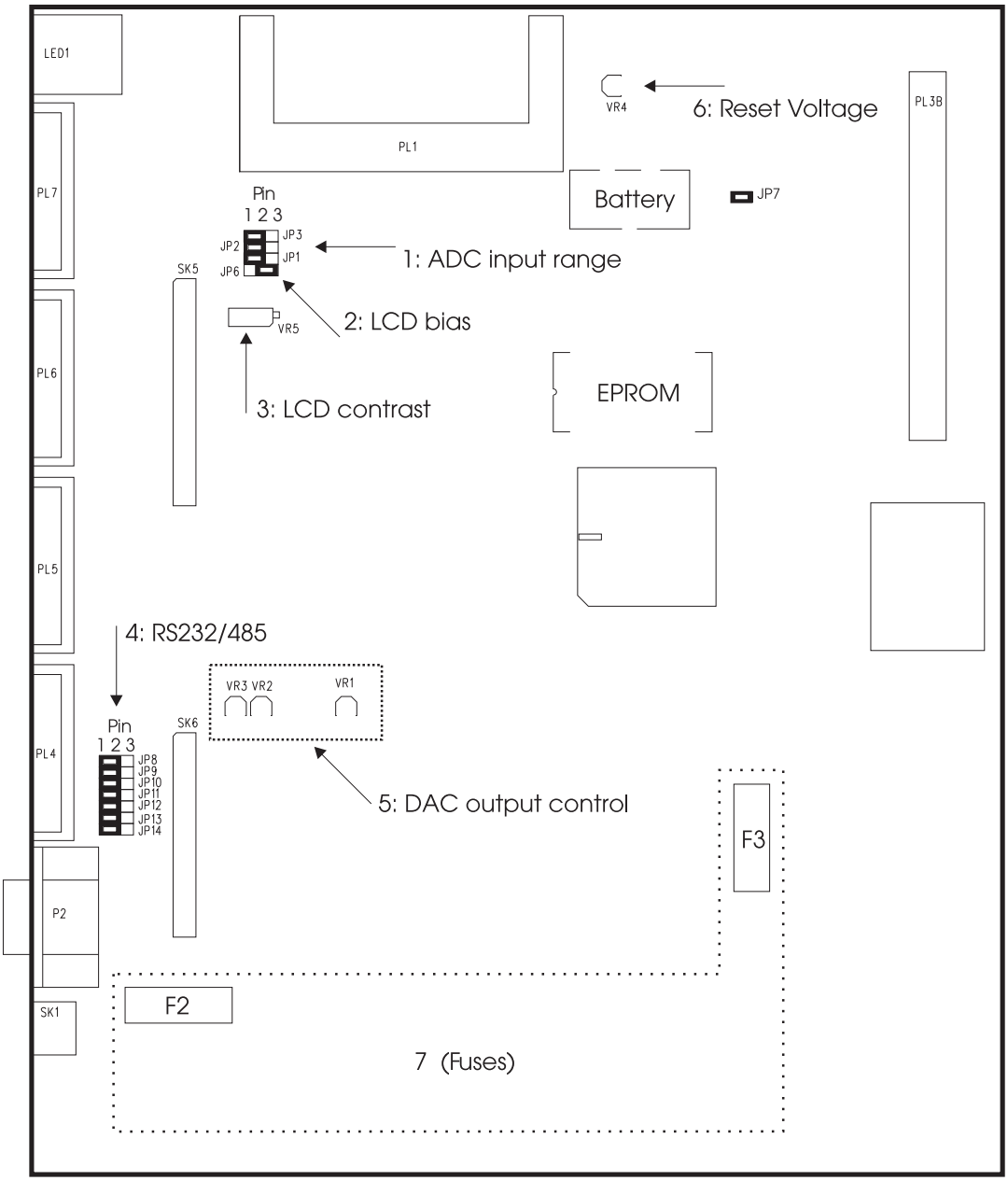


Figure 55: Board Settings

11.1 Jumpers JP1, 2, 3: 1

These jumpers set the voltage range for the inputs on the ADC (unipolar or bipolar). Placing the jumper over pins 1 and 2 sets the voltage range to (10V. Placing the jumper over pins 2 and 3 sets the voltage range to 0-5V.

Figure 55 shows the jumpers placed over pins 1 and 2, indicating all inputs set to ±10V.

11.2 Potentiometers VR1, 2, 3: 5

The potentiometers are used to adjust the offset voltage on the DAC outputs. By varying them it is possible to obtain exactly 0V on an output when the demand is zero.

These potentiometers are factory set and should not be altered.

11.3 Potentiometer VR4: 6

On power-up and power-down, voltages on the power rails are variable and not predictable. In order that the processor may behave predictably, it must be reset when the voltage level is within specified limits. It is necessary to detect the voltages on its power rail (5V) with special circuitry and issue one reset signal when the voltage level is within these limits.

The voltage level is factory preset to 4.7V and should not be altered.

11.4 Fuses F2, F3: 7

All the fuses are located at the bottom of the board and are shown enclosed by a dotted line.

- F2 is placed on the 24V input line and has a rating of 5A (anti-surge/slow-blow).
- F3 is the output drive fuse and has a rating of 650mA (anti-surge/slow-blow).

12. CE Marking

12.1 Directives

CE marking indicates that a product complies with all of the relevant directives. A brief discussion of the directives follow.

12.1.1 Machinery Directive 89/392/EEC

This relates to the safety of machinery. Any machine having a moving part has to comply with the essential requirements of this directive. This also applies to some mechanical sub-systems, such as fans. Electronic sub-systems, such as *SmartMove*, do not fall within its scope. It should be noted that to meet the safety requirements of this directive, the primary machine safety system cannot be based on controller software. The paying of a notified body to audit the software and provide a safety case is prohibitively expensive. A hard-wired system using guard switches (to protect personnel or equipment) should be used.

12.1.2 Low Voltage Directive 72/23/EEC

Applies to equipment powered from 75V dc to 1500V dc or 50V ac to 1000V ac. Therefore this directive does not apply to *SmartMove*.

12.1.3 EMC Directive 89/336/EEC

This directive applies to all ‘apparatus liable to cause disturbance or the performance of which is liable to be affected by such disturbance’. The expression ‘apparatus’ is defined for the purpose of the regulations as ‘..an electrical or electronic appliance or system consisting of a finished product or products having an intrinsic function which is intended for the end user, and is supplied or intended for supply or taken into service or intended to be taken into service as a single commercial unit’.

This wording brings automated machines within the scope of the directive but not necessarily their electronic sub-systems. This is because of their lack of intrinsic function to the end user. However, as *SmartMove* is built to issue E202-4A (refer to section 15) it will be CE marked with respect of the EMC directive, subject to the following conditions.

12.2 EMC Performance of *SmartMove*

The data applies to products built to E202-4A or later. It should be noted that the equipment is suitable for light or heavy industrial use. The text from the full report is appended.

Test	Standard	Comments
EN55081-1 Generic emissions, light industrial	EN55022 Radiated emissions	Connections to all ports as recommended in this manual, with the exception of connections to **pulse, direction, reset, +5V, +12V, –12V, 0V,** keypad, serial port operated in RS232 mode. Passed the B limit.
EN55082-2	IEC801-2	Air discharge, contact to case, had no effect.
Generic immunity, heavy industrial	ESD	Contact to shells of D-type connectors caused controller reset but no permanent damage. No contact to connector pins.
	IEC801-3 Radiated immunity	Connections to all ports as recommended in this manual, except no connections to **pulse, direction, reset, +5V, +12V, –12B, 0V**, keypad, serial port operated in RS232 mode. Test covered 10V/m 26-80MHz, 10V/m 80-1000MHz with 80% 1kHz modulation, equipment functioned correctly throughout.
	IEC801-4 Fast burst transients	1kV via clamp to **CAN, demand/command, Din, Dout, pulse,** RS232, equipment functioned correctly throughout. 2kV to 18 Vac input.

12.3 Conditions of CE Marking

The apparatus shall be connected in accordance with the recommendations of this manual. There are certain restrictions regarding the length and type of cable:

Cable type	Ports
unrestricted	**usr-V+, usr-gnd,** user inputs, user outputs, **home limit, drive error, power** input, **drive enable** relay
unrestricted but screening/shielding is advised	pulse, direction, reset, demand/command, stop
must be screened/shielded	serial cable
screened/shielded twisted pair	encoders, **CAN, RS485**
<2.9m (10 ft) long	analog inputs, system reset, **+5V, +12V, –12V, 0V**

13. MINT Programmer's Toolkit

The MINT Programmer's Toolkit is a set of PC applications and examples for use with SmartMove. The main component of the MINT Programmer's Toolkit is cTERM for Windows.

Installing *cTERM* on the PC.

If running Windows 95:

1. Insert disk 1 into the floppy drive
2. Select *Start* and then *Run*
3. Type **`A:\setup`** and follow the on screen instructions.
4. From the *MINT Programmer's Toolkit* group select *cTERM*.

If running Window 3.1:

1. Insert disk 1 into the floppy drive
2. Select *Program Manager* and then *Run*
3. Type **`A:\setup`** and follow the on screen instructions.
4. From the *MINT Programmer's Toolkit* group select *cTERM*.

13.1 Default cTERM Window

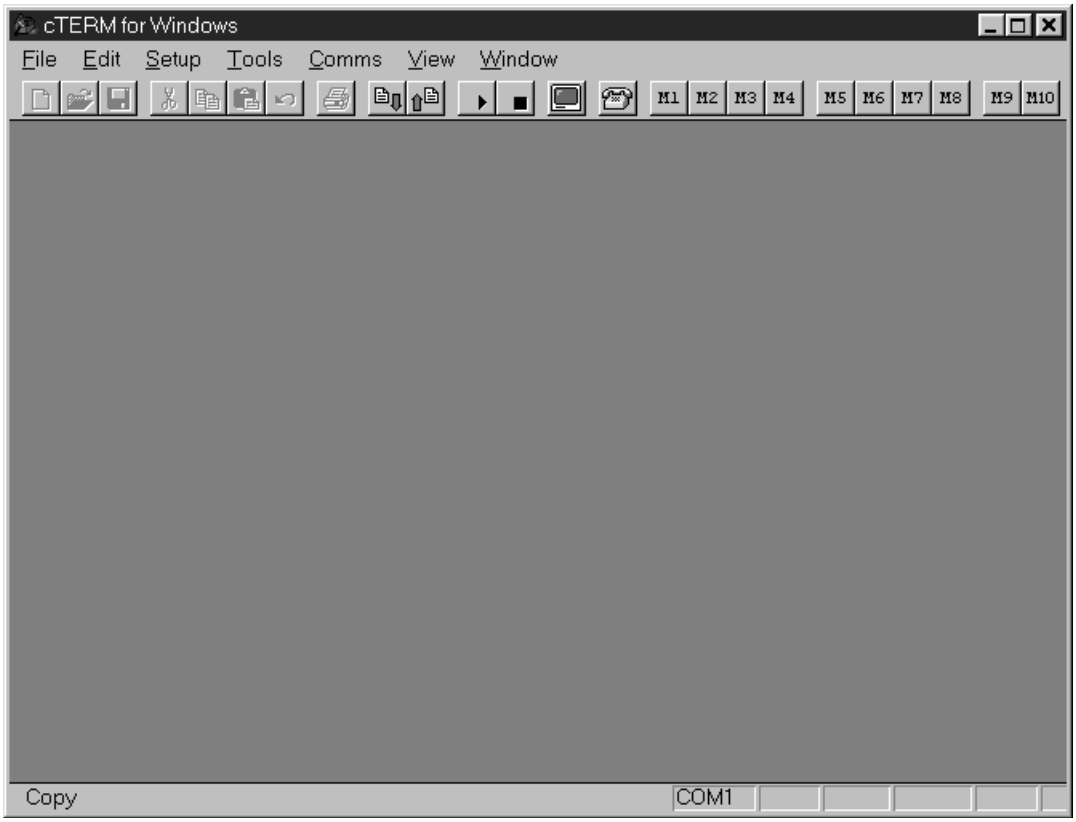


Figure 56: Default cTERM Window

13.2 Selecting a Controller and COM Port

The first time *cTERM for Windows* is used it will be configured for a *EuroSystem* (or member of the *EuroSystem* family, these being, *SmartMove*, *EuroStep* or *EuroServo*) on **COM1**.

13.2.1 Specifying a Controller

To select a controller:

1. Select the *Setup* menu and then choose *Controller*.

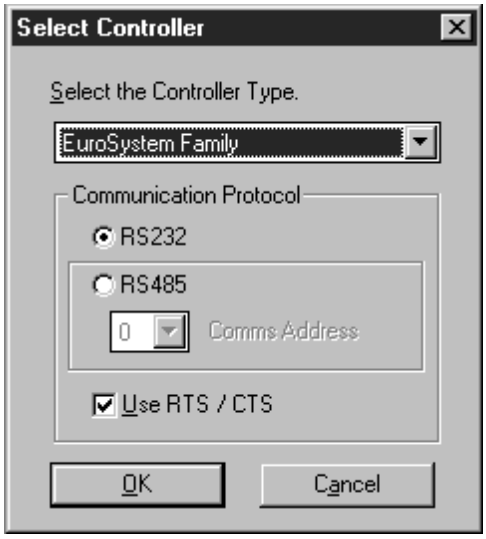


Figure 57: Set Controller Dialog Box

The *Select Controller* dialog box allows selection of:

- **Controller Type –** This drop-down menu allows the choice of:
  - *EuroSystem Family*
  - *NextMove BX*
  - *ServoNode 34, 50 or 51*
- **RS232 or RS485 –** The RS485 radio button should be selected if a multi-drop RS485 system is being used. Note that if RS485 RTS/CTS is being used, it will have no effect and should not be relied on to provide hardware handshaking.

  Comms Address (RS485 only) – This drop-down menu is available if RS485 is selected. If a multi drop system is being used, the controllers address should be entered here.
- **Use RTS/CTS –** This check box should be left enabled in most cases, except:
  - When using a *ServoNode 50*. These do not support RTS/CTS. Therefore *cTERM for Windows* will automatically clear the check box when *ServoNode 50* is selected.

- When using a *NextMove BX* with MINT version earlier than 3.8.
- If there is a problem updating the firmware with *ServoNode 34* or *51*. A problem may exist with RTS/CTS on some PCs and only shows while operating at the higher speeds involved in updating the firmware.
- If RS485 RTS/CTS is being used, it will have no effect. No hardware handshaking is available on RS485.

13.2.2 Setting up a COM Port

The *Setup COM Port* dialog box allows the serial port number and serial port speed to be specified.

To set-up a *COM port*:

1. From the *Setup* menu select *COM Port*.

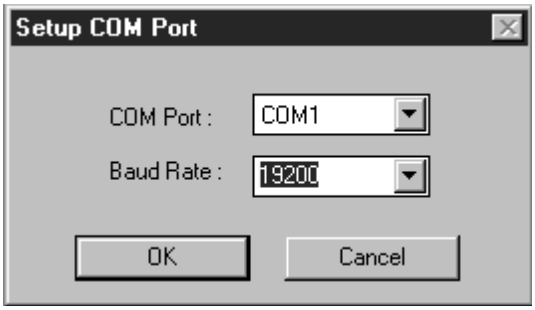


Figure 58: Setup Com Port Dialog Box

The *Setup Com Port* Dialog Box displays:

- **COM Port –** This drop-down menu allows selection of COM1 to COM4.
- **Baud Rate –** This drop-down menu allows selection of the baud rate. It is generally recommended that the highest baud rate supported by the controller is selected. That is:
  - **19200** on the *EuroSystem family* of controllers running MINT v2.67 or higher. Refer to the **BAUD** keyword in the MINT Programming Manual.
  - **9600** on the *EuroSystem family* of controllers running MINT version less than v2.67.
  - **9600** on *ServoNode 50*.
  - **38400** on *NextMove BX* (RS232) , 19200 on *NextMove BX* (RS485).

13.3 The Main Toolbar

Many of the tools in *cTERM for Windows* can be launched from the main toolbar. The tools are described in more detail later in the manual.

 **Download File:** if an editor is selected, the editor file will be downloaded. Otherwise a dialog box will prompt for the file to download.

 **Upload File:** if an editor is selected, the file will be uploaded to that editor. If an editor is not selected, a dialog box will appear, prompting an upload to an open editor. A new editor for uploading should be created, or 'save to file' initiated.

 Send MINT **`RUN`** command.

 Send *Comms abort* and MINT **BREAK `([Ctrl]+[E])`** command.

 **Terminal:** for communication with the MINT command line.

 **MINT Comms:** for communication with a running MINT program using the protected mode communications array.

 **to**  Execute macros 1 through to 10.

13.4 Terminal

The Terminal window provides access to the MINT command line.

To enable the Terminal Window:

1. From the *Tools* menu select *Terminal*, or
2. Click the  icon on the toolbar.

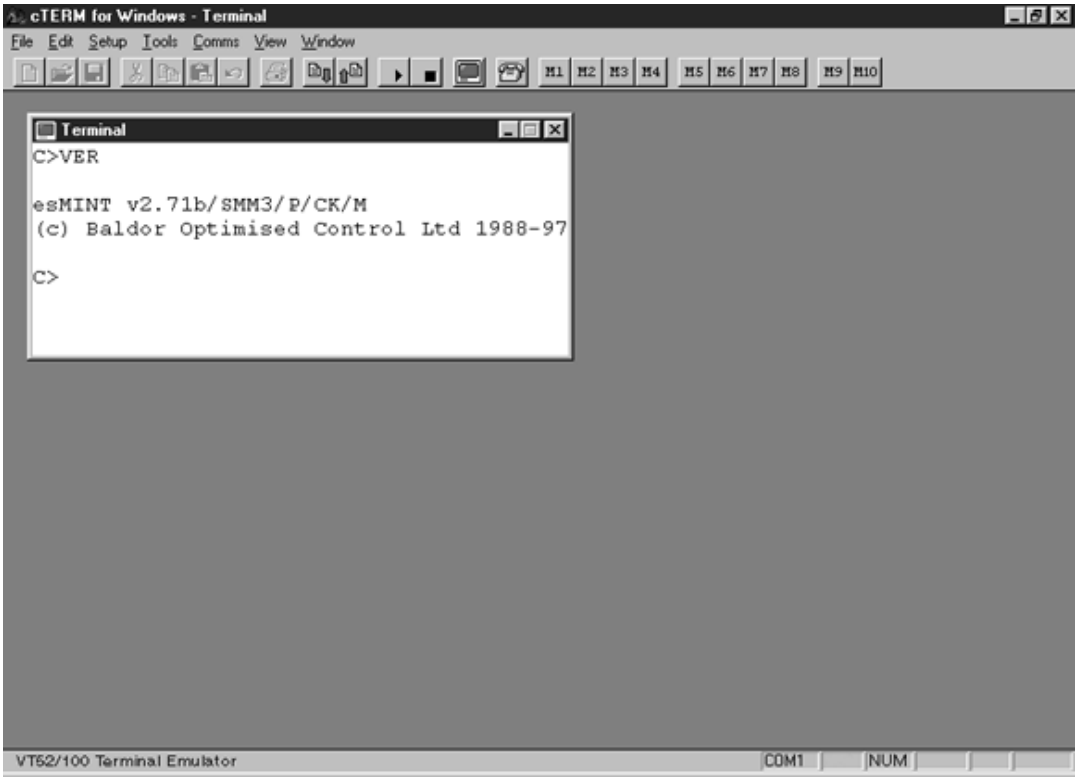


Figure 59: Terminal window

The terminal can be resized up to a maximum size of 50 lines by 80 columns either by dragging the border, or:

1. From the *Setup* menu select *Terminal*

The *Select Terminal Size* Dialog Box enables the terminal to be sized by entering values for **Lines (1-50)** or **Columns (10-80)**. Refer to Figure 60.

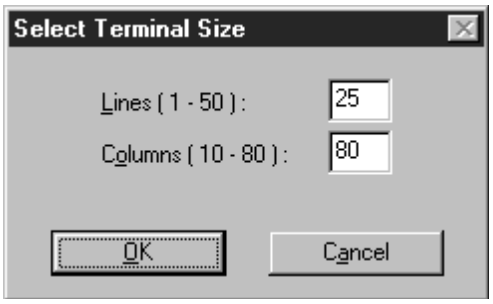


Figure 60: Select Terminal Size Dialog Box

13.5 Editor

A *cTERM* editor with *cut and paste* and *search and replace* facilities is available from the *File* menu. Either a new file can be selected or an existing file opened. Separate editors are supported for both the MINT *program* and MINT *Configuration files*.

There are several ways to initiate an editor:

1. From the *File* menu, select *New File*, and then choose from *Program, Config* or *Array*.
2. To open an existing file, from the *File* menu, select *Open File*, and then choose from *Program*, *Config* or *Array*. Directories may then be searched for the file.
3. To upload a file into an editor from the controller, from the *File* menu, select *Upload File*, or from the toolbar click .

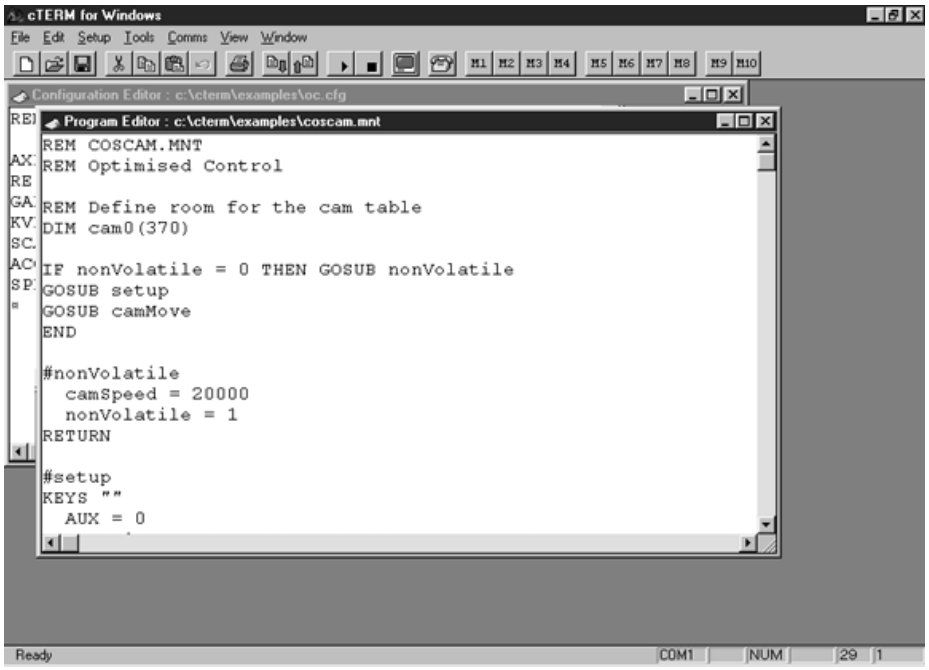


Figure 61: Editor Window

The standard Windows *Cut, Copy* and *Paste* functions are available both on the toolbar and in the *Edit* menu.

Other editor functions include:

- Ability to change the font settings from the *Setup* ⇒ *Font* menu.
- Downloading the files to the controller by clicking the icon on the button bar. By using the *Setup* ⇒ *Squash* menu, files can be compressed on download. Alternatively, selecting *Edit* ⇒ *Squash* will squash the current editor file to a new file. Full details of *Squash* can be found in section 13.10.
- Uploading the files from the controller into the current menu. *cTERM* will automatically upload the correct file from the controller depending upon which editor window is open.
- Ability for MINT files to be dragged from Windows *Program Manager* or *Explorer* onto the *cTERM* desktop. This will automatically open the editor.
- Printing from the editor using the *Print* icon on the tool bar. The file is printed to the currently installed Windows printer.

13.6 The Comms Window

The MINT Comms Protocol is supported through the use of the Comms Window, allowing the debugging of a MINT program. The Comms Protocol is discussed in [1].

To initiate the Comms Window:

- From the Comms menu, select Comms Window, or
- Click on the toolbar.

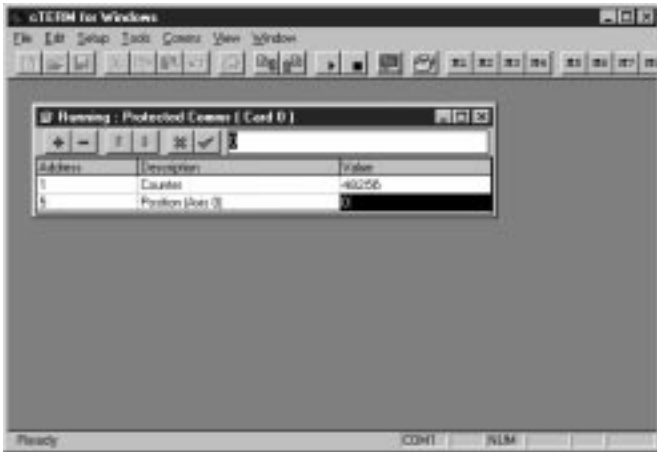


Figure 62: Comms Window

Locations are added by clicking the button, and removed by clicking the button. The **Address** field shows the *comms array index* and the **Description** field is free form text. The data within the **Value** field is then updated using a *timebase*.

It is possible to change the value of a comms location by clicking on the **Value** field and editing it in the **Edit** field. The button should be clicked to update the value and for it to be written back to the controller. Note that *Enter* will not write back the value. The button can be clicked to cancel the value.

Lines in the grid can be rearranged by selecting the required comms location to be moved. The buttons can then be clicked to move it up or down.

Notes:

1. The *Comms Window* polls the controller on a regular basis. This will slow the MINT program running on the controller as it will have to spend a high percentage of its time processing the comms requests. The more comms locations added to the window, the slower the MINT execution.

2. To pause the *Comms Window*, the *Terminal Window* should be selected. Only one *Terminal Window* and *Comms Window* can be active at any one time. Therefore, selecting one disables the other.
3. For a multi-drop RS485 system, the card address for individual controllers is selected by using either *Comms Address* from the *Comms* menu, or *Controller* from the *Setup* menu.

13.7 Project Files

cTERM for Windows uses Project Files to record the following information:

- Open editors.
- **COM** port settings.
- Locations displayed in the *Comms Window*.
- Terminal settings.

This is particularly useful if many elements have been added to the *Comms Window*. In order to save a project, the *File* ⇒ *Save Desktop As* menu option should be selected. The information is saved to a file with a **`.CTW`** extension. Once saved, the **`.CTW`** file can be added to the program manager as an icon in Windows 3.1, or to the Taskbar in Windows 95. Selecting the icon will start *cTERM for Windows* with the desktop settings. This can be used to create one icon for each **COM** port, for example. There are several ways to open a desktop:

- If *cTERM for Windows* is started normally, the most recent desktop is used.
- Create an icon for the desktop (as above).
- The name of the desktop file can be passed to *cTERM for Windows* as a parameter. For example:

  Command Line : **`c:\ctwin\ctwin.exe c:\ctwin\com1.ctw`**
- Drag the desktop file from the *File Manager* or *Windows Explorer* and drop it on to *cTERM for Windows*.

13.8 Capture/Graphing

cTERM for Windows allows a move to be set-up and plotted in order to assist in servo tuning. This uses the **CAPTURE** keyword implemented in MINT v2.65 or higher. As there can be a large amount of data to be transferred, it is recommended that the MINT **BAUD** keyword is used to set the baud rate to *19200* before using the capture facility.

To initiate *Capture*:

1. From the *Tools* menu, select *Capture*.

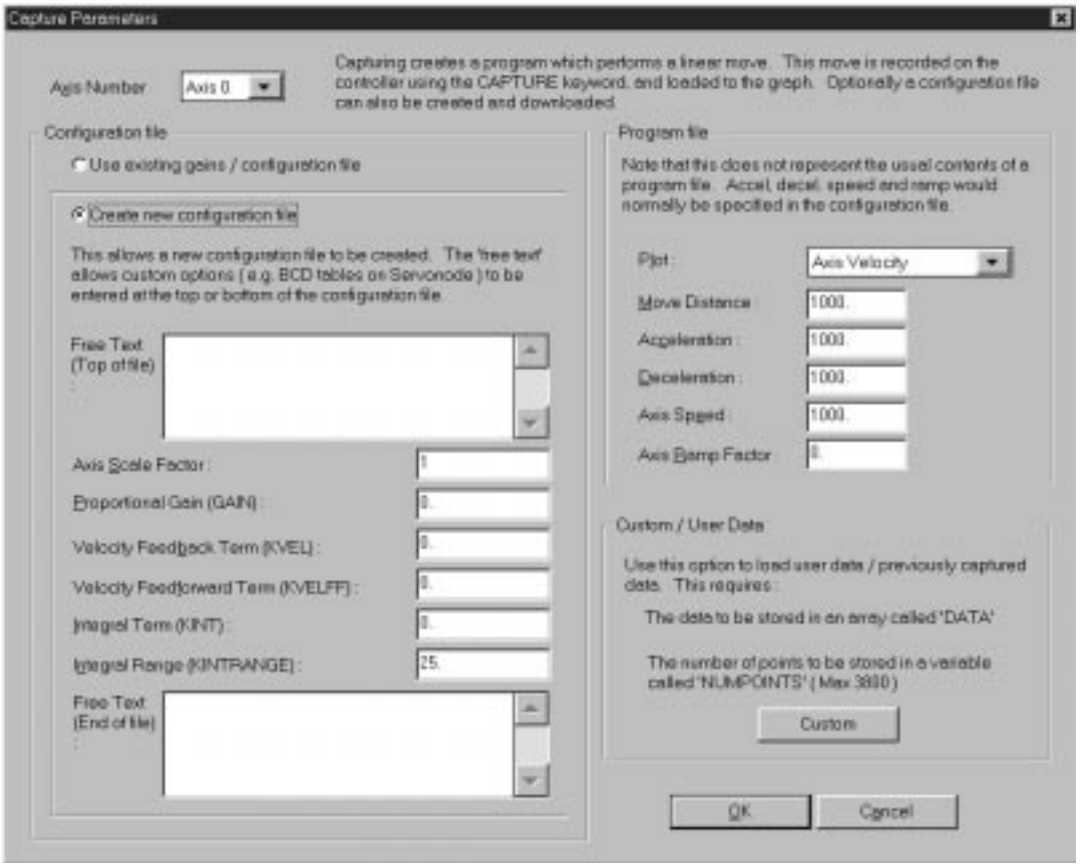


Figure 63: Capture Parameters Window

In order for *cTERM for Windows* to set-up and perform a move, the required move parameters in the *Capture Parameters* dialog should be selected.

Configuration file section

- **Use existing gains / configuration file –** Clicking this radio-button enables the current gains can to be used. With this option selected, the *Configuration file* will not be modified.
- **Create New Configuration file –** Clicking this radio-button allows *cTERM for Windows* to create a new *Configuration file.*

Program file section

It is also possible for *cTERM for Windows* to load and plot custom data on a graph. To accomplish this:

- The data should be stored on the controller in an array called **`data`**.
- A variable on the controller called **`numpoints`** should be created which holds the number of points to be plotted.
- The *Custom* button should be clicked to begin loading the data.

Example:

The following example MINT program performs a series of moves, storing all the data required for the graph.

```
DIM data(4000)
DATATIME = 8000
CAPTURE = 3
TIME = 0
MA = 25
GO
MA = 0
GO
MA = 25
GO
MA = 0
GO
PAUSE IDLE
NUMPOINTS = TIME/LT
```

Figure 64 shows the graph produced by this MINT program.

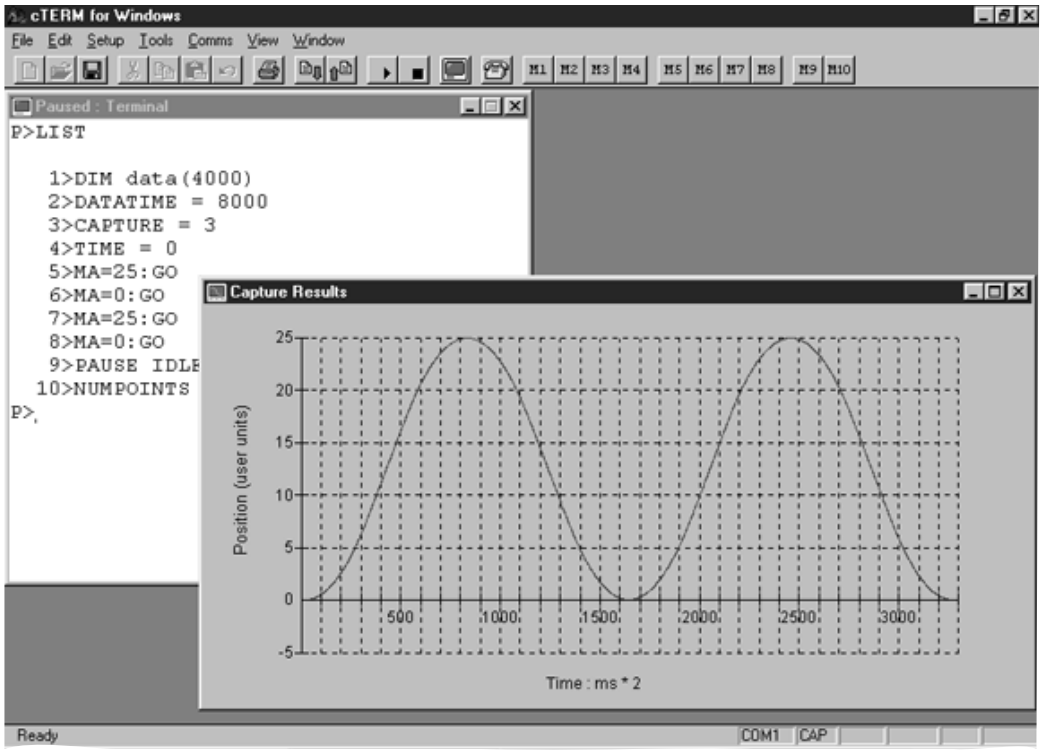


Figure 64: MINT Program Graph

Once the data has been loaded to the graph it can be saved by using *File* ⇒ *SaveAs* or by clicking the [ ] button. It will be saved as a *Comma Separated Variable* (**.CSV**) file, which can be loaded into most spreadsheets. It can also be printed using *File* ⇒ *Print* or by clicking the [ ] button.

13.9 Macros

Ten macro buttons are located on the right hand side of the cTERM for Windows toolbar. Each macro allows a text string to be sent to the controller.

To initiate the setup of Macro Entry:

1. From the Setup menu, select Macros.

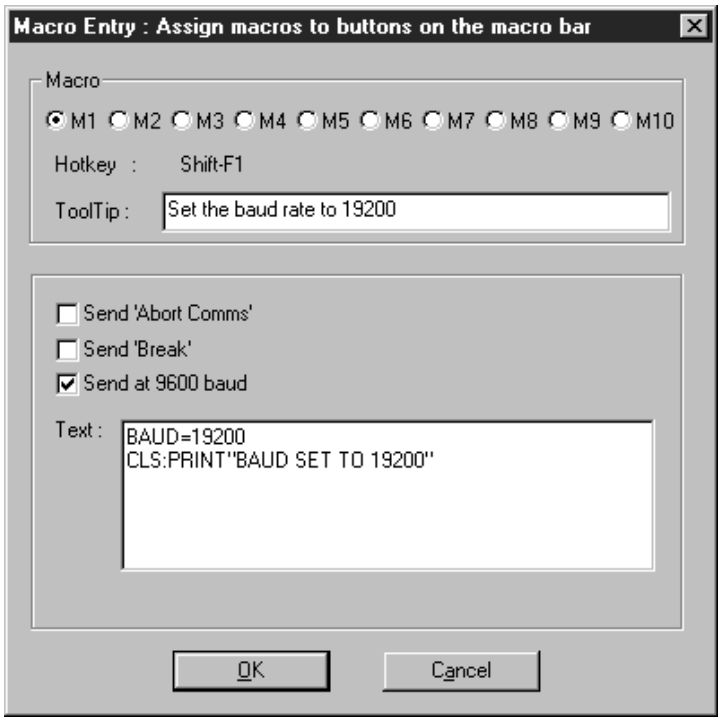


Figure 65: Macro Entry Window

The *Macro Entry* dialog box allows selection of:

- **M1 to M10 –** Click a macro number radio button to select the macro to edit. (The Hotkey field also shows how to achieve this with keyboard shortcut. This hotkey cannot be changed.)
- **ToolTip –** This field displays the text that will be shown when the mouse is placed over a button.
- **Send 'Abort Comms' –** This check box is used to abort protected mode communications (**`COMMSON`**) on the controller before sending the macro text.
- **Send 'Break' –** This check box is used to send the **BREAK** command (**`[Ctrl]+[E]`**) to the controller before sending the macro text.

- **Send at 9600 baud –** This check box is used to specify that the string be sent at 9600 baud. This is useful when using the macro to change that baud rate when the controller has been reset. Refer to the example above.
- **Text –** This field allows up to 255 characters of free text to be sent. The **Enter** key should be pressed at the end of the text if **Enter** is to be sent to the controller.

13.10 Squash

Squash allows a program to be compacted by removing redundant characters – blank lines for example. It is possible to save from an editor in the squashed format, or to have *cTERM* Squash each time it downloads a file.

Squash can perform several operations on a file. However, some of them can make the code unreadable. Therefore, the options are individually selectable.

To initiate the *Setup Squash Parameters* dialog box:

1. From the *Setup* menu, select *Macros*.
2. Click the *Parameters* tab.

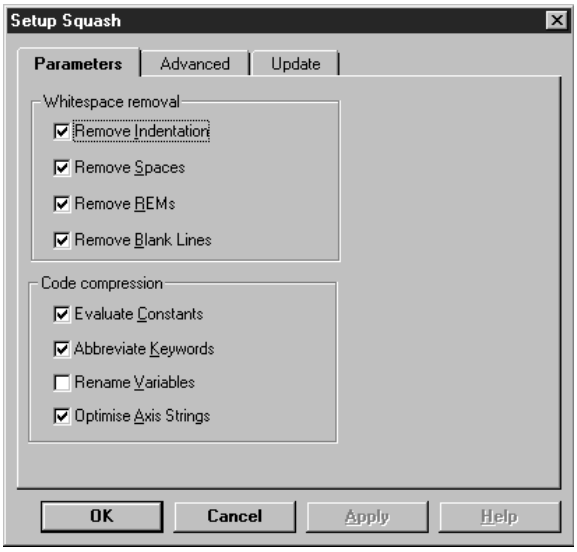


Figure 66: Setup Squash Window - Parameters

Whitespace removal section

- **Remove Indentation –** Clicking this check box removes spaces from the start of lines.

- **Remove Spaces –** Clicking this check box removes any unnecessary spaces other than at the start of a line.
- **Remove REMs –** Clicking this check box removes all **`REM`** statements.
- **Remove Blank Lines –** Clicking this check box removes any blank lines.

Code compression section

- **Evaluate Constants –** Clicking this check box replaces any MINT constants with their numeric values. For example, it replaces **`_servo`** with 1.
- **Abbreviate Keywords –** Clicking this check box replaces any longhand MINT keywords with their two or three letter forms.
- **Rename Variables –** Clicking this check box replaces any user variable names with single letter variable names. It will replace the first variable it finds with **`a`**, the second with **`b`** and so on. It is possible for *cTERM* to create a file of the variables it has renamed, and what it has renamed them with.
- **Optimize Axis Strings –** Clicking this check box will replace any single axis statements in the square bracket form with the dot form. For example: **`MA[0]`** will be squashed to **`MA.0`**

To initiate the *Setup Squash Advanced* dialog box:

1. From the *Setup* menu, select *Macros*.
2. Click the *Advanced* tab.

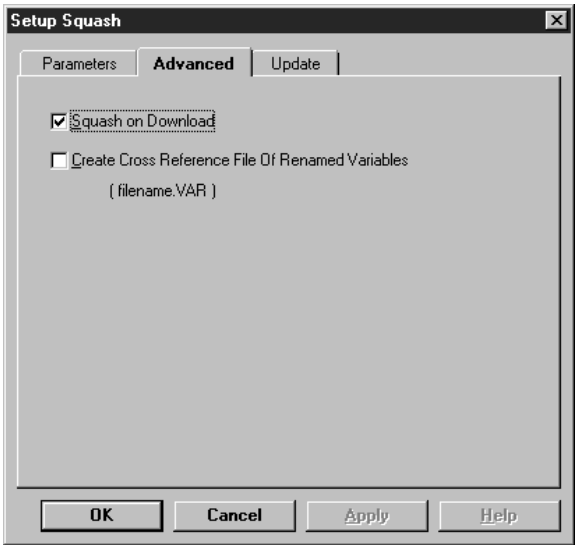


Figure 67: Setup Squash Window - Advanced

- **Squash on Download –** Clicking this check box Squashes each file as it is downloaded. This allows the file to be edited in the editor in its full 'readable' form, but will download a compact version of the code.
- **Create Cross Reference File Of Renamed Variables –** Clicking this check box creates a file specifying which variables have been replaced if the *Rename Variables* option was selected. The file will be stored in the same place as the program being squashed and will have a **.VAR** extension.

The *Update* tab is not currently implemented.

14. Trouble Shooting Guide

14.1 LED Status Display

The 7 segment status display on the front panel of *SmartMove* provides an indication of the move which is in progress and any error conditions. For two and three axis controllers, a flashing dot indicates an error condition.

Motion Status:

Display	Meaning
-	Servo power off
0	Servo powered up & idle
c	Cam profiling
c	Cam table (superscript)
C	Circular interpolation
E	Encoder following mode
F	Flying shear (no flashing dot)
H	Homing (datuming)
J	Jogging (velocity) mode
o	Offset mode
P	Linear positional move
q	Torque control mode
S	Stop asserted
U	Pulse following mode

Error Status:

Display	Meaning
E	External error, typically generated by the drive.
E	Software abort or interpreter error
F	Max following error exceeded
L	Limit switch open
o	Digital outputs short circuit or over current.

14.2 Trouble Shooting

Symptom	Check
Status display blank and controller not functioning.	• Check the power supply (18Vac or 24V dc) is connected and switched on.
Status display shows S.	• Check stop switch input is correctly wired and has power applied
Status display shows L.	• Check limit switch input is correctly wired has power applied
	• Check that opto-isolator supply (UP) is connected.
Cannot communicate with controller over RS232 port (cannot get **P>** or **C>** prompt by pressing return.)	• Verify that the terminal emulator program is loaded and set-up correctly.
	• Check that the terminal emulator program is configured for the correct serial port (**COM1** or **COM2**) and that the lead is plugged in both ends.
	• Confirm that a mouse driver or other serial device is not conflicting with *cTERM*
	• Check wiring for RS232 lead.
	• Check that there is not a program ready running on the controller (type **[Ctrl]+[E]** to abort the program).
	• Check that the MINT Comms Protocol is not running.
	• Check that the controller card is not an RS485 model. If a RS485 controller is used, confirm that the RS232 to RS485 converter is working correctly.
	• For RS485 devices, check that the correct card has been selected (using $).

Symptom	Check
Controller appears to be powered-up and working but will not cause motors to turn over.	• Check that the connections between motor and controller are correct. (Type **`TQ[0,1,2] = 100`**; to set all axis numbers to +10V and verify that this voltage appears on the **demand/command** input to the servo drive - remember to ensure that the motors will not cause any physical damage when doing this). • Ensure that the servo drive is enabled and working when the controller is not in error. (When the controller is first powered up the servo drives should be disabled if there is no program loaded for automatic execution (there is often an LED on the front of the drive to indicate status). Typing **`RESET`** at the **`C>`** prompt should cause the drives to be enabled.) • Check that *Configuration buffer* is loaded into controller and has been **`RUN`**, or that the servo loop gains are correctly set-up. (Type **`PRINT GN`** to verify that the controller proportional gain is no zero and refer to servo-system set-up).
Motor runs off uncontrollably when controller is switched on (Status display shows an 8).	• Check that encoders are connected to controller and are functioning correctly. (Use a dual trace oscilloscope to display both channels of the encoder simultaneously). • Check that servo drives are connected correctly to controller and that with zero demand there is 0V at the drive **demand/command** input. (Type **`SO[0,1,2]`** to set all **demand/command** outputs to 0V and verify that this voltage appears at the output from the controller. A voltage of +10V or -10V indicates that the controller analog output is damaged.) • Verify that the servo drives are correctly set-up and that the motor does not move with 0V on the **demand/command** input. • Verify that the controller and servo drive are correctly grounded to a common ground point.

Symptom	Check
Motor runs off uncontrollably when controller is switched on and servo loop gains are applied or a when move is set in progress, motor then stops after a short time and status display shows an F.	• Check that encoder 0 and **demand 0/command 0 D0** are connected to the same axes of motion; repeat for axis 1 and 2. • Check servo drive connection is correct with respect to polarity of servo drive demand/command. (Try swapping the **+demand/command+** and **-demand/-command** connections to the servo drive; note: this is not possible with some servo drives due to ground reference problems in which case you need to swap the encoder channels A and B.) • Check maximum following error is set to a reasonable value. (F indicates maximum following error exceeded; for setting up the maximum following error should be set to a high value. Type **`MF[0,1,2] = 16000`**; to set all axes to maximum following error of 16000 encoder counts.)
Motor is under control, but vibrates or overshoots during a move.	• Servo loop gains may be set too high. (Go through the servo system set-up procedure to establish correct gains.)

Symptom	Check
Motor is under control, but when moved to a position and back to start does not return to the same position.	• Check that the encoders channels A and B are clean signals and that they are correctly wired to the controller. (Use a dual trace oscilloscope to display both channels of the encoder at the controller back plane. Verify that when the motor turns, the two square wave signals are 90 degrees out of phase.) • Check that the encoder signal is free from electrical noise. Use the oscilloscope as above – verify that the complimentary outputs on the encoder (if fitted) are correctly wired. • If single ended encoders are fitted to the motors and the signals are noisy, try re-routing the encoder cables to avoid any source of electrical noise (notably the motor power leads). If this fails, the only solution may be to fit encoders with differential line driver outputs. • Ensure that the encoder leads use screened/shielded cable and that the screen/shield is attached to the screen connection on the encoder plug **at the controller end only.**) • Verify that the controller and servo drive are correctly grounded to a common ground point.

15. Product History

The history of the product is briefly summarized below:

Build Issue	Description
E202-2A	Product enters production on issue 2 PCB, operation at 16MHz. U5 is to E205-2.
E202-3A	Issue 3 PCB introduced. No major functional changes except to increase the hold-up time of the 18Vac input power supply unit from 4 to 8 ms
E202-3H	Modifications to allow operation of the CAN, U5 is to E205-3, expansion bus inoperative
E202-4A	Issue 4 PCB introduced. 16MHz operation on initial production. CAN and expansion bus operational. PALs E205-4 and E206-4 used. EMC improvements allow the product to be CE marked.

16. Bibliography

1. *MINT Programming Guide (for SmartMove, EuroSystem and SmartStep)*. Baldor Optimised Control 1998, document reference number: MN1260.
2. *CAN Peripherals Installation and Programming Manual*. Baldor Optimised Control 1998, document reference number: MN1255.

BALDOR ELECTRIC COMPANY
P. O. Box 2400
Fort Smith, AR 72902-2400
(501) 646-4711 Fax (501) 648-5792

MN1250

Printed in USA
6/98 Custom 10000