EuroSystem
EuroStep

Getting Started Guide

Issue: 2.0

Mint
THE MOTION LANGUAGE

Ref: MN00104-004;Start.doc/MC,EP/0497

Baldor Optimised Control Ltd.
178-180 Hotwell Road
Bristol
BS8 4RP
U.K.
Tel: (+44) (117) 987 3100
FAX: (+44) (117) 987 3101
BBS: (+44) (117) 987 3102

Technical Support
Tel: (+44) (117) 907 3470
E-mail: support.europe@optimised-control.com

Manual Revision History

Issue	Date	Reference	Comments
1.0	Mar 92	startes1/isg; M00104-001	First release of Getting Started Guide
1.1	Aug 92	startes1/mc; M00104-002	Manual revised for clarity and completeness. Include minimum wiring diagram.
1.2	Apr 96	start/mc; MN00104-003	Tidied up
2.0	Apr 97	start/mc,ep; MN00104-004	Added EuroStep and stepper revision history

1. Introduction

This manual covers getting up and running with the EuroSystem and EuroStep servo controller; it is essential to read parts two and three of this guide before opening the box and powering up the controller. This guide assumes nothing other than a very basic familiarity with PC's. It does however assume that you have a standard EuroSystem or EuroStep with RS232 serial port and MINT motion programming language installed. If you don't have this configuration, then 99% of this guide is still relevant and it is worth reading on. It also helps to have the operator keypad and interface board set-up in accordance with the relevant manual, although this is not essential. This guide mainly covers the application of servo motors, although many of the set-up procedures and the software is common to stepper motors.

1.1 How to use the Manual

The EuroSystem/EuroStep controller manual comes in several parts as follows:

1. This GETTING STARTED guide, covering the essentials of setting up the controller and also looking at some common applications.
2. The MINT Programming Guide, a reference manual which describes how to use the MINT motion programming language, a BASIC like operating language for motion control.
3. The HARDWARE GUIDE, giving details of wiring to the controller.
4. The cTERM and MINT Utilities manual, covering use of the enclosed diskette and terminal emulator program.
5. Release notes, covering upgrade information for the controller.

1.2 What you need to get Started

Before setting up the controller, make sure that you have the following items available:

1. This manual with the enclosed cTERM (Applications and Utilities) disk.
2. The controller, mounted in a rack with power supply.
3. The motor/amplifier combination that you intend to use.
4. An IBM PC or close compatible, running DOS2.0 or later with at least one free serial port.
5. A small screwdriver, soldering iron and some electrical cable.
6. An RS232 cable from your EuroSystem supplier, or the components to build one up.
7. The appropriate amplifier and encoder cables from your system supplier, or the connectors to build them up. (You require at least some 9 core screened cable and 9 Way 'D' type male plugs).

1.3 Key to Symbols used in this Guide

Throughout the Getting started guide various icons are used to indicate specific functions:

The screwdriver icon indicates that you need to make a physical connection to the controller by way of the screw termination's on the back plane.

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

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

1.4 Introduction to Servo Positioning Systems

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

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

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 repeated typically 500 times per second to ensure that the motor is always in the correct position. A typical closed loop control system is illustrated below:

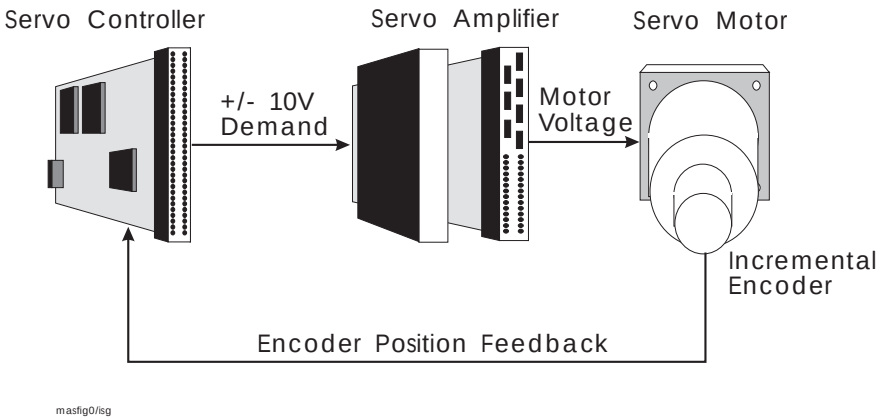


Figure 1: Typical closed loop positioning system

Setting up of a servo system is discussed in section 2.

1.4.1 Features of EuroSystem Controller

The EuroSystem controller is a programmable computer designed specifically for motion control of up to three axes of servo control or 3 axes of stepper control, integrated on one eurocard sized printed circuit board. The controller is supplied as a printed circuit board and is normally built up in a 19" rack with the EuroSystem back plane, a power supply and options such as the third axis servo board or operator keypad with LCD message display.

The major features of the controller are as follows:

- Three axis position controller for closed loop servo motors (two axes on board, additional axis using expansion card).
- Stand-alone operation or controlled by host computer over RS232/485 link, up to 16 cards on RS485 multi drop link
- Standard eurocard format (100 by 160mm) using surface mount technology, ideal for machine "design-in".
- Incremental encoder feedback, two channel plus index
- 12 bit analogue +/-10V servo amplifier outputs

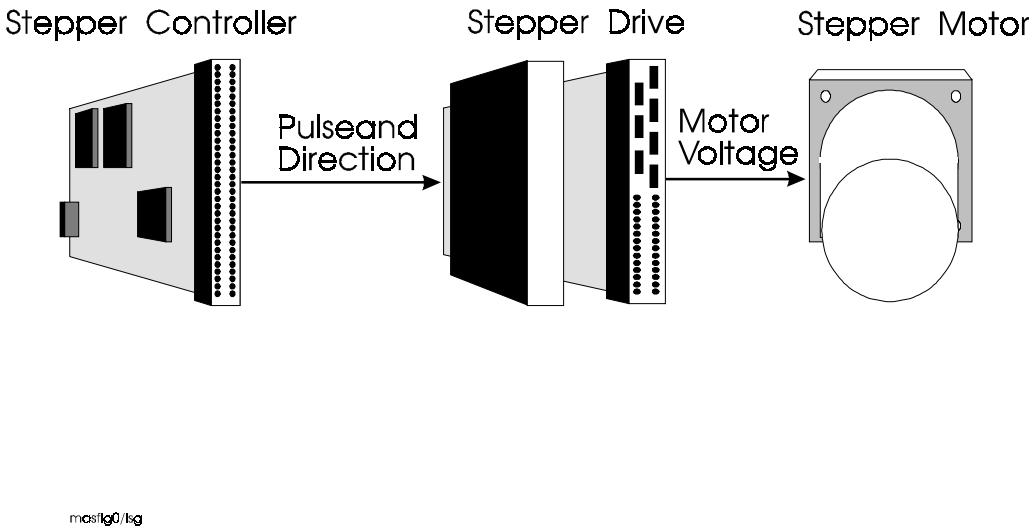
- Limit switch, Home switch and Error inputs, NPN with standard back plane (opto-isolated with opto-back plane).
- 8 uncommitted digital inputs and 8 outputs, NPN with standard back plane (opto-isolated with opto-back plane)
- Two 10 bit analog inputs
- Pulse/direction input for master/slave operations
- Option connection to LCD display and keypad interface for up to 64 keys in 8x8 matrix
- +5V, +/-12V power requirement for board, (+12-24V for opto-isolated I/O.)
- Back plane with plug-in screw terminal blocks, standard or opto-isolated
- Option board connector for expansion

1.5 Introduction to Stepper Positioning Systems

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

1. 1.Position controller - performs real time positional control of the motor(s), stores the application program and communicates with the user and other control equipment.
2. Stepper Drive - takes pulse and direction signals from the from the position controller and amplifies these low level signals to deliver the electrical power into the stepper motor
3. Stepper Motor - translates electrical power from the stepper drive into rotary or linear movement. The motor is not usually fitted with any form of feedback device, it is therefore an open loop system.

The controller works by providing a pulse and direction output for each axis of motion. The number of pulses correspond to the move distance and the pulse frequency correspond to the speed of each axis. These pulses are delivered with programmable acceleration and deceleration ramps to allow the system load to be moved without position loss. A typical open loop control system is illustrated below:



masfig0/lsg

Figure 2: Typical open loop positioning system

Setting up of a stepper system is discussed in section 3.

1.5.1 Features of EuroStep Controller

The EuroStep controller is a programmable computer designed specifically for motion control of up to three axes of stepper control, integrated on one eurocard sized printed circuit board. The controller is supplied as a printed circuit board and is normally built up in a 19" rack with the EuroSystem back plane, a power supply and options such as the operator keypad with LCD message display.

The major features of the controller are as follows:

- Three axis position controller for open loop stepper motors
- Stand-alone operation or controlled by host computer over RS232/485 link, up to 16 cards on RS485 multi drop link
- Standard eurocard format (100 by 160mm) using surface mount technology, ideal for machine "design-in".
- Limit switch, Home switch and Error inputs, NPN with standard back plane (opto-isolated with opto-back plane).
- 8 uncommitted digital inputs and 8 outputs, NPN with standard back plane (opto-isolated with opto-back plane)
- Two 10 bit analog inputs
- Pulse/direction input for master/slave operations
- Option connection to LCD display and keypad interface for up to 64 keys in 8x8 matrix
- +5V, +/-12V power requirement for board, (+12-24V for opto-isolated I/O.)
- Back plane with plug-in screw terminal blocks, standard or opto-isolated
- Option board connector for expansion

1.5.2 Programming Features MINT

MINT™ is the standard motion control firmware installed on the EuroSystem servo controller. An introduction to the features of the language is given in chapter 3 of this getting started guide and in the MINT Programming Guide.

- Easy to use Basic-like motion control language
- Two axis circular interpolation on axes 0,1
- Three axis linear interpolation
- Master/slave operation with software gearbox (EuroSystem only)
- Infeed and phasing machine control (EuroSystem only)
- 28K bytes non-volatile memory for storing programs and data
- On board program editor using terminal or IBM PC

1.5.3 cTERM

cTERM (available in both DOS and Windows formats) is a powerful terminal emulator program supplied specifically for use with the EuroSystem controller:

- Terminal emulator program for IBM PCs and compatibles.
- Automatically sets up serial port for controller
- Easy to use 'pop-down' menu structure
- File up-load and down-load
- Powerful debugging facilities for controller communications

- Full screen program editor supplied - user configurable so that your favourite editor can be loaded from within cTERM

1.6 Firmware Options

The controller is normally shipped with the Interpolation MINT basic motion programming firmware installed. **This start-up guide assumes that the controller is fitted with the standard MINT software.** However, for special applications, there are other variants of firmware which support particular special functions. Details are provided

2. Servo System Set Up

The following is a step by step guide to setting up a two axis servo system, it is not applicable to stepper systems, please see section 3.

It is worthwhile but not essential to familiarise yourself with the MINT programming language and the editor before starting the set-up procedure. A summary of the use of MINT is given in section three of this start-up guide, and covered in more detail in the MINT programmers reference guide.

2.1 Minimum System Wiring

The controller will work with many servo systems, both electric and hydraulic, as long as they take a +/-10V input as a velocity or torque demand and provide feedback in the form of an incremental encoder. (Absolute encoders may also be used with the addition of an option board).

A minimum system is one where the controller and drive are configured to work with as little external wiring as possible. This step-by-step example covers setting-up systems with two axes of motion (amplifiers and motors). Connections to the controller are normally made via the controller back plane. Paragraphs instructing you to make a connection are marked with the screwdriver icon:

Minimum configuration for a 2 axis servo system.

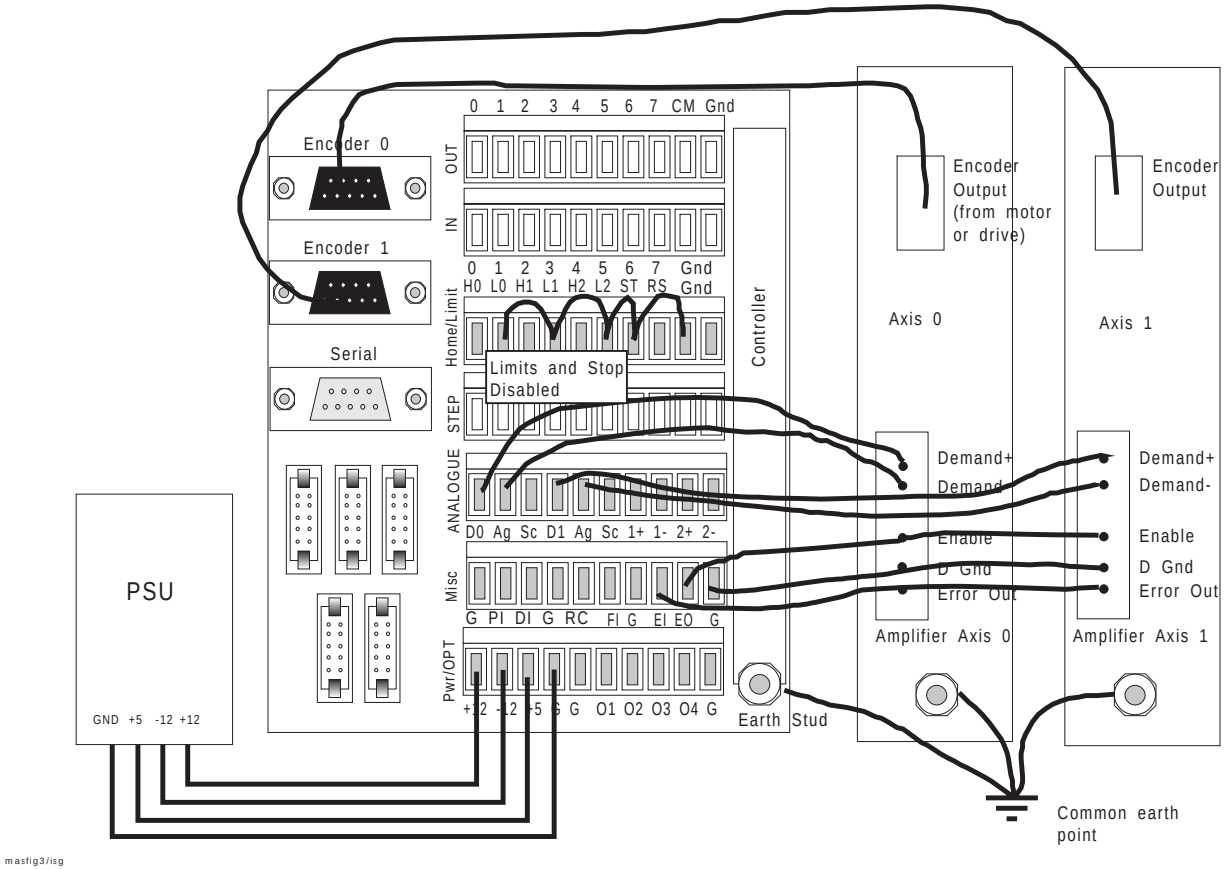
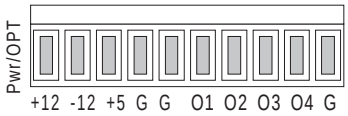


Figure 3: Minimum 2 axis configuration

Please note that the following wiring instructions relate to the standard non-isolated controller back plane only - the procedure for setting up the isolated back plane is similar - but different voltages etc. are required as detailed in the hardware guide.

2.1.1 Power Connections: "Pwr/Opt" Screw Connector

The controller power supply requirements are: +5V @ 0.5A (labelled "+5" on the connector), +12V @ 100mA "+12" and -12V @ 100mA "-12" and ground "G".



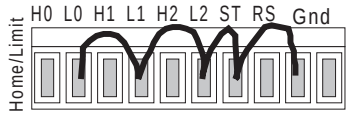
If not a rack system with internal power supply already fitted, connect +12V,-12V,+5V and G (0V) on the "Pwr/Misc" connector to a stabilised three rail power supply with at least 1A capacity on the 5V line.

2.1.2 Limit and Stop Switches: "Home/Limit" Connector

The limit switch inputs are connected through normally closed over-travel switches on the end of the axes of motion and cause motion to stop when floating. The inputs must be grounded to operate; usually through normally closed switches on the axes and guards (if two limit switches on one axis are used, they must be connected in series). This arrangement means that the controller is fail safe, if a wire breaks, the controller stops motion.

The Stop switch input causes all axes to decelerate to a halt and is used for connection to machine guards. This also must br grounded in order to allow motion.

For the purposes of setting up the controller, these inputs can be linked out on the back plane:



Remove the screw connector labelled "Home/Limit" and connect L0 (limit axis 0), L1, L2, ST (stop motion input) to Gnd. Use insulated thin gauge wire for the connections.

If the limit or stop inputs are not properly connected, the controller will display an "L" or an "S" on the status display on the front of the board on power up.

2.1.3 Amplifier Demand Signals: "ANALOGUE" Connector

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

Before proceeding further with the set-up procedure, please ensure the amplifiers have been correctly set-up for the motor in accordance with the instructions in your amplifier manual.

On the "ANALOGUE" screw connector, connect "D0" (+/-10V demand output 0) to the demand+ input on the first amplifier and "Ag" (analog ground) to either the demand- input on the first amplifier or the analog ground terminal on the amplifier. This connects the analog demand output form the controller to the analog input (speed or torque demand) on the motor amplifier. It is best to make these connections using screened twisted pair cable (especially when the amplifier is distant from the controller), the cable screen should be connected to the Sc input on the controller and left unconnected at the other end. Repeat for the second amplifier (if fitted) using "D1".

2.1.4 Amplifier Enable

On the "MISC" screw connector, connect the enable input on the amplifier to the enable output from the controller. The enable output from the controller is configurable by jumpers so that it is active high or active low.

Connect the common terminal "EO" to the enable input and the "G" to the amplifier ground.

It is essential that the amplifiers are disabled during power-up of the controller or when a controller error occurs. This ensures that the amplifiers are only able to drive the motors when the controller is performing servo-loop closure. This involves changing the jumpers on the controller JP4 JP5 and JP6 to set the correct sense of the enable output - see the HARDWARE REFERENCE GUIDE for further information.

On power up, the amplifiers should be disabled. You may find that by issuing the RESET command the amplifiers are disabled. If this is the case then the error output is the wrong sense.

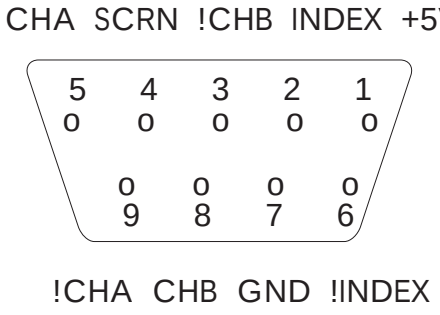
2.1.5 Encoder Connections

The encoders are the position sensors used by the controller to measure axis position. They 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 controller will work with three channel incremental encoders (CHA, CHB, 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 independently and that the screen is connected at the controller end only. Maximum cable length is dependent on the encoder specification, but should be kept as short as possible.

Tip**: In many brushless amplifiers, the motor is fitted with a resolver and the encoder signal is synthesised by the amplifier. In these instances, the encoder connections are wired to the amplifier rather than the motor. **When making connections to encoder outputs from a brushless drive, do not connect the +5V supplies on the controller and amplifier together since this may cause noise problems.

The encoder should be wired to a 9 pin 'D' male plug, using good quality multi-core screened cable, according to the following diagram. If the encoder is a single-ended type (i.e. no compliment 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.



masfig4/isg

Pin No.	Signal Name	Function	Type
1	+5V	Power to Encoder	Output
2	INDEX	Index Mark	Input
3	!CHB	Channel B Compliment	Input
4	SCRN	Cable Screen	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

2.1.6 Ground Connections

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

Connect the earth stud on the controller and amplifiers to a common earth point using heavy duty cable. Make sure that the connection is in a 'star' configuration as shown in the minimum system wiring diagram.

2.1.7 Serial Cable

The RS232 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, but required for programming. Use a standard serial cable from your system supplier or build one up according to the wiring diagram in the Hardware manual.

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.

2.2 Testing System Wiring

In order to check that the encoder, tacho (if connected) and motor are wired up correctly, it is recommended that the motor is tested and commissioned 'on the bench' and not in situ.

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 sense.
2. Check encoders work.
3. Check the amplifier is working.
4. Check encoder and motor are connected the right way round.
5. Set-up system gains for satisfactory closed loop control.

The program examples in this manual are contained on the cTERM disk that is enclosed with the manual. The disk icon is used to indicate a program (each program is enclosed in a box) the name under the disk icon indicates the name of the program on the disk. These programs can be modified by the use of an external editor or by use of the on-board editor on the controller. Typing RUN will execute the programs. Single line examples can entered direct at the command line for immediate execution, indicated by the prompt icon: P>

2.2.1 Starting cTERM

The cTERM disk contains a program for IBM compatibles that makes the computer work like a terminal emulator. A terminal emulator program accepts character input from the computer keyboard and sends these characters down the serial port so that they can be interpreted and processed by the controller microprocessor. The terminal emulator program displays any information that is sent back from the controller on the computer screen. Therefore, you use the terminal emulator program to 'talk' to the controller. For example, if you press the "A" key on the keyboard, an "A" character is sent down the serial port and the controller responds by sending the same character back which appears on the computer screen. Of course, it all happens so quickly that you don't notice any delay between typing the character and it appearing on the screen.

The terminal emulator therefore allows you to create programs in the memory of the controller by typing them into an IBM compatible computer, using the editor on the controller.

A number of other features are available within the cTERM program, for instance, the ability to use your favourite editor on the computer to create programs for the controller and then download these at the press of a button. These features are covered in the cTERM manual.

To run cTERM, insert the disk in your 3.5" floppy disk drive slot (drive A: or B:). Log-on to that drive by typing "A:" or "B:" followed by return, then type "cterm" and press return.

Details of using cTERM are given in the manual cTERM and MINT Utilities. Please familiarise yourself with operation of cTERM, especially up and downloading of files, by reading section two of this guide before you proceed further.

Experienced computer users who have a fixed disc drive may wish to copy all the files on the cTERM disk into a directory their fixed disk.

Next plug your serial cable into COM1 in the back of the computer, and into the 'D' type marked "Serial Port" on the controller back plane, or into the front of the controller (which is a duplicate of the connections on the back plane).

Plug the power lead into the euro-socket on the back of the control system rack and switch-on. The controller should power-up and the message:

```
MINT vX.XX
C>
```

should be displayed on the computer screen and also on the LCD display in the rack (if fitted) If this is not the case, check the wiring of the serial cable and refer to the Trouble-Shooting section in the back of this guide.

2.2.2 Example Programs

Throughout this section example programs are given, identified by the disk icon: 🖫

Under the disk icon, the file name is given and this file can be found on the cTERM disk in the directory \START, for you to download to the controller. Some files names have ".MNT" after the name which indicates that they are MINT program files and should be downloaded to the program storage space on the controller. Others, have ".CFG" after the filename, which indicates that they are configuration files and should be downloaded to the controller configuration storage space. Details of how to download are given in the cTERM manual, more information on configuration and program files is given in section 3 of this guide.

MINT has a integral editor than can be used to make simple changes to program and configuration files. **Details of using the on-board editor is given in section 10 of the MINT Programming Guide, it would be worthwhile to spend a few minutes familiarising yourself with operation of the editor before proceeding.**

2.2.3 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;
ABORT   : REM Disable the amplifier
LOOP
  PRINT POS[0];POS[1]; : BOL
ENDL
```

To download the program to the controller, load cTERM for DOS*, press F3 and type the program name: "encoder" in the PROGRAM box; then press return. cTERM should indicate that the program has downloaded successfully or will display an error message in the event of a fault. Press F10 to go into the terminal screen and type "RUN" (press return) this should start execution of the program.

The program works by setting all system gains to zero and disabling the amplifiers so that the motor shaft can be moved by hand. If the amplifier is not disabled, check the connection of the enable output from the controller. Alternatively, disconnect the amplifier ensuring that the encoder is still powered up.

* See cTERM and MNT Utilities manual for details on using cTERM for Windows.

The program will print the position (encoder value) of axes 0 and 1 to the terminal. By moving the motor backwards and forwards you should see the position change. Note 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 anti-clockwise. If the position does not change then check the following:

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

Make a note of which direction gives an increase in position. Repeat for other axes.

To check the index (marker) pulse, see a later section.

2.2.4 Checking Motor Polarity

To confirm the polarity of the motor connections, use the TORQUE command as shown (refer to the MINT Programming Guide for further details on the TORQUE command):

```
P>        TQ = 1
```

Starting with a value of 1, increase the torque value until the motor starts to move. The motor should move in a positive direction i.e. the position (encoder) should increase. Downloading and running the following program can be used to check this:

MOTOR.MNT

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

Tip: you will need to use MINTs editor to change line 3 of this program. From the "P>" you can do this by typing:

```
P>        EDIT 3
```

After making changes to the line, press return to finish. If the "C>" prompt is displayed, you can change to the program file by typing :

```
C>        PROG
```

If the position decreases, swap the motor cables over or swap the A and B channels on the encoder, first removing power from the amplifier. Repeat the operation.

Repeat the operation with a negative value of torque, for example:

```
P>        TQ = -1
```

The position should now decrease. Repeat the operation for the other axes.

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

- The amplifier is enabled.
- The motor is connected.
- The amplifier is correctly configured.

➥ The controller is correctly connected to the amplifier

More suggestions are given in the fault finding guide at the back of this manual.

2.3 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 minimise the error between demand and actual position measured with an incremental Encoder. Every 1 or 2 milliseconds the controller compares desired and actual positions and calculates the correct demand for the motor. The torque is calculated by a PIVF (Proportional, Integral, Velocity Feedback and Velocity Feed forward) algorithm.

Control could be achieved by applying a torque proportional to the error alone, but this is a rather simplistic approach. Imagine 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 resultant to the motor via an amplifier. 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 onset of instability a damping term is incorporated in the servo loop algorithm, called Velocity feedback gain. 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 amplifier, called a velocity servo). The effect of too high proportional gain, or too low velocity feedback gain is illustrated by the "Underdamped" line in the figure below:

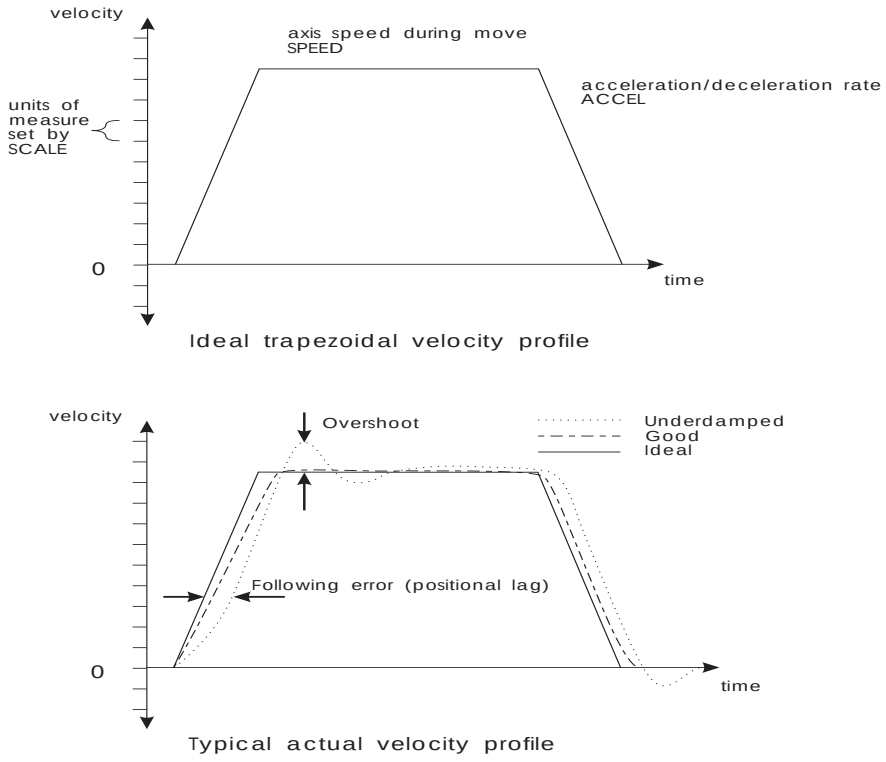


Figure 4: Move response

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 speed 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.

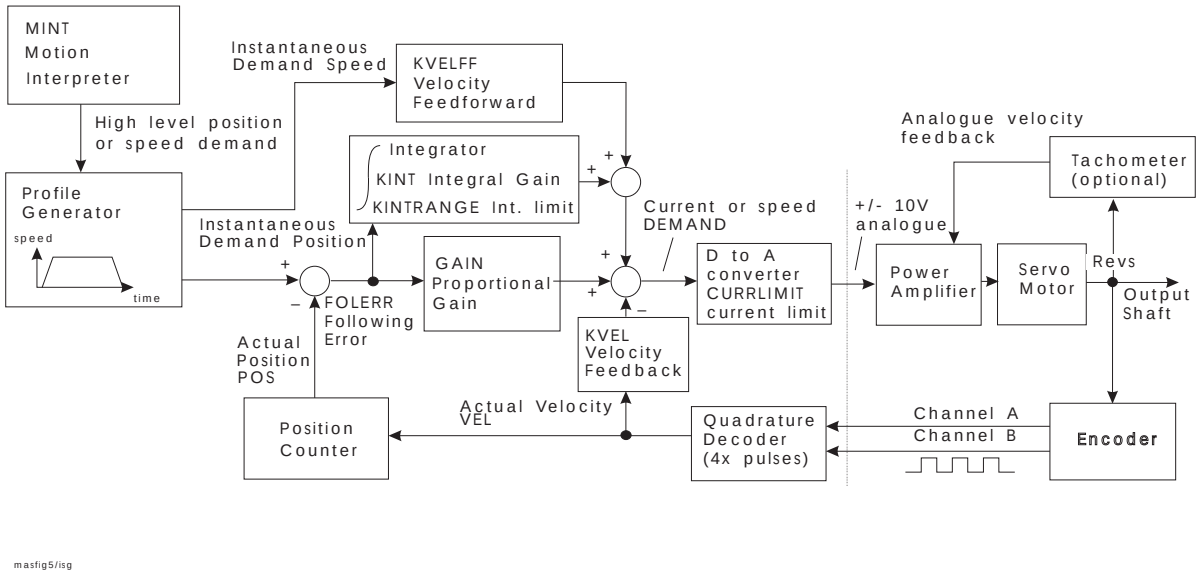
The final term in the control loop is Velocity feed forward. This is useful for increasing the response and reducing the following error.

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

- Current or torque amplifiers use the demand signal to control the current flowing in the motor armature and hence the torque of the motor.
- Velocity controlled amplifiers (velocity servo) uses the demand signal as a servo speed reference.

For general purpose applications, the torque amplifier is cheaper and simpler to set up, but the velocity servo gives better control, especially in high performance applications. For torque amplifiers, velocity feedback must be used to stabilise the system, but 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, amplifier, motor and gearbox is presented below. The servo amplifier may be a simple current amplifier, or incorporate internal velocity feedback via a tachometer:



masfig5/isg

Figure 5: Servo loop block diagram showing relevant MINT keywords in capitals

You see that we have a four term controller incorporating proportional, velocity feedback/feed forward and integral gains.

The equation of the loop closure algorithm is as follows:

$$\mathbf{Demand = GN.e - KV.v + KF.V + KI.\Sigma e}$$

e - following error (quad counts)
v - actual axis velocity (quad counts/sample time)
V - demand axis velocity (quad counts/sample time)

Keyword	Abbreviation	Description
GAIN	GN	Proportional servo loop gain
KVEL	KV	Velocity feedback gain
KVELFF	KF	Velocity feed forward 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 your 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 system configuration file.

Two other keywords, KINTRANGE and CURRLIMIT, are used to control the demand 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 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 amplifiers, determines the maximum value of the demand output as a percentage of the full scale demand. Therefore if CURRLIMIT = 50, the maximum demand output will be ±5V.

The encoder gain (measured in pulses/rev) is one factor that is hardware dependant 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 (See the MINT Programming Guide for details of scaled integers).

All servo loop parameters default to zero, so the motor will have no power applied to it on power up. Most servo amplifier can be set up in either current (torque) control mode or velocity control mode. The procedure for setting system gains differs slightly for each.

2.3.1 Setting System Gains for Current Control

After you have confirmed that the encoder and motor are correctly wired up. You should start by applying some feedback gain, KV. Start with a value of 1 and increase it until you feel some resistance in the motor. For example:

```
P>      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, apply some proportional gain, GN. Start off with a value which is a quarter of the feedback gain i.e. GN = KV / 4. If the motor starts to vibrate, increase the velocity feedback gain (damping), KV, or decrease the proportional gain, GN. Increase proportional gain, GN, until the motor shaft becomes stiff.

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

```
P>      KF = KV
```

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

```
P>      JOG = 1000
```

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

```
P>      JOG = -1000
```

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

By using the statement:

```
P>       PRINT FE
```

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

Try increasing the JOG speed to say 5000, 10000, 15000 and 20000 and check the following error (FE). The program below can be used to check the following error and demand output from the controller:

FOLLERR.MNT

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

This can also be used to determine the maximum speed of the motors without a load. An error will be generated if the motor can not keep up because the jog speed is too high. This is signified by an 'F' on the 7 segment status display. It may be necessary to increase/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:

```
P>       TQ = 20
P>       ? 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, you should ensure that the amplifier output is within its continuous rating to avoid over current. For many amplifiers, this is 50% of the peak output, therefore a controller demand of 50 (see the DEMAND keyword) will keep the amplifier within its continuous rating.

2.3.2 Fine Tuning System Gains

The above 'rules of thumb' for setting system gains, while adequate to get the system moving, will not provide the optimum response without further fine tuning of the system gains. The basic aim here is to set the Proportional Gain as high as possible without getting overshoot or instability or hunting (buzzing) on an encoder edge when stationary.

This is best achieved by attempting some short positional moves (say 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. However, EuroSystem has the capability to output the instantaneous velocity from a +/-10V tuning output on the controller. See the AUX keyword in the MINT Programming Guide for further details of this.

If you do not have an oscilloscope, the following program can be used to print the axis speed and following error to the terminal screen during a position (relative move):

MOVE1.MNT

```
REM Program file for trial moves

AXES[0]         REM Only axes 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 = 4
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 and the system, if well behaved, should have a relatively low error during the constant speed part of the move.

Try altering the gains, the acceleration and speed 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.

2.3.3 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 amplifier which may not be enough 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 which is 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.

2.3.4 System Gains for Velocity Control

Velocity controlled drives incorporate the velocity feedback term in the amplifier 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:

```
P>       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 tacho output for fast point-to-point moves. This enables you to see the velocity/time profile for the motor in order to monitor actual acceleration and overshoot.

Referring to the servo loop block diagram on Page 16, 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, a very 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 of ω RPM from the profile generator results in a demand output to the velocity drive which gives ω RPM on the motor shaft (i.e. 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 doing anything except compensating 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, you need to consider the workings of the servo loop closure algorithms. 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 - see the MINT programming guide for further information).

In this example the velocity of the servo is 3000RPM with a +10V input, and the encoder has 500 counts per revolution.

At 3000RPM we require an analog voltage of +10V. This relates to:

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

Now we can calculate the number of quadrature counts per loop closure time 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.

$$\frac{50 * 500 * 4}{500} = 200 \text{ quadrature counts per servo loop closure time}$$

(Note: for 1 mS loop closure time, this expression becomes: 50*500*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:

$$\text{KVELFF} = \frac{2048}{200} = 10.24$$

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.

You can investigate the effect of velocity feed forward gain by jogging the motor at constant speed and printing out the following error, FE, for different values of KF. When attempting this make sure that you have 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.

2.3.5 The Configuration File

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

Whenever the program is RUN, the configuration file is first executed. A list of parameters that you may need to set up follows:

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

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

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;
KF = KVEL; REM assuming a torque amplifier

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
MFOLERR = 1;   : REM 1 rev maximum following error
```

In the above example, the default axis list is 0,1,2 the semicolon is used to apply all the parameters to all three axes, see the MINT Programming Guide 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 you to determine the best values for your system.

To download the example configuration file type F3 from within the cTERM terminal emulator and select "CONFIG", then type the example configuration file name: CONFIG. You can then use the MINT editor to change the values to those that you have identified during the set-up procedure.

2.4 Encoder Marker Pulse

With the amplifiers configured with the correct gains, it is now possible to check the marker pulse on the encoder. This is achieved using the MINT HOME command as follows:

```
P>        HOME = 6
```

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

Using the command:

```
P>        HOME = 4
```

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

3. Stepper System Set Up

The following is a step by step guide to setting up a two axis stepper system on EuroSystem or EuroStep controllers.

It is worthwhile but not essential to familiarise yourself with the MINT programming language and the editor before starting the set-up procedure. A summary of the use of MINT is given in section three of this start-up guide, and covered in more detail in the MINT programmers reference guide.

3.1 Minimum System Wiring

The controller will work with many stepper motor drives. Outputs are open collector, 100 mA sink with a frequency range of 10Hz to 200KHz. In addition there is a boost output provided for each axis which is used to signal a short term increase in drive current when used with drives which have this feature.

A minimum system is one where the controller and drive are configured to work with as little external wiring as possible. This step-by-step example covers setting-up systems with two axes of motion (amplifiers and motors). Connections to the controller are normally made via the controller back plane. Paragraphs instructing you to make a connection are marked with the screwdriver icon:

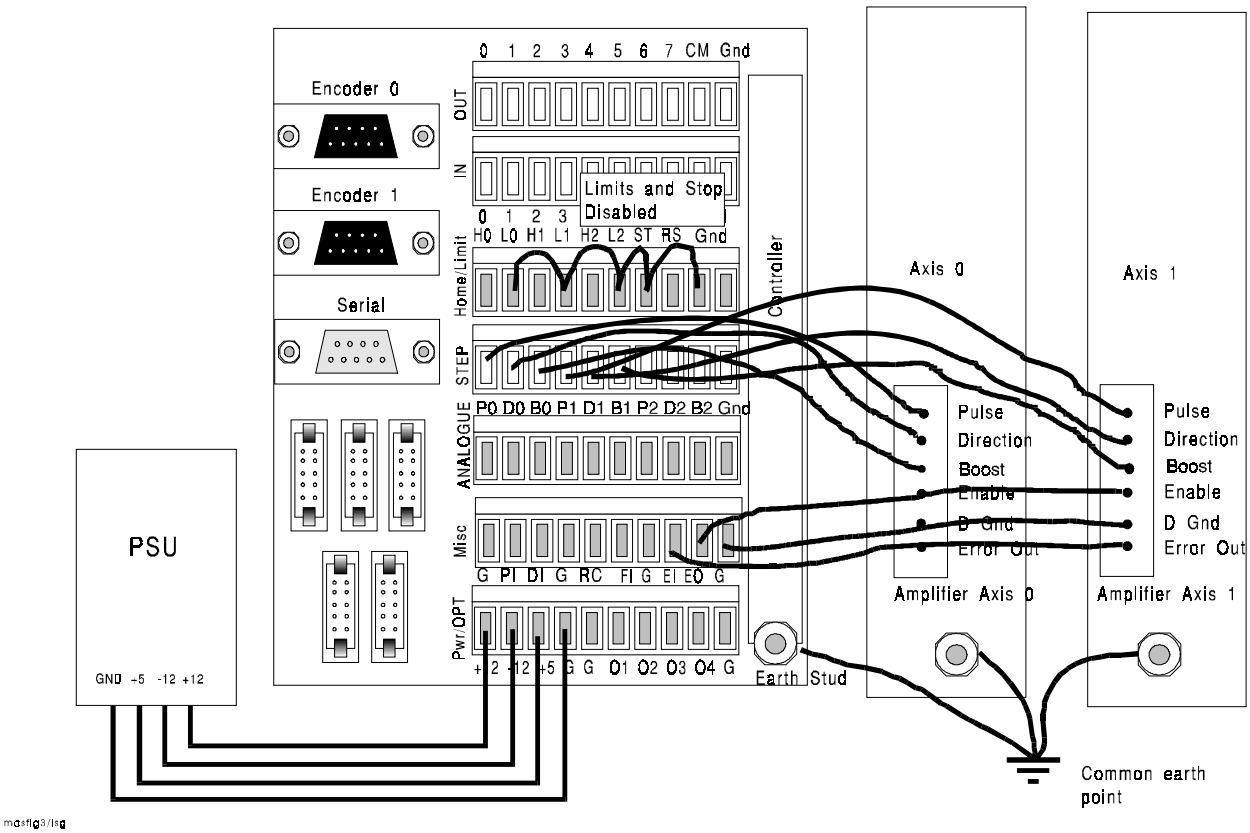
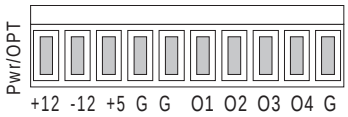


Figure 6: Minimum configuration for a 2 axis stepper system

Please note that the following wiring instructions relate to the standard non-isolated controller back plane only - the procedure for setting up the isolated back plane is similar - but different voltages etc. are required as detailed in the hardware guide.

3.1.1 Power Connections: "Pwr/Opt" Screw Connector

The controller power supply requirements are: +5V @ 0.5A (labelled "+5" on the connector), +12V @ 100mA "+12" and -12V @ 100mA "-12" and ground "G".



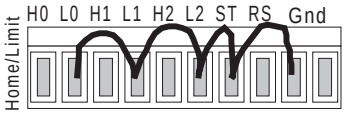
If not a rack system with internal power supply already fitted, connect +12V,-12V,+5V and G (0V) on the "Pwr/Misc" connector to a stabilised three rail power supply with at least 1A capacity on the 5V line.

3.1.2 Limit and Stop Switches: "Home/Limit" Connector

The limit switch inputs are connected through normally closed over-travel switches on the end of the axes of motion and cause motion to stop when floating. The inputs must be grounded to operate; usually through normally closed switches on the axes and guards (if two limit switches on one axis are used, they must be connected in series). This arrangement means that the controller is fail safe, if a wire breaks, the controller stops motion.

The Stop switch input causes all axes to decelerate to a halt and is used for connection to machine guards. This also must br grounded in order to allow motion.

For the purposes of setting up the controller, these inputs can be linked out on the back plane:



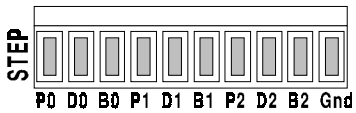
Remove the screw connector labelled "Home/Limit" and connect L0 (limit axis 0), L1, L2, ST (stop motion input) to Gnd. Use insulated thin gauge wire for the connections.

If the limit or stop inputs are not properly connected, the controller will display an "L" or an "S" on the status display on the front of the board on power up.

3.1.3 Step and Direction: "STEP" Connector

The controller uses pulse and direction signal for each stepper motor drive. The number of pulses delivered to the drive relate to the distance the motor will travel, each pulse translates to a single step. The distance that the motor travels per step is dependant on the drive set up (full, half or microstepping). The speed of the motor is proportional to the pulse frequency. Furthermore the acceleration and deceleration profile of the system is controlled by the rate of change of pulses. You should consult your stepper drive and motor supplier for recommended speeds and acceleration rates for your particular system.

Before proceeding further with the set-up procedure, please ensure the stepper drives have been correctly set-up for the motor in accordance with the instructions in your amplifier manual.



On the "STEP" screw connector, connect "P0" (Pulse output axis 0) to the step input on the first drive and "D0" (Direction output axis 0) to the direction input on the first drive. For some stepper drives there is a "BOOST" input which is used to enable a short term current increase typically during acceleration. If this is required connect the "B0" (Boost output axis 0) to the boost input of the first drive. For each axis connect "Gnd" to the amplifier ground. The Pulse signal should use an individually screened cable with the screen connected to the ground stud on the backplane and not at the drive end. Other signals should use twisted pair screened cable with the screen connected to to the ground stud on the backplane and not at the drive end.

3.1.4 Stepper Drive Enable

On the "MISC" screw connector, connect the enable input on the amplifier to the enable output from the controller. The enable output from the controller is configurable by jumpers so that it is active high or active low.

Connect the common terminal "EO" to the enable input and the "G" to the amplifier ground.

It is essential that the amplifiers are disabled during power-up of the controller or when a controller error occurs. This ensures that the amplifiers are only able to drive the motors when the controller is performing servo-loop closure. This involves changing the jumpers on the controller JP4 JP5 and JP6 to set the correct sense of the enable output - see the HARDWARE REFERENCE GUIDE for further information.

On power up, the amplifiers should be disabled. You may find that by issuing the RESET command the amplifiers are disabled. If this is the case then the error output is the wrong sense.

3.1.5 Ground Connections

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

Connect the earth stud on the controller and amplifiers to a common earth point using heavy duty cable. Make sure that the connection is in a 'star' configuration as shown in the minimum system wiring diagram.

3.1.6 Serial Cable

The RS232 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, but required for programming. Use a standard serial cable from your system supplier or build one up according to the wiring diagram in the Hardware manual.

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.

3.2 Testing System Wiring

In order to check that the motor is wired up correctly, it is recommended that the motor is tested and commissioned 'on the bench' and not in situ.

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 sense.
2. Check the amplifier is working.
3. Check that the motor is connected the right way round.

The program examples in this manual and contained on the cTERM disk that is enclosed with the manual cover the setting up of closed loop servo systems with EuroSystem. Consequently they are more complex than that required for verifying proper operation of the stepper system but many tasks required are the same. In particular the setting up of system gains and following errors are not required for stepper systems. The disk icon is used to indicate a program (each program is enclosed in a box) the name under the disk icon indicates the name of the program on the disk. These programs can be modified by the use of an external editor or by use of the on-board editor on the controller. Typing RUN will execute the programs. Single line examples can entered direct at the command line for immediate execution, indicated by the prompt icon: P>

3.2.1 Starting cTERM

The cTERM disk contains a program for PCs that makes the computer work like a terminal emulator. A terminal emulator program accepts character input from the computer keyboard and sends these characters down the serial port so that they can be interpreted and processed by the controller microprocessor. The terminal emulator program displays any information that is sent back from the controller on the computer screen. Therefore, you use the terminal emulator program to 'talk' to the controller. For example, if you press the "A" key on the keyboard, an "A" character is sent down the serial port and the controller responds by sending the same character back which appears on the computer screen. Of course, it all happens so quickly that you don't notice any delay between typing the character and it appearing on the screen.

The terminal emulator therefore allows you to create programs in the memory of the controller by typing them into an IBM compatible computer, using the editor on the controller.

A number of other features are available within the cTERM program, for instance, the ability to use your favourite editor on the computer to create programs for the controller and then download these at the press of a button. These features are covered in the cTERM and MINT Utilities manual.

To run cTERM, insert the disk in your 3.5" floppy disk drive slot (drive A: or B:). Log-on to that drive by typing "A:" or "B:" followed by return, then type "cterm" and press return.

Details of using cTERM are given in the manual cTERM and MINT Utilities. Please familiarise yourself with operation of cTERM, especially up and downloading of files, by reading section two of this guide before you proceed further.

Next plug your serial cable into COM1 in the back of the computer, and into the 'D' type marked "Serial Port" on the controller back plane, or into the front of the controller (which is a duplicate of the connections on the back plane).

Plug the power lead into the euro-socket on the back of the control system rack and switch-on. The controller should power-up and the message:

```
MINT vX.XX
C>
```

should be displayed on the computer screen and also on the LCD display in the rack (if fitted) If this is not the case, check the wiring of the serial cable and refer to the Trouble-Shooting section in the back of this guide.

3.2.2 The Configuration File

The configuration file is used to store all the defaults appropriate to the particular system. Once suitable acceleration rates and speeds have been found, these should incorporated into a configuration file for your system.

Whenever the program is RUN, the configuration file is first executed. A list of parameters that you may need to set up follows:

- Scale factor - to set the units of measure of the application - see MINT Programming Guide for details of using SCALE.
- Default speeds (SPEED) and accelerations (ACCEL) for the system - to determine the shape of the trapezoidal velocity profile (RAMP).

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

A typical Configuration file for a three axis open loop stepper system may be:

```
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 system
CONFIG[0,1,2] = _stepper; : REM configure all axes as steppers

REM position/SPEED parameters
SCALE = 2000;  : REM units revs (2000 pulses per rev ministepping drive)
SPEED = 10;    : REM max SPEED 10 revs/sec
ACCEL = 20;    : REM max accel 20 revs/sec^2
```

In the above example, the default axis list is 0,1,2 the semicolon is used to apply all the parameters to all three axes, see the MINT Programming Guide for further details on program language syntax.

Please note that the above values are given only as an example. It is up to you to determine the best values for your system.

3.2.3 Running the System

Having set the system up correctly, proper operation can be verified by typing commands directly at the P> prompt.

Example:

Assuming that the SCALE has been set to revs/s.

```
JOG = 10
```

will jog the motor on axis 0 to run at a constant speed of 10 revs/s

```
JOG = -10
```

will jog the motor in a negative direction of 10 revs/s

Should the direction need to be reversed, this can be achieved by reversing one of the stepper drive phases, please refer to the drive suppliers instruction.

Other simple MINT commands such as MOVER and MOVEA may also be used to verify the system, please refer to Section 4 for more specific information and examples.

4. Introduction to MINT Programming Language

4.1 Your first MINT Program

MINT is the programming language used to program the controller 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. You may have a an application where a thumbwheel switch fed into the digital inputs of the controller sets a distance to move and the a potentiometer fed into one analog input on the controller changes the slew speed. A simple MINT program would be written to achieve this.

MINT programs consist of two files. The configuration file stores information relating to the machine set-up, for instance the servo loop gains. The program file stores the actual motion control program. In fact the two files are the same and can contain the same instructions, except the configuration file is only 1K maximum size, while the program file can be up to 27K. In addition 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 about this is given in the MINT programming guide).

This first MINT program consists of a simple configuration and program file which is used to index a motor a set distance entered via a computer or via the operator keypad. The configuration and program files can be found on the APPLICATIONS AND UTILITIES DISK in the directory \START. Please note that all the following examples refer to closed loop servo systems, programs should be modified for use with open loop stepper systems.

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 factor for revs assuming 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 you to insert comments in the program which make it more readable at a later date. It is important to keep back-up copies of your programs and configuration files on disk and the first line of this program indicates the name that we have given to the disk file. Download this configuration file to the controller as explained in the cTERM and MINT Utilities manual. Remember that you may have to change the gains and scale factor to suit your motor.

Tip: *When downloading a configuration file from cTERM you may get the message "Cannot Download" printed on your computer screen. A common reason for this is that there is already a program running on the controller. You can check this (and abort the program) by going into the terminal screen and pressing the "E" character 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
```

Download the program file, FIRST.MNT, to the controller and type "RUN" 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. Verify that the configuration is in accordance with the instructions in the Setting up section of this guide or refer to the Fault Finding section in the back of this manual.

4.1.1 Program Narrative

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

```
index_length = 0
```

this defines a variable called *index_length* and initialises it to the value 0. *index_length* is used later in the program to store the length of move entered by the operator. The second MINT statement:

```
KEYS""
```

is used to initialise (turn on) the keypad interface on the controller. The KEYS command can be used to set-up the layout of any 8x8 matrix keyboard, but this example uses the default layout which is that of the standard 27 key operator keypad.

The operator keypad works just like a standard serial terminal. Pressing any of the keys on the terminal causes a character to be placed in the serial port buffer so that it can be read by the program by using the INPUT, INKEY keywords etc. that can be found in the popular programming language BASIC on which MINT was based. Similarly the keyword PRINT can be used to output data to the 20 character by four line LCD screen 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 next statement:

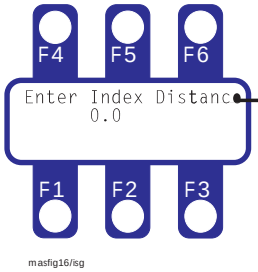
```
LINE 1,"Enter Index Distance"
```

prints a message on line one of the terminal. The LINE keyword was specially written to make printing information on the LCD display easy. Note that you could equally use the standard BASIC LOCATE and PRINT statements to do this; i.e.:

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

The LINE keyword however ensures that there are no characters 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 that he wants 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:



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

Tip: *It often helps if there is a BEEP at the keypad whenever a key is hit, since this gives the operator audible feedback that he has hit the key. You can do this by putting the statement BEEPON in the program.*

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 using the accelerations and speed previously set up in the configuration file. The GO command is required to actually start the motion, it is useful if you need to synchronise both relative and absolute moves on two or more motors. GO is not required for certain types of moves, notably continuous speed control using the jog command.

You can see that the MOVER statement is surrounded by a FOR.. NEXT loop statement. This causes the statements inside the loop to be executed 10 times, each time the number being stored in the variable *cycle* which is printed to the operator terminal each time round the loop.

Finally, program arrives at the ENDL statement, which causes it to jump back to the LOOP statement so that you can enter the next index length. FOR .. NEXT and LOOP .. ENDL are two examples of the four different loop statements in the MINT language. These statements are very useful for writing machine control programs.

To end program execution, type the keys "CTRL-E" at the terminal screen.

4.2 A Simple Cut to Length Feeder

In many applications, the MINT programming language 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 and records the total length of product that has been cut.

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

The program file FEEDER.MNT and the configuration file FEEDER.CFG can be found on the cTERM disk. Start cTERM and download these files to the controller as before and type RUN at the P> prompt

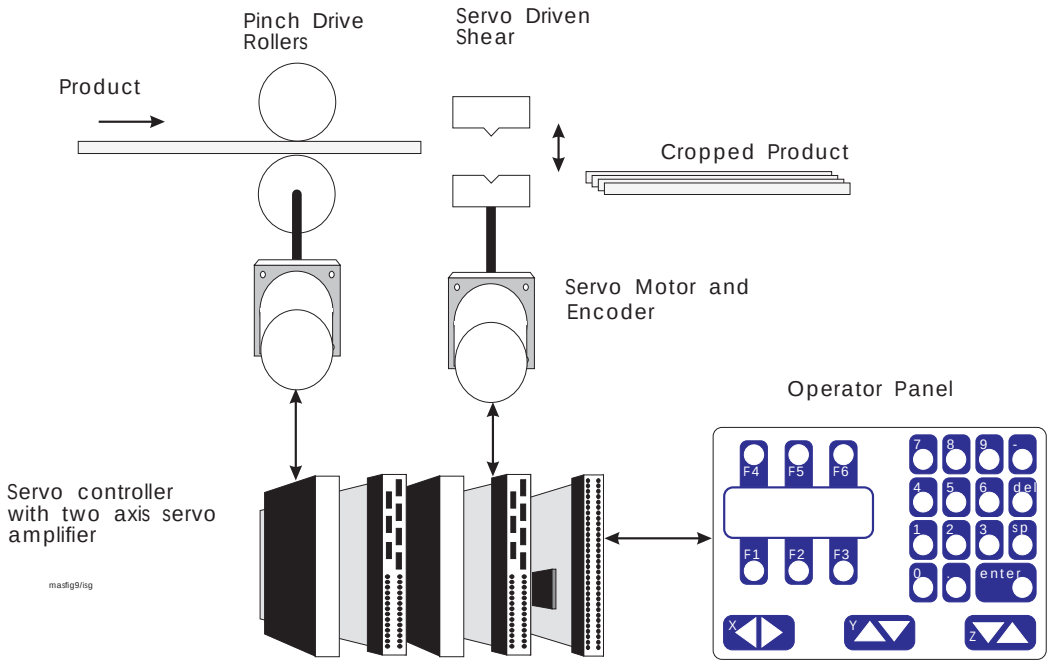


Figure 7: Schematic Diagram of the Cut to Length Feeder

4.2.1 Configuration file FEEDER.CFG

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

REM File name: feeder.cfg
REM Configuration file 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/sec^2
RAMP = 0;       REM no 's' ramping

END
```

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

The very top of the file contains the statement "AUTO". This command signifies that the configuration then program file should be run automatically on power-up. If automatic execution is required, AUTO must always be placed at the very start of the configuration file. When you turn 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) while the controller compiles and executes the program file. AUTO must always be on the first line of the configuration file, this is the only MINT command that cannot be used anywhere in a program or configuration file.

The statement "AXES[0,1]", indicates that the controller is fitted with two axes of motion and that all commands thereafter will relate to these two axes unless explicitly indicated otherwise by enclosing the axis number in brackets are the command (e.g. SPEED[1] = 10 sets the speed of axis 1 to 10, but SP = 10; sets the speed of both 0 and 1 to 10. Note the use of the semicolon to set both axes to 10, otherwise you would have to type SPEED = 10,10).

Also note the inclusion of the RESET command. While not strictly necessary, this ensures that the controller starts in a known state with all motion parameters and error flags reset to their default values.

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 you wanted to program speeds and distances in mm, you would set SCALE = 50.

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 that you are using - refer to section 2 of this guide for further information.**

4.2.2 Program file FEEDER.MNT

```
REM File ... feeder.mnt
REM Author . Ivor Gillbe
REM Date ... 29-3-92
REM
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 initialise REM call initialise 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 initialised 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

#initialise
```

```
 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 up 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
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
```

4.2.3 Cut To Length Program Narrative

The program has been written in a structured method to make it easy to maintain, according to the following prototype:

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

#main
...
RETURN
```

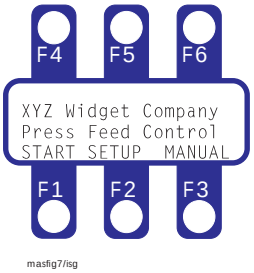
The program is well commented and therefore is fairly easy to follow. By making suitable adjustments to the SCALE factor and gains in the configuration file it should be possible to get the program working on any system equipped with a keypad and at least one axis of motion.

The *#non_volatile* subroutine is defined but never called. This is used to define some values which we use in the program to store the product length etc. entered by the user. 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 initialisation sub-routine, *#init* is called first. Initialisation sets up some defaults for the system, first enabling the keyboard with the KEYS" " keyword. The operator keyboard works in exactly the same way as a serial terminal, i.e. pressing a key causes the appropriate character to be sent to the controller serial port buffer. The function keys labelled 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 checked in the program, giving a context sensitive 'soft key' type operator interface.

BEEPOFF causes the automatic beep on pressing a key to be turned off. The beep from the buzzer is used to acknowledge a key press, this program is written so that a beep is only issued when a valid (active function key) is pressed. 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:

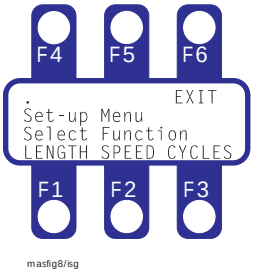


Note the use of START, SETUP and MANUAL to create user definable legends for the function keys F1, F2 and F3 and the keypad. The routine then loops round checking to see if a key is pressed at the keyboard. F1, F2 and F3 return the characters "A", "B" and "C" when pressed. For instance pressing F3 causes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

Pressing F1 will start the operation, F2 sets up the parameters of 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 (see 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*, somewhat 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, with the display showing:



Pressing F1, F2 or F3 calls the *#get_len*, *#get_cy* and *#get_sp* subroutines which allows 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.

As in many programs, the actual movement part of the code is very simple. the #start routine contains the code to do this. In this example, a relative move (i.e. distance from start) is executed on the material feed axis; followed by an absolute (i.e. 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 BASIC 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 special routine that 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 up an error message and waits for a key to be pressed before re-running the program from the start.

Try running the program and changing some of the parameters at the operator keypad. You can abort program execution by pressing CTRL-E at the terminal screen.

4.2.4 Using Batch Numbers

A common requirement in a cut-to-length machine is to store the parameters (speed, length and number of cycles in our example) that have been entered for different products and restore these quickly using a batch code. MINT allows you to do 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 guide.

The program FEEDER2.CFG on the cTERM disk is a simple 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 user to enter 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 inputted information is that they can be uploaded into a computer and stored to disk just like 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 can all be achieved, while the program on the control system is running by using the LOAD and SAVE commands.

4.3 X-Y Teach and Replay Program

This example program illustrates the use of array variables for storing and replaying positional data which is programmed by an operator using the X and Y joystick keys on the operator terminal. 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. Use the configuration file CONFIG.CFG on the cTERM disk 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.

Like many programs, this example has a lot of terminal I/O and therefore formatting of the information printed on the screen is very important. Note the use of the LINES keyword to print information on the screen, and the colon after any statement on line 4. This suppresses the carriage return that follows any normal print statement. This is very important, since the LCD screen has only four lines of display, a carriage return on line four will cause the entire screen to clear.

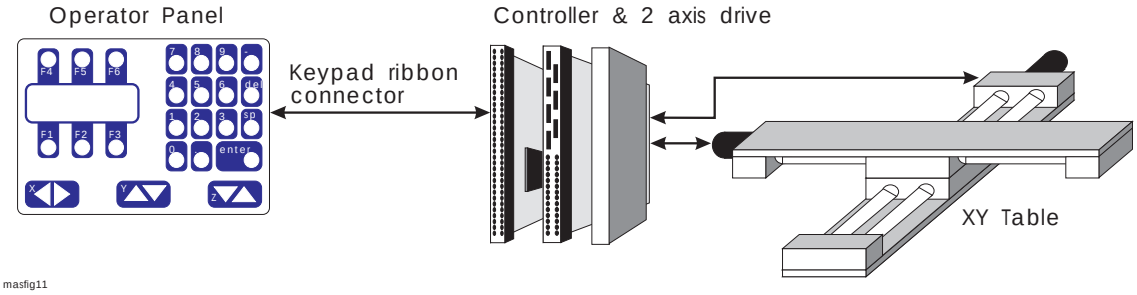


Figure 8: XY Table Control System

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 pressure on the operator panel X and Y keys. It is relatively easy 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 - 1024 between -10V and +10V (or 0-5V). The best 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 A1 and A2), will return 0 when the joystick is in the fully negative position and 1024 when in the fully positive. A value of 512 will be returned when the joystick is central. Most joysticks are equipped with trimmers to adjust the zero position, but electrical noise can cause some jitter about 0V, so it is good practice to put 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 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 feature that would be worthwhile adding to the teach program is to datum the axes on 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 Guide.

4.4 Software Gearbox Example - Coil Winding Machine

The controller has a 'software gearbox' function that makes it ideal for controlling machines that have to follow a master encoder or pulse train. A typical requirement is to wind wire onto a drum which rotates 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 which the controller uses as a reference for a servo motor which drives the spool back and forth via a lead screw.

Encoders can be incremental or multi-turn absolute types. Multi-turn absolute encoders are more expensive than incremental, but always indicate the absolute position of the axis. In the spooler example, there is no real advantage in using an absolute encoder to measure spool rotation, but if the servo motor is fitted with an absolute encoder then it is not necessary to datum the unit on power-up. This is an important feature when recovering from a power failure, since it allows the machine to re-start with a part full spool.

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

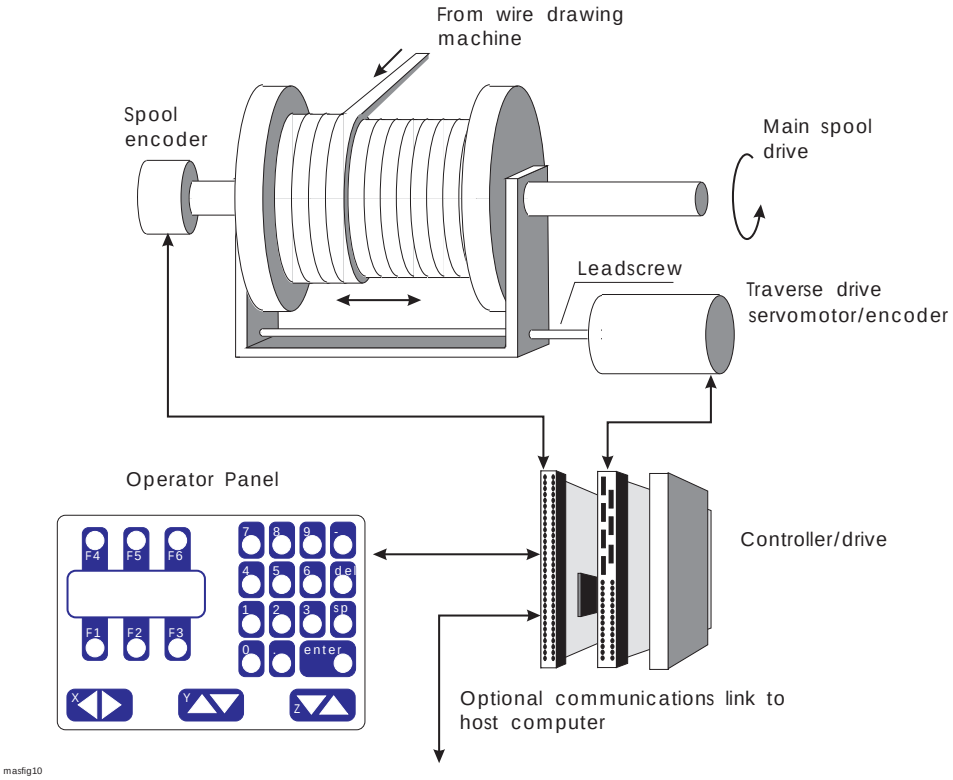


Figure 9: General Layout of Coil Winding Machine

The example MINT program uses the standard operator panel with it's six function keys which return 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 (this program is not included on the cTERM disk).

```
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 datum on limit switch (negative direction)
HOME = _neg

REM infinite loop to provide operator display
LOOP
  CLS : REM clear operator panel 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
```

4.4.1 Program Narrative

The variable *ratio* is used to store the 'software gearbox' ratio, i.e. the number of servo motor encoder pulses moved per pulse from the spool encoder. This ratio is variable anywhere 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 a simple 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.

4.4.2 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 100 location array called COMMS. COMMS works like a letterbox, 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 is discussed in detail in the MINT Programming Guide and the CTERM AND MINT UTILITES manual, which covers the use of cTERM for reading and writing to the controller using protected communications.

4.5 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. *Product* is next transferred to the *Synchronising 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 variable, 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 order to achieve high throughput, the offset acceleration and deceleration must be performed as quickly as possible. This can result in the product slipping on the belt and thus introducing a further positional error which means that it is not accurately synchronised with the flights. Problems with products slipping have been reduced by using 'S' shaped smoothed acceleration-deceleration profiles.

In practice it is often possible to have two products on the synchronising 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 terminal, a single line LCD display and keypad, which is panel mounted on a convenient part of the machine. The operator terminal is also used to enter set up information, for instance product length and the speed of the metering conveyor.

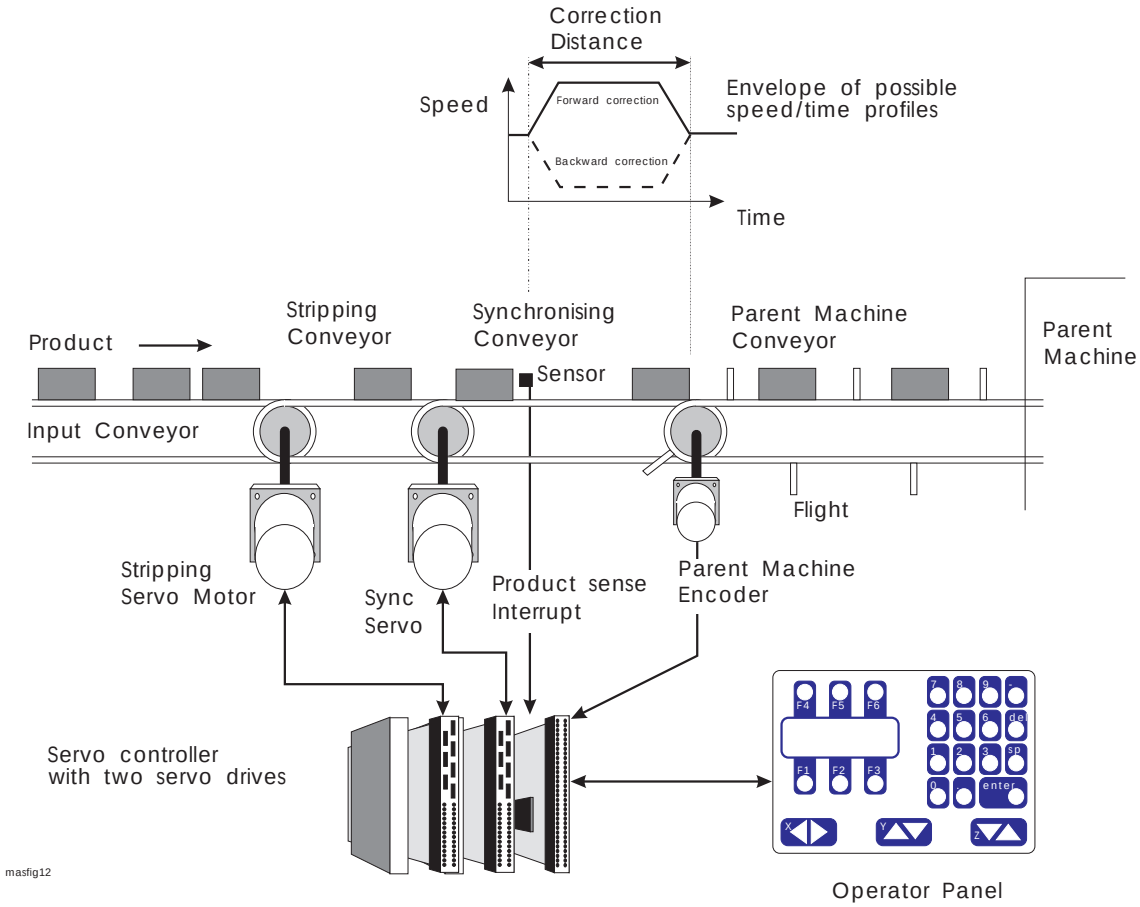


Figure 10: Diagram of Product Infeed Machine

The program is a simple example of the application of the controller in an Infeed machine. In more complicated applications of the machine 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 and painlessly. 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.

```
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 Synchronising 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.

5. Trouble Shooting Guide

Symptom	Check
Status display blank and controller not functioning.	1. Check power supply, +5V,+12V and -12V is connected and switched on.
Status display shows 'S'	1. Check stop switch input is correctly wired and has power applied
Status display shows 'L'	1. Check limit switch input is correctly wired has power applied
Status display shows 'L' and wiring to switches is o.k.	1. Check that opto-isolator supply (UP) is connected.
Status display shows an '8.' and controller does not communicate over serial port.	1. Check that the 5V supply is at least 4.9V (controller is held in reset).
Cannot communicate with controller over RS232 port (cannot get "P>" or "C>" prompt by pressing return.	1. Verify that the terminal emulator program is loaded and set-up correctly. 2. 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. 3. Check wiring for RS232 lead. 4. Check that +/-12V is supplied to the controller. 5. Check that there is not a program ready running on the controller (type Cntl-E to abort the program). 6. Check that the controller card is not an RS485 model (model number on card should not end in "/4"). 7. Check that card address switch is set correctly (normally 0000).
Controller appears to be powered-up and working but will not cause motors to turn over.	1. 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 input to the amplifier - remember to ensure that the motors will not cause any physical damage when doing this). 2. Ensure that the amplifier is enabled and working when the controller is not in error. (When the controller is first powered up the amplifiers should be disabled if there is no program loaded for automatic execution (there is often an LED on the front of the amplifier to indicate status). Typing "RESET" at the "C>" prompt should cause the amplifiers to be enabled.) 3. Check that configuration file 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).

Symptom	Check
Motor runs off uncontrollably when controller is switched on (Status display shows an '8').	1. Check that encoders are connected to controller and are functioning correctly. (Use a dual trace oscilloscope to display both channels of the encoder simultaneously). 2. Check that amplifiers are connected correctly to controller and that with zero demand there is 0V at the amplifier demand input. (Type "SO[0,1,2]" to set all demand 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.) 3. Verify that amplifiers are correctly set-up and that the motor does not move with 0V on the demand input. 4. Verify that the controller and amplifier are correctly grounded to a common earth point.
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'.	1. Check that encoder 0 and demand 0 "D0" are connected to the same axes of motion; repeat for axis 1 and 2. 2. Check amplifier connection is correct with respect to polarity of amplifier demand. (Try swapping the +demand and -demand connections to the amplifier; note: this is not possible with some amplifiers due to ground reference problems in which case you need to swap the encoder channels A and B.) 3. 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.	1. Servo loop gains may be set too high. (Go throughout the servo system set-up procedure to establish correct gains.)
Motor is under control, but when moved to a position and back to start does not return to the same position.	1. 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.) 2. 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. 3. 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. 4. Ensure that the encoder leads use screened cable and that the screen is attached to the screen connection on the encoder plug **(at the controller end only**.) 5. Verify that the controller and amplifier are correctly grounded to a common earth point.