NATIONAAL LUCHT- EN RUIMTEVAARTLABORATORIUM

NATIONAL AEROSPACE LABORATORY NLR

THE NETHERLANDS

NLR CR 89301 L

EXTENDING THE PROTOTYPE OPTICAL DIAGNOSTIC INSTRUMENT (PODI) FOR ANALYSES OF TELESCIENCE AND IMAGE PROCESSING

by

A.J. Mooij, E.A. Kuijpers
and H. Keppel

NATIONAAL LUCHT- EN RUIMTEVAARTLABORATORIUM
NATIONAL AEROSPACE LABORATORY NLR

Anthony Fokkerweg 2, 1059 CM AMSTERDAM, The Netherlands
P.O. Box 90502 , 1006 BM AMSTERDAM, The Netherlands
Telephone : 31-(0)20-5113113
Telex : 11118 (nlraa nl)
Fax : 31-(0)20-178024

NLR CONTRACT REPORT

CR 89301 L

EXTENDING THE PROTOTYPE OPTICAL DIAGNOSTIC INSTRUMENT (PODI) FOR ANALYSES OF TELESCIENCE AND IMAGE PROCESSING

by

A.J. Mooij, E.A. Kuijpers and H. Keppel

This investigation has been carried out under a contract awarded by the Netherlands Agency for Aerospace Programs (NIVR), Contract number: 02607 N

Division: Space
Prepared: AJM/ EAK/ HK/
Approved: HFAR/

Completed : 890602
Order number: 101.918/672.701
Typ. : MM

SUMMARY

The Prototype Optical Diagnostic Instrument (PODI) has been extended for remote control and image processing. A new breadboard called TELEPODI (Teleoperated PODI) resulted in which telescience and image processing for microgee instrumentation can be studied.

Software libraries which have been developed to allow for remote control using several positioning devices are described. For the stake system in PODI these devices are stepper motors, to be controlled by two controllers in a daisy chained RS-232 configuration. Via IEEE-488 a Universal IEEE-488 interface is controlled. The digital I/O of this interface is used for the control of motorised drives which can be used for adjusting the plane of focus and control of a mirror for an interferometer as stake passenger.

The extensions of PODI are being used for several projects related to image processing and telescience.

CONTENTS

CONTENTS (continued)

(60 pages in total)

ACKNOWLEDGEMENT

As many people have contributed in extending PODI to TELEPODI and the number of authors should be limited, the following discussion of contributions is added. D. van de Assem and R. Huyser (now with Fokker Space and Systems) have proposed many of the hardware extensions of PODI which resulted in a very interesting basis for studying telescience and image processing for microgee instruments. D. van de Assem, I.G. van de Berg and H. Keppel have selected the positioning devices taking into account the requirements for remote control from the optics point of view and requirements for interfacing at the NLR Space division. H. Keppel designed electronics for positioning devices in TELEPODI. The NLR Electrical Engineering department contributed in several ways to TELEPODI (e.g. support for integration of VME boards for additional memory and interfacing to IEEE-488, support for integration of CADISS, consultancy from M. Versteeg for many questions related to VME and UNIX is gratefully acknowledged). A.J. Mooij did most of the coding and documentation for the software libraries for position control. E.A. Kuijpers was responsible for integration of contributions and image processing extensions.

LIST OF ABBREVIATIONS

CADISS	-	Compression And Decompression of Imaging Sensor Signals
BIN	-	Binary
DEC	-	Decimal
EOT	-	End Of Transmission
FICAD	-	Fluid Physics Instrumentation Compression and Decompression
GPIB	-	General Purpose Interface Bus (brand version of IEEE-488 standard)
HEX	-	Hexadecimal
IEEE-488	-	bus standard, see GPIB
MMI	-	Man/Machine Interface
PODI	-	Prototype Optical Diagnostic Instrument
TCP/IP	-	Transmission Control Protocol/Internet Protocol
TELEPODI	-	TELEoperated PODI
UDP	-	Internet User Datagram Protocol
VDU	-	Visual Display Unit

LIST OF FIGURES

1 INTRODUCTION

Experiments in Columbus will be executed in telescience mode in which instruments and experiments will be controlled from the ground. A number of activities have been started by NLR to be able to support scientists for conducting experiments in telescience mode for Columbus.

One of these activities is called TELEPODI which is based on extending PODI (Ref. 1). PODI is a "Prototype Optical Diagnostic Instrument" for microgravity fluid physics research, developed at NLR under (previous) contract with ESA and NIVR. Basis of the PODI design are general requirements, including telescience possibilities, for fluid physics instrumentation in Columbus.

This report gives an overview of TELEPODI including documentation of hardware and software extensions which have been realised. The extensions are based on a proposal for studying telescience and image processing for PODI (Ref. 2). The hardware and software extensions are being used for several projects.

In August 1988 an unsolicited proposal for executing experiments with TELEPODI was sent to ESA (Ref. 3). In this unsolicited proposal the installation of a Telescience Test Bed was anticipated for experiments, but only a rough estimate of work involved could be given. After the installation of the Telescience Test Bed at ESTEC, ESA asked for an update of the unsolicited proposal which was sent in June 1989 to ESA (Ref. 4). In the new proposal several experiments related to fluid physics telescience are proposed. HCS, a Dutch firm, will be responsible for developing an MMI using modern software development tools.

TELEPODI is a general tool for studying telescience and image processing for microgee instrumentation. To interface TELEPODI to the Telescience Test Bed at ESTEC is not the only follow on. For example, the digitisation, graphics and image visualisation possibilities are currently also being used for analysing the trajectory of the Wet Satellite Model based on video sequences (experiment by Dr. J.P.B. Vreeburg).

Chapter 2 will give an overview of TELEPODI. The chapters 3, 4 and 5 describe in more detail the hardware and software extensions for control of devices and image acquisition. Chapter 6 discusses some preliminary experiences using the available functions.

2 TELEPODI OVERVIEW

2.1 TELEPODI subsystems

2.1.1 Optics

PODI consists of a whole-field Schlieren and scene observation system combined with a narrow field optical stimulus/observation system which can accommodate various optical diagnostic instruments. An equal beam Twyman-Green interferometer and a moiré deflectometer have been developed as passengers for the narrow field system.

As in TELEPODI the telescience aspects are studied, the interferometer will be the basic stake passenger as a mirror needs to be controlled by computer to adjust the shape and number of fringes. The adjustments needed depend on the properties to be measured. This interactive aspect makes the interferometer more interesting than the moiré deflectometer in a telescience context.

In the current set-up the whole-field system of PODI contains two cameras. One camera is focused on a grid illuminating the experiment volume. This camera will be used for Schlieren or for grid deformation methods. Another camera is focused in the experiment volume. For both cameras the plane of focus can be adjusted by using motorised drives for position control.

2.1.2 Electronics

The electronics for extending PODI has been selected taking into account budget limitations, possibility for general use in breadboarding for microgee

instrumentation, compatability with NLR equipment, and requirements as generated from the point of view of the people who built PODI.

The following is now available:

a. PODI workstation (VME/UNIX crate, see ref. 5 for motivation)
   - Maxvideo Digimax and Framestore board for image acquisition
   - VME/IEEE-488 Interface
   - Ethernet board with support for TCP/IP
   - 68020 based VME board (Tadpole) with UNIX system V with 2 Mbyte of memory
   - Harddisk, streamer tape, floppy disk
   - Memory board for 4 Mbyte

b. Positioning devices and interfaces
   - Two stepper motors with controllers for x-y positioning of the stake (Ref. 6)
   - VME board for general IEEE-488 interface (Ref. 7)
   - Universal IEEE-488 interface (24 bits LISTEN and 32 bits TALK digital I/O) (Ref. 8)
   - 8 motorised drives for position control (see App. B)
   - Motorised drive controller for 8 motorised drives
   - Two stepper motor control interfaces

c. Three video camera
   - One Sony camera and two HTH cameras

In figure 1 the hardware for extending PODI is depicted. The VME/UNIX crate is basically a workstation (PODI workstation). In figure 2 TELEPODI set-up is depicted in which also the results of the CADISS (Ref. 10) and FICAD (Ref. 11) project are included. CADISS is a multi-processor system for compression and decompression of images. A VME based connection to the VAX (result of FICAD project) makes it possible to use CADISS for the images generated in PODI.

2.1.3 Software

The following software is available:

- Software library for control of stepper motors via RS-232
  This library contains an initialisation procedure and functions for relative or absolute positioning with two degrees of freedom. This library will be discussed in chapter 3. The functions to be used for the stake are based on the L and H parameters and commands as described in the instructions for SLO-SYN micro series indexers from Superior Electric (Ref. 6).
- Software library for the control of the VME/IEEE-488 interface
  General Instruments with an IEEE-488 interface can be added in TELEPODI using the VME board for interfacing to the IEEE-488 bus. The software for this control had to be adapted for use in the PODI workstation. This library is being used for control of motorised drives, see also section 2.2.2. In the following the acronym GPIB (General Purpose Interface Bus) will also be used instead of IEEE-488.
- Library for control of motorised drives using IEEE-488
  A motorised drive can be selected for which a speed and time interval have been specified. This library will be discussed in chapter 4.
- Library for control of Maxvideo Digimax and Framestore board (Ref. 12)
  This library is the basis for a number of functions for image acquisition, camera selection and display. These functions form the basis for interfacing to hardware for image digitisation.
- Image processing package TCL-image (Ref. 14, see for proposal Ref. 13)
  TCL-image is distributed by Multihouse TSI and has been developed at the Delft Centre of Image Processing. It contains a general command interface to many image processing functions. User written functions can easily be added. An image library can be used in combination with application programs. About the integration of this image processing package with the existing library for the Maxivdeo Digimax and Framestore boards, will be reported elsewhere.
- Library for control of CADISS and FICAD
  The software for the FICAD project has been extended to allow interactive control of compression and decompression parameters in CADISS. Fixed error

or fixed length compression can be chosen with 8x8 or 16x16 subwindows. The software of the FICAD project has also been extended to allow direct reading and writing via ethernet of images instead of using disk-files.

For a number of practical reasons all routines are written in the C-programming language, except for functions related to the FICAD project on the VAX. Some reasons are: experience with the C-programming language exceeded experience for other languages considerably, all image processing had to be based on the C-programming language, many UNIX operating system related calls were needed which can in most cases only be used optimally when programming in the C-programming language. The UNIX operating system was chosen for compatibility with other requirements. As the number of users will be limited, limitations in real-time performance of UNIX were not considered to be a problem.

2.2 Interfaces

2.2.1 RS-232

The workstation for PODI contains two RS-232 ports (Ref. 16). One port is used for the console and the other is currently used for control of the stepper motors.

When the terminal port is used for control of the stepper motors, the "getty" process which handles access to the computer via a terminal for that port must be disabled. This is done by editing the file /etc/inittab as described in the UNIX manuals (Ref. 15).

The stepper motors used in the PODI workstation are controlled via terminal port ttyb at 9600 BAUD with start/stop input and output control, ECHO enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. The system will transmit START/STOP (XON/XOFF) characters when the input queue is nearly

empty/full. This is controlled by the function set_ttyb (see App. C for programmers manual). There are 8 data bits. Parity shall be disabled.

The terminal port which corresponds with device ttya is used for the console with specific requirements for the terminal settings. The RS232 Console lead should be wired to an asynchronous terminal running at 9600 baud with parity disabled, 1 start bit, 1 stop bit, 8 data bits. A minimum three wire connection should be made to start with, together with any strapping the terminal end may require for three wire (Rxd, TxD and GND) operation (see Ref. 16 and Fig. 2.3).

The initialisation of parameters for the RS232 lines is fairly difficult in general. For testing the functions "cu" (a general UNIX utility) and "kermit" (Ref. 9) may be used.

2.2.2 IEEE-488

The motorised drives are controlled via a Universal IEEE-488 interface with digital I/O. This Universal IEEE-488 is controlled via a VME board in the PODI workstation for interfacing to IEEE-488. The routines can only be used with the effective User Id of root as the "phys" call is used for accessing the VME board under the UNIX operating system. To be able to use the routines as normal user, the sticky bit for the executable file should be set and the owner of the executable must be root.

2.2.3 Interfacing via ethernet

An ethernet interface is available with support for the UDP and TCP/IP protocol. Using this interface PODI is connected to a VAX and an IRIS 3020 (see Fig. 2.2).

It is noted that the performance for communication depends not only on the receiver but also on the sender. E.g. transfer of files from PODI to the VAX is possible at a rate higher than 50 kbytes/s. Transfer of files from PODI to the IRIS 3020 is currently possible at a rate of approximately 9 kbyte/s. Strangely

enough, transfer of at least 50 kbytes/s is possible for transfer from the IRIS 3020 to PODI. Transfer of files between the VAX and the IRIS 3020 is possible at a data rate higher than 50 kbytes/s. After acceptance testing of the PODI workstation at least 50 kbytes/s was measured for transfer from the PODI workstation to the IRIS 3020. A considerable decrease in performance was measured after installation of a new release of the operating system and ethernet software for the IRIS 3020. Probably this new release has less optimal parameters from the point of view of PODI. However, formally no one seems to be to blame as the protocols are implemented correctly and no specifications about speeds are given for UNIX systems, especially for non-standard systems.

As the data rates of at least 50 kbyte/s using ethernet, were envisaged in the original proposal for communication between PODI and the IRIS 3020, and this was considered to be near the minimum needed for Columbus infrastructure simulations, a considerable effort has been devoted to improving the performance as the operating system for the IRIS 3020 should not be replaced by the old version. This was only partly successful.

For high-speed communication between PODI and the IRIS 3020 two solutions were analysed:

1. The UDP protocol is a very simple protocol using internet packages, with no error correction. It is possible to design an error correcting protocol on top of this protocol. A method in which for every package containing image data the line number in the image was included, resulted in approximately 5 s for transmission of a 512x512 byte image. Therefore the experiments showed that performance could be increased considerably but a considerable effort was envisaged for the design of a reliable protocol.
2. When a C-compiler for the VAX connected to the LAN to which PODI is also connected became available. the flexibility for using ethernet increased considerable. It became possible to write efficient application programs which copy incoming messages from PODI to messages to the IRIS 3020 and performance for communication from PODI to the IRIS 3020 increased to approximately 40 kbytes/s.

In the TELEPODI set-up transmission via the VAX is used in general as the effort to design a new protocol would distract too much from the objectives of TELEPODI.

3 CONTROL OF STEPPER MOTORS VIA RS232

3.1 Description Of Stepper Motors

In figure 3.1 the stake system as originally built under NIVR/ESA contract is depicted. The wheels for positioning of the stake are now replaced by two stepper motors and two SLO-SYN micro series indexers (from Superior Electric) have been installed for control of both stepper motors.

The SLO-SYN micro series indexer, will hereafter be referred to as indexer. The indexer enables the user to send commands to or receive information from SLO-SYN stepper motors. The indexers can be either programmed or be used in direct mode. Two indexers and two motors are in use for the PODI stake system in direct mode.

The next chapters describe the NLR written software to use the indexers with a remote computer. The software developed by NLR allows absolute movement with respect to the current origin or relative movement within specific limits for the stake system. As the indexers are daisy chained (see Fig. 3.2), the motors cannot be used simultaneously.

The parameters for position control refer to the rulers on the stake system (see Fig. 3.1). 10 cm in horizontal direction and 10 cm in vertical direction corresponds with the stake in the centre before the experiment volume. The unit for translation is 0.001 cm. This implies that the centre corresponds with coordinates (x,y) = (10000,10000) where x is position on horizontal ruler and y is position on vertical ruler in 10^{-5} m.

3.2 Initialisation of indexers.

The RS-232 cable for the connection of the indexers and the Tadpole computer has one 25 pin connector which should be connected to the upper RS-232 connector of the PODI workstation on the backside, which corresponds with the terminal port called ttyb. On the other end of the cable are two 9 pin connectors, connected as depicted in figure 3.2.

The two indexers are mounted together on a baseplate. Seen from the side of the indexers where the power switch is located, the first connector must be connected to the left indexer and the second connector to the right indexer. The horizontal axis motor of the stake system must be connected to the left indexer and the vertical axis motor to the right indexer.

3.3 Test software and library

3.3.1 Init_stake

With the program "init_stake" all L parameters and all L and H commands can be send to the indexers. For example, the stake can be reinitialised in case for some reason the memory for parameters (Electrical Erasable Read Only Memory) is corrupted, and the parameters for each indexer can be read.

The program "init_stake" performs the following actions. The indexers are initialised by calling "open_stake". Then the device attention command <nn is sent with nn = 0 addressing both indexers. The command L26 n with n = 1 is sent to force the indexers to send an END OF TRANSMISSION (EOT) character (ASCII code 04) after all transmissions to the host. Then the program asks the operator to enter L or H instructions or END in case he wants to stop the program. As long as no END is entered the commands are sent to the indexer(s) with the C-function "stake_command". Using the function "print_stake" the contents of a buffer containing a reply from the indexer to the host computer can be printed.

The current status of an indexer can be read by sending the string "H16" to indexer with address nn. A list of parameters is returned, including the address. If this doesn't work the addressing of indexers is probably corrupted. The following procedure should be used to correct the addresses. To set the address for a specific indexer, disconnect all other indexers and send "<00 L21 nn" where nn is an address which can have a value of "00" to "99".

For application in TELEPODI, the left indexer (horizontal axis) must be assigned the device identification number "01" and the right indexer (vertical axis) the

number "02". The program "init_stake" asks for a motor number and an instruction. Therefore using the program "init_stake" motor number "00" and instrument "L21 01" resp. motor number "00" and instruction "L21 02" should be send to the connected indexer with the other indexer disconnected.

3.3.2 Stake

The purpose of this program is to set both indexers in a well defined state. The user is asked to give the current position of the stake as can be read from the rulers on the stake system (see Fig. 3.1). The indexers 01 and 02 are initialised. The origin is set to the center of the field of view. The screen is cleared and, according to a menu, the user can enter new absolute positions or relative displacements in order to proof the proper operation of the stake. On exit the stake is moved to the center of the field of view.

3.4 Programmers library

The library which contains all necessary functions for the control of the stake system is described in appendix C.

4 CONTROL VIA UNIVERSAL IEEE-488 INTERFACE

4.1 Description of motorised drive

The routines in the library for control of the motorised drives via an GPIB interface (IEEE-488 interface) are discussed in the following sections. The routines are written to be used with the GPIB/VME primitives from National Instruments. For execution root permissions are needed as discussed in section 2.2.2.

The interface is assumed to be at GPIB address 0 with talk address (MTA) 0100 (octal) or '@' (ASCII), and listen address (MLA) 040 (octal) or ' ' (space in

ASCII)). The universal IEEE interface is at GPIB address 26 (032 (octal)), with talk address 0132 (octal) or 'Z', and listen address 072(octal) or ':').

The digital I/O of the Universal IEEE-488 interface is used for control of the motorised drive controller. The switch FORWARD/REVERSE on the motorised drive control box shall be in the middle position for computer control. The 1-4/5-8 motor selection switch shall be in the 1-4 position. The speed control potentiometer is to be set to a maximum speed which can be obtained using computer control. For control of the interferometer this potentiometer should not be in maximum position, as interactive control would become difficult due to the sensitivity for position changes.

The motorised drives can only be started and stopped. Position readout is not possible. To read the limit position of the motor it is necessary to read the status continuously; only at the moment a limit position is reached the status gives 1 for the forward limit or 2 for the reverse limit, else the status is 0. Forward corresponds with an expansion and reverse corresponds with a retraction of the motorised drive axis.

4.2 Hardware Extensions

For remote control of instruments in the optical path of the PODI test bed a computerised motor control has been developed. The starting-point for this design was a control unit, fabricated by the Newport Corporation, who also delivered the small DC motors for the necessary linear movements. The control unit is primarily meant for manual control of four motors, one at a time, selectable by a front-panel mounted switch; the direction of motion also is controlled by a switch. The velocity of the selected motor is controlled by a slide-potentiometer on the front-panel of the control unit. A simple computer interface is integrated in the controller, but this interface did not fulfil our requirements, so some modifications had to be implemented. These modifications are discussed in the following sections.

4.2.1 Standard Interface IEEE-488

The Newport motorised drive controller has a so called computer interface, but this interface is not compatible with any known standard bus concept. In order to make the Newport controller controllable by any computer with an IEEE-488 interface, use is made of an universal IEEE-488 interface unit, constructed at NLR and giving an output of 32 bits to the IEEE-488 bus and accepting 24 bits input from the bus. For this application not all bits are necessary.

The controlling computer can also be interrupted with a SRQ from the interface unit (Ref. NLR Memo RL-86-004 U). On the frontpanel of this unit an eight position dip-switch is available for the selection of the IEEE address; for the selection of Talker only or Listener only and for the generation of a Service Request (SRQ). The last three switches have to be in the downward position to assure proper operation. To control this interface a National Instruments GPIB-1014P interface board has been added.

4.2.2 Velocity Control

In the Newport controller as delivered, the velocity of the selected motorised drive could not be controlled by computer. Computerised drive selection is possible, but the velocity control had to be carried out manually. For computer controlled velocity a simple D/A converter, consisting of an eight position CMOS switch and an eight resistor ladder network was built into the controller. The switch positions are controlled by three bits from the computer, giving eight possible speeds of which one is for zero speed; three are for reverse speed and four are destined to forward speed. The ladder network is used as a voltage divider between + 5 V and 0 V, which gives, starting from 0 V, an increase of 0.625 V at each tap of the divider. This brings the output of the CMOS. switch, depending on the binary selection from the controlling computer, between 0 V and + 5 V in steps of 0.625 V. This output is amplified to a level between + and - 10 V with an operational amplifier. The amplifier has an offset of + 2 Volts for

the generation of negative outputs for the reverse motions. The output is defined as :

$$U_{out} = U_{offset} - (U_{input} - U_{offset}) R_{feedback\ resistor} / R_{input\ resistor}.$$

For an offset-voltage of + 2 Volts and a resistor ratio of 4 the output is:

$$U_u = 10 - 4U_i$$

giving outputs between + 10 and - 10 Volts in steps of 2.5 Volts. This output is connected to the analog input of the Newport controller. The three bits binary input from the computer are wired into the controller via three unused pins of the computer connector of the Newport controller. This gives the possibility to vary the speed and direction of each motorised drive under computer control.

4.2.3 Computer limit detection

If the selected motorised drive is at its forward or reverse traverse an automatic action shall occur that prevents the drive from stalling. To this end a speed command in the opposite direction shall be issued. The Newport controller delivers this signal as a 15 Volts kick to release the drive. As a sensor for the occurrence of these limits the Newport controller delivers a forward and a reverse limit signal, but not a combined limit signal with which the computer can be signalled that a limit has been reached.

On the printed circuit board of the Newport controller a gate is available that combines these two signals. This signal, going high if any limit is reached, is sent to the controlling computer. The run/stop signal was not needed any more because a speed of zero is available, so the wire for this signal was disconnected from the board and connected to a voltage divider on the output of the or-gate. The combined limit signal could be used to initiate a Service Request, which is a bus signal with which the controlling computer can be interrupted.

The interrupt handling routine for the IEEE interface board of the Tadpole System is not available, so this scheme cannot be used. Continuous checking of the statusbit could be possible if this status was continuously loaded into the IEEE system. The combined limit signal is only available if an actual status change has occurred, so this would lead to a "hangup" of the controlling computer. Inside the Newport controller a so called Jogging oscillator is available. This oscillator runs at a frequency of about 12 Hz. The amplitude of this signal is 15 Volts. With a built-in voltage divider to 5 Volts this signal is used to load the limit status bits in the IEEE controller. In this way 12 times per second the status of the limit switches (see next paragraph) is available for the controlling Tadpole system; 0 for no limit; 1 for forward limit and 2 for reverse limit. The status is sent to the computer in the third word out of four words.

4.2.4 Direction Detection

The forward limit signal of the Newport controller is only available if the manual limit switch is in the forward position; the reverse limit signal only if the switch is in the reverse position. This means that after each speed command that changes the direction of the motor a manual operation on the Newport controller has to be carried out. This makes it useless for computer control. Extra wiring has been added to make these switch positions computer controlled, but this is only operational if the manual switch is in its neutral (centre) position.

4.2.5 Control of eight motorised drives

Only four motorised drives can be controlled by the Newport controller; six are needed. This asked for a second motor controller, to be modified in the same way as described above. Since none of the motors runs simultaneously with any other an extra controller is not necessary.

Addition of a relay that switches the motor outputs from the original set of four motor output connectors to an other set of four connectors and vice versa, makes it possible to control eight motors, which is more than adequate. The relay is driven by a built in transistor and this transistor is computer controlled. The wire, controlling this transistor has been connected to an unused pin of the connector of the Newport controller. Instead of the original available two bits for computerised motor selection, now three bits are available for control of eight motors. The accompanying schematic diagram (Fig. 4.1) shows the implemented modifications and the connections between the Newport controller and the Universal IEEE-488 interface, together with the byte and bit numbers for the controlling computer.

4.2.6 Connectors

The Newport controller is connected to the Universal IEEE-488 interface by means of a 15 pin connector at the controller side and two 37 pin connectors at the interface side. Looking at the interface from the rear, the left connector is for input (LISTEN for PODI workstation) and the right connector for output (TALK for PODI workstation) (see Fig. 2.3 for numbering). The connections are:

Newport controller	Universal IEEE488 interface Right connector	
15 pin connector	37 pin connector (computer is TALKER)	
	pin	first byte bit
9 motor bit 0	4	0 MSB
2 motor bit 1	3	1
15 motor group 2/1	2	2
10 computer/manual motor select	1	3
12 run/stop	Not connected	
7 speed bit 0	8	4
8 speed bit 1	7	5

14	speed bit 2	6	6
3	comp/manual speed select	5	7 LSB
	ext. data ready	29	
			second byte
11	remote forward	9	3
4	remote reverse	10	2

Plume controller (not implemented)

heat resistor	13	7
select high/low current	14	6

Newport controller

15 pin connector

13	forward limit	25	0
5	reverse limit	26	1
6	ground	37 on both connectors	

Universal IEEE488 interface

Left connector

37 pin connector

(computer is LISTENER)

	pin	third byte bit
forward limit	25	0
reverse limit	26	1
ground	37 on both connectors	

To select motor 1 with full negative speed, the following codes have to be sent:

```
BIN       DEC  HEX  DESCRIPTION
00001101 = 13 = 0D  full negative speed.
    | ||
    | || ----- computer control bit.
    |  ------ sign bit.
    \ /
     |
       ------- speed selection bits.

BIN       DEC  HEX
00010000 =  16 = 10
\ /|
 |  ----- computer control bit.
  ------- motor selection bits.
```

Using an OR operation both binary numbers are combined into one byte and extended with a second byte containing direction information (bit 2 and bit 3).

This second byte can also contain information for control of other functions. E.g. the plume demonstration experiment as is available in PODI (Refs. 1, 4). In this experiment a resistor is heated during a short period which generates a "plume" which can be visualised using Schlieren or the stake passengers. This resistor may be controlled using relays.

4.3 Test Software

The program "tmotor" can send speeds and times to all motors in order to check the proper behaviour of the motors. After initialising the GPIB interface the VDU screen is cleared and the user is asked to enter a motor number between 1 and 8 to choose a motor or to enter 0 to stop the program in which case the GPIB interface is cleared and set to off-line. If the user did not enter 0, the time interval in ms during which the motor shell run is to be entered. Then the required speed is asked with 8 choices: one choice is stop, four speeds can be chosen in the forward direction and three speeds in the reverse direction.

The requested speed and time are sent to the selected motor by "tset_motor". The current system time is read from the system clock, the requested motor is started with the required speed. The time is read until the elapsed time is greater than or equal to the requested time. The motor is set at zero speed.

4.4 Programmers library

The library which contains all functions needed for the control of the motorised drives, the IEEE-488 interface and the Plume is an option to be implemented in hardware, is described in appendix D.

In the include file tablelib.h the definitions for the various speeds and the limit positions are given.

5 ROUTINES FOR IMAGE PROCESSING

5.1 Description of image processing

For using the Maxvideo Digmax and Framestore board a number of functions have been developed based on an existing libraries which accompany these boards. Some performance measurements were given in reference 3.

Important functions which have been used for initial assessment for image processing and telescience (see chapter 6) and which are implemented based on the existing libraries are:

1. initialisation

   An input and output look-up table has to be initialised, all multiplexers are set in a state used for normal operation. Interlacing and internal synchronisation are selected.

2. acquisition of single camera images

   Camera signals may not be synchronised (e.g. Sony camera generates independent synchronisation signal). Therefore for single camera acquisition two functions are available. One for single frame acquisition

in case the camera for input is not changed and another for the case the camera is changed.

3. high rate digitisation of images

   Image digitisation is possible in real-time. To store images in real-time on disk is not possible. Using the VME board with 4 Mbyte memory as an intermediate storage capacity, it is possible to digitise approximately 15 512x512 byte images in 5 seconds. After digitisation the images can be stored on disk files. This is very useful for breadboarding of image processing algorithms as the real-time capabilities of the processor board in the PODI workstation are limited. The plume experiment in PODI (Ref. 1, 4) is an example of a dynamic process which can be analysed using the extra memory for intermediate storage.

For the work described in reference 16 a library has been made of standard functions, as no general facilities were available. This included algorithms for labelling and run-length coding, calculation of image parameters, histogram, disk I/O etc. After succeeding in obtaining the image processing TCL-image package (Ref. 14), further development of those fairly standard algorithms could be stopped and more advanced applications of image processing can be considered. See appendix F for an overview of functions available in TCL-image.

5.2 General test program

Most of the implemented functions are integrated into a test program called "demo" which is tailored to specific experiments. For general image processing TCL-image is now used. The test program called "demo" proved to be very useful in adjusting TCL-image to the available hardware in PODI and is very useful in testing proper functioning of all components. For small experiments and developing algorithms for approximation of Schlieren (Ref. 17) this program was also used.

For testing purposes using demo, option choice number 5 can be used to initialise the Maxvideo Digimax and Framestore board. This is needed after a

system shutdown. Option number 19 is used for selecting continuous framegrabbing and option number 4 for to stop after acquiring the current frame.

6 PRELIMINARY EXPERIENCES USING A DEMONSTRATION SET-UP

Based on the libraries discussed in the previous chapters a number of telescience demonstration programs have been made. In the demonstration configuration PODI is situated in the optical laboratory of the Space Division. PODI is controlled from an IRIS 3020 in the Guidance and Control Laboratory of the Space Division which is in another building.

The dataflow is depicted in figure 6.1. First programs are started on PODI workstation and the VAX. After that, a program is started on the IRIS 3020 which tries to connect to the program running on PODI. After a connection is established, the PODI program tries to connect to the VAX program, and if successful the communication between the IRIS 3020 and the VAX program is established. During initialisation it is possible to select for a program which allows compression and decompression using CADISS.

In a pseudo language the demonstration programs on the three computers involved are described in appendix G.

The demonstration was considered as a verification of proper functioning of all components. The MMI on the IRIS 3020 is based on using the menu facilities and possibilities to visualise digitised video images.

Only limited experience with the set-up was available at the time of writing but the development of the demonstration programs proved to be useful in itself for improvement of libraries. The design of the software for control of positioning devices was adjusted to enable control in the presence of delays. The menu facilities of the IRIS 3020 had some unexpected shortcomings in using roll-over menus. An MMI design in which graphics and images are integrated proved to be very powerful for pointing the desired location of the stake using a mouse in a window in which digitised images are visualised. During testing of the software

for the stake system, it became clear that the definition of RS-232 protocol should not be changed when interfacing with another system. Based on specifications sent by Superior Electric, a reset switch using the parallel interface of the Indexers has been added. The use of this switch should be avoided but may prove useful in case the terminal line settings have to be changed.

The performance of a number of features have to be evaluated, e.g. the repeatability of movements of the motorised drives as no position feedback is available. Also, in a following phase algorithms for semi-autonomous focusing and adjustment of the interferometer to allow for calibration with a very low bandwidth may be developed. As discussed in Ref. 2 many other topics related to telescience and image processing need further elaboration in the current set-up.

7 REFERENCES

1. van den Assem, D., R.H. Huijser, Development of an optical diagnostic instrument, final report, NLR TR 87079 U
2. Kuijpers, E.A., Telescience and image processing for podi: An analysis and a proposal for a simulation setup. NLR TR 88005 L
3. Proposal for study of MTFF fluid physics Telescience scenarios and demonstration in TELEPODI, Memorandum RS-88-035 L
4. A Proposal for using TELEPODI as part of the Telescience Test Bed pilot experiment program, Memorandum RS-89-029
5. Kuijpers, E.A., Proposal for an electronic subsystem for the prototype optical diagnostic instrument (PODI), Memorandum RS-87-086 L
6 Instructions for SLO-SYN micro series motion controls indexer models 230-PI and 430-PI. Superior Electric
7. GPIB-1014P User Manual, National Instruments
8 Keppel, H., Universal IEEE488 Interface, Memorandum RL-86-004 U
9. Da Cruz, F., Kermit: A File Transfer Protocol, Digital Press, 1987
10. Roefs H.F.A., A. Monkel, CADISS: A multiprocessor system for image compression/ decompression on board scientific satellites, NLR TR 87076
11. Börger, J.B. A., Nauta, A. Monkel, A.J. Mooij, The FICAD image compression configuration, NLR TR 87172, to be published

12. MaxVideo Software Primitives, Datacube, September 1986, Doc. no. UM0000-2
13. Kuijpers, E.A., Voorstel voor de aanschaf van een aanvullend beeldverwerkingspakket voor ruimtevaarttoepassingen, Memorandum RS-89-007, in Dutch
14. TCL-image user manual and programmers manual
15. UniPlus+ System V manuals, Unisoft systems, July 16, 1985
16. TP 20V/TP 20M 68020 Processor card user manual, Tadpole Technology, Manual version 1.0
17. Hunsche, P.M. and E.A. Kuijpers, Diagnostics for use in Space: approximation of Schlieren using image processing, NLR CR 89237 L

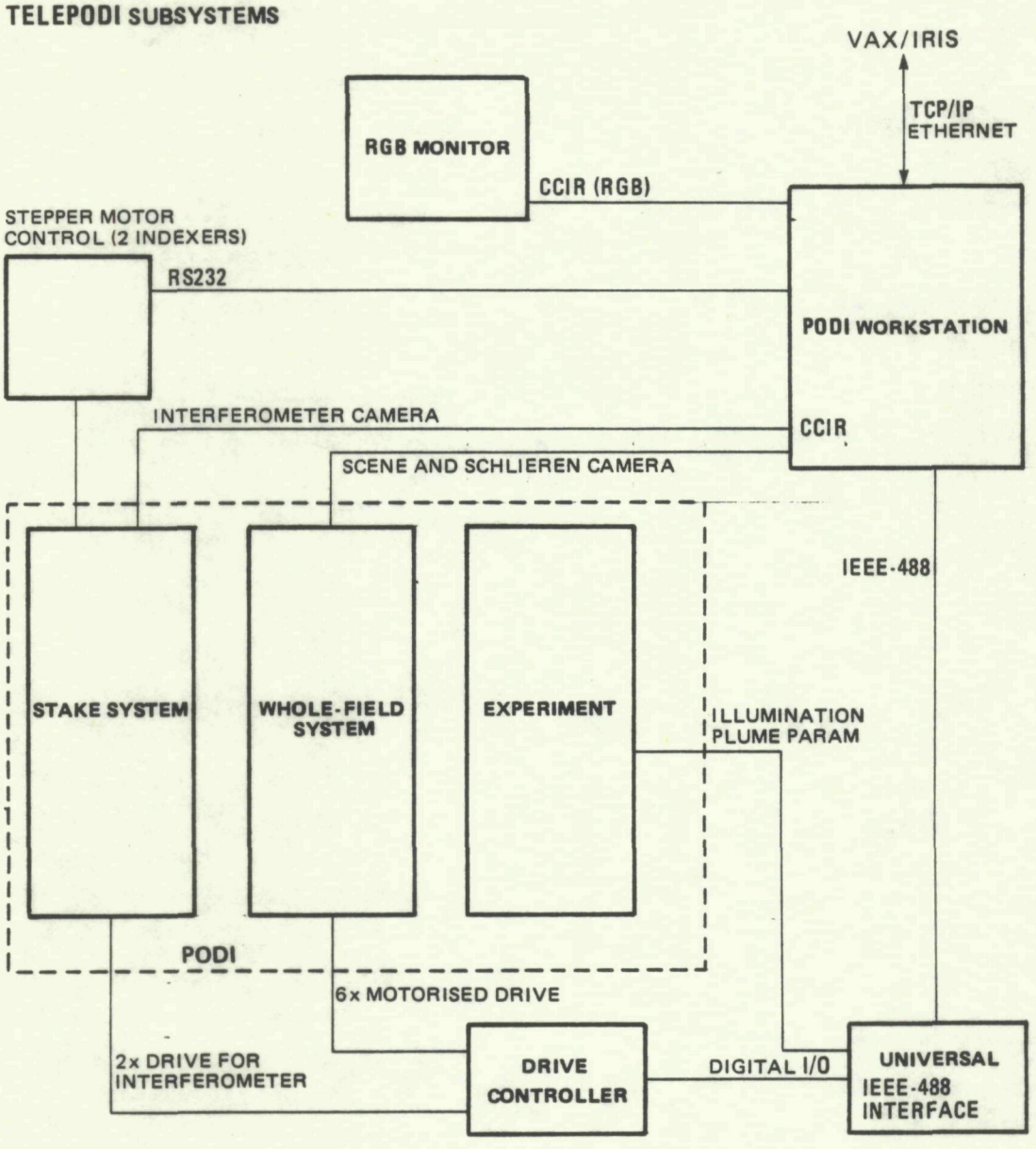


Fig. 2.1 Description of subsystems in TELEPODI

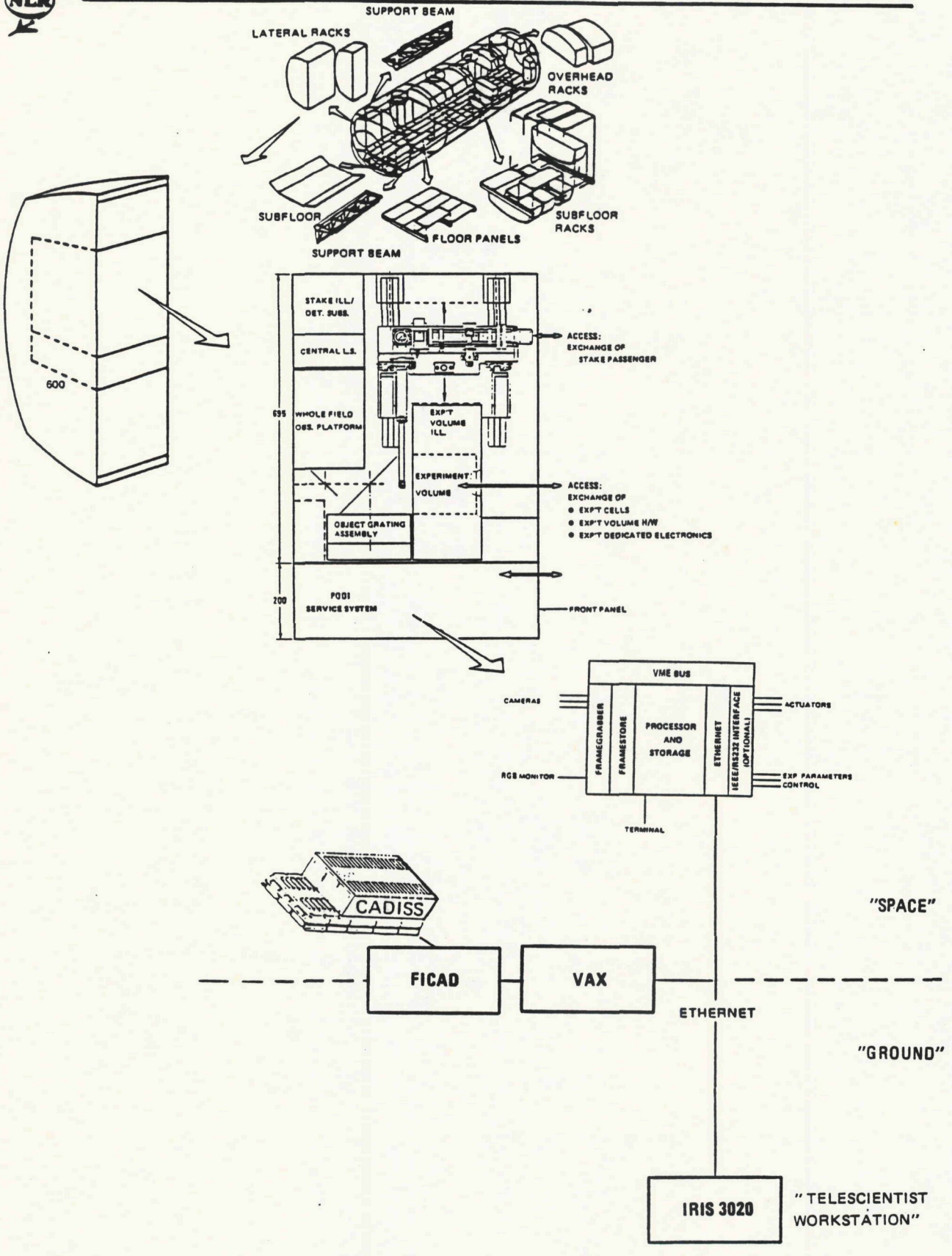
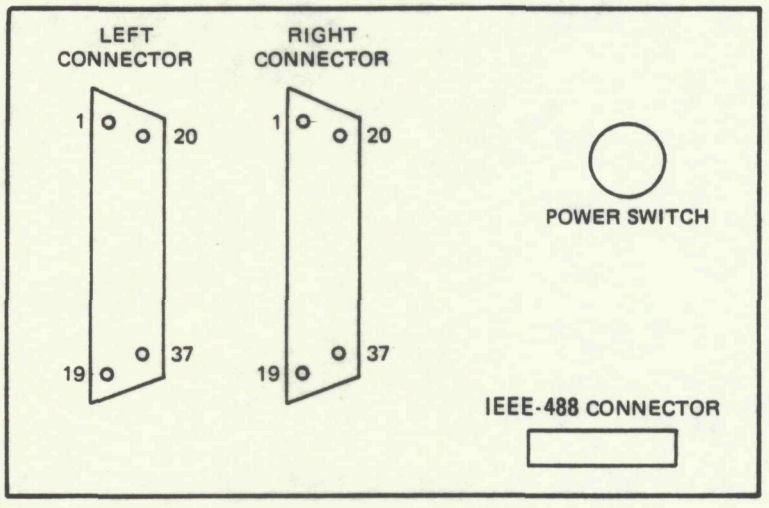
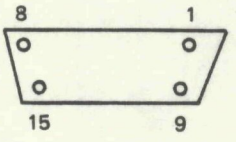


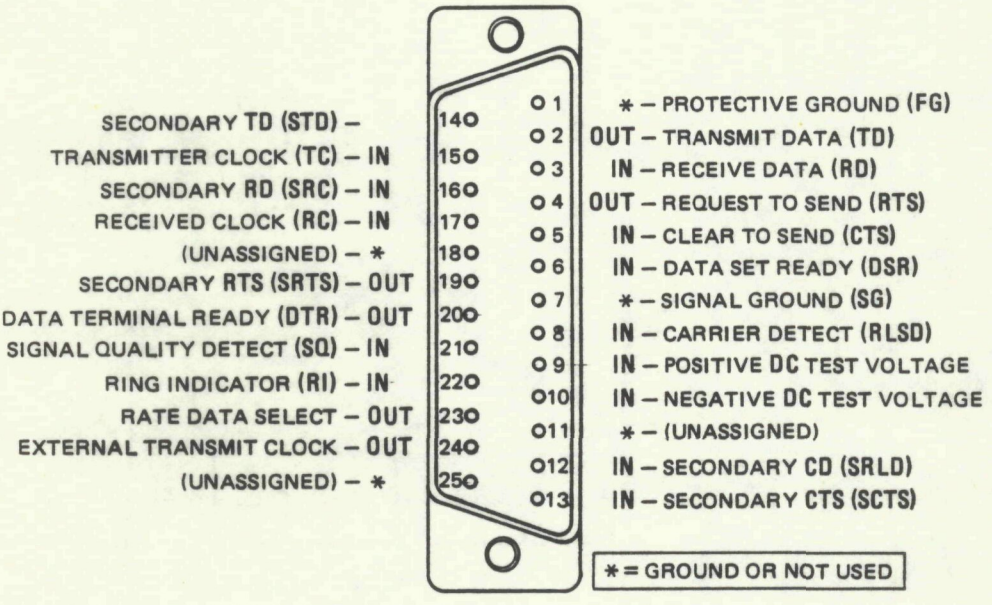
Fig. 2.2 TELEPODI set-up



UNIVERSAL IEEE-488 DIGITAL I/O CONNECTORS



NUMBERING ON FEMALE CABLE CONNECTOR FOR MOTORISED DRIVES



RS-232 CONNECTION WITH DB-25 PIN ASSIGNMENTS

Fig. 2.3 Some connectors in TELEPODI

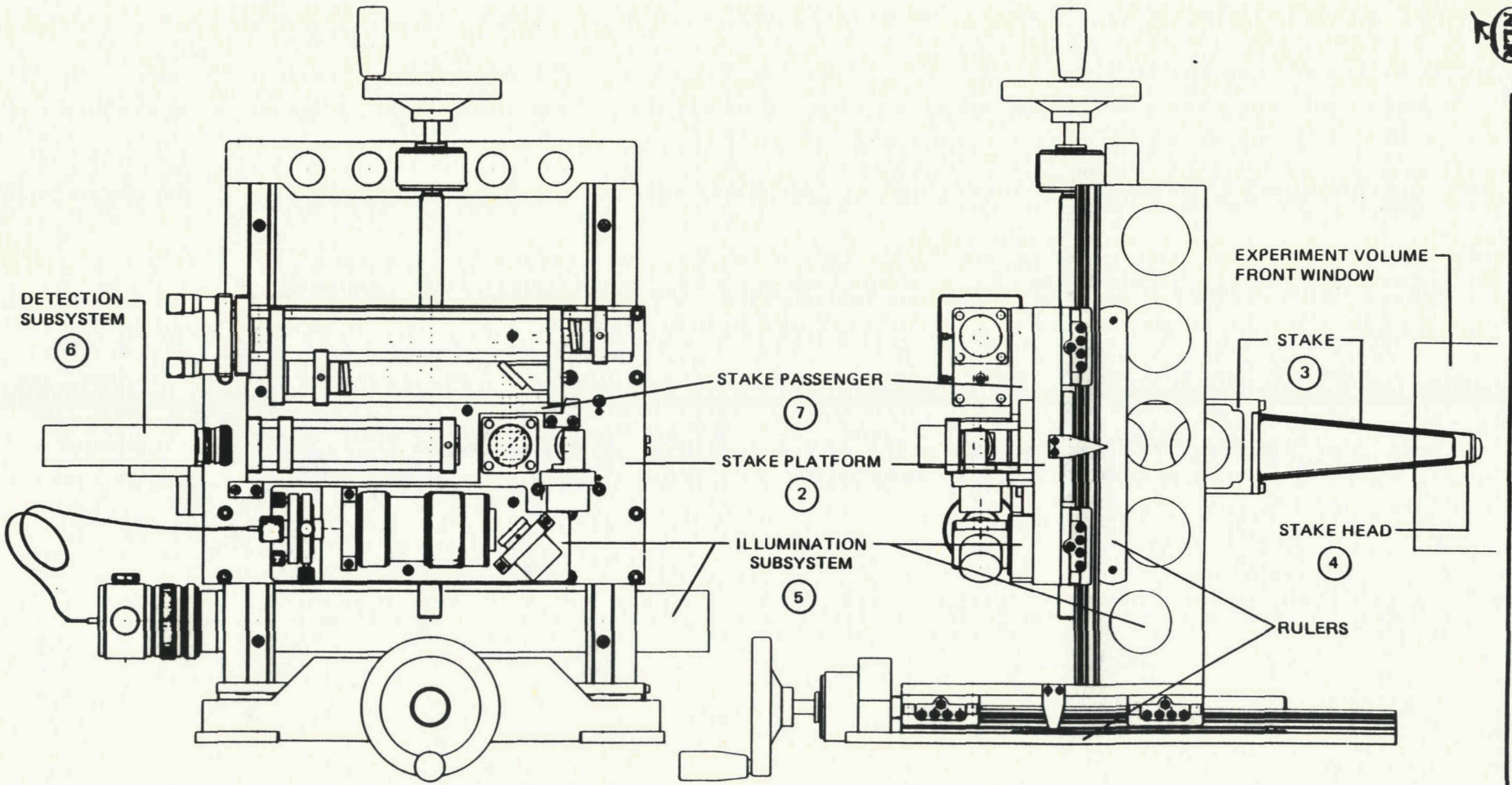


Fig. 3.1 Drawing of completely integrated stake system accomodating the interferometer passenger

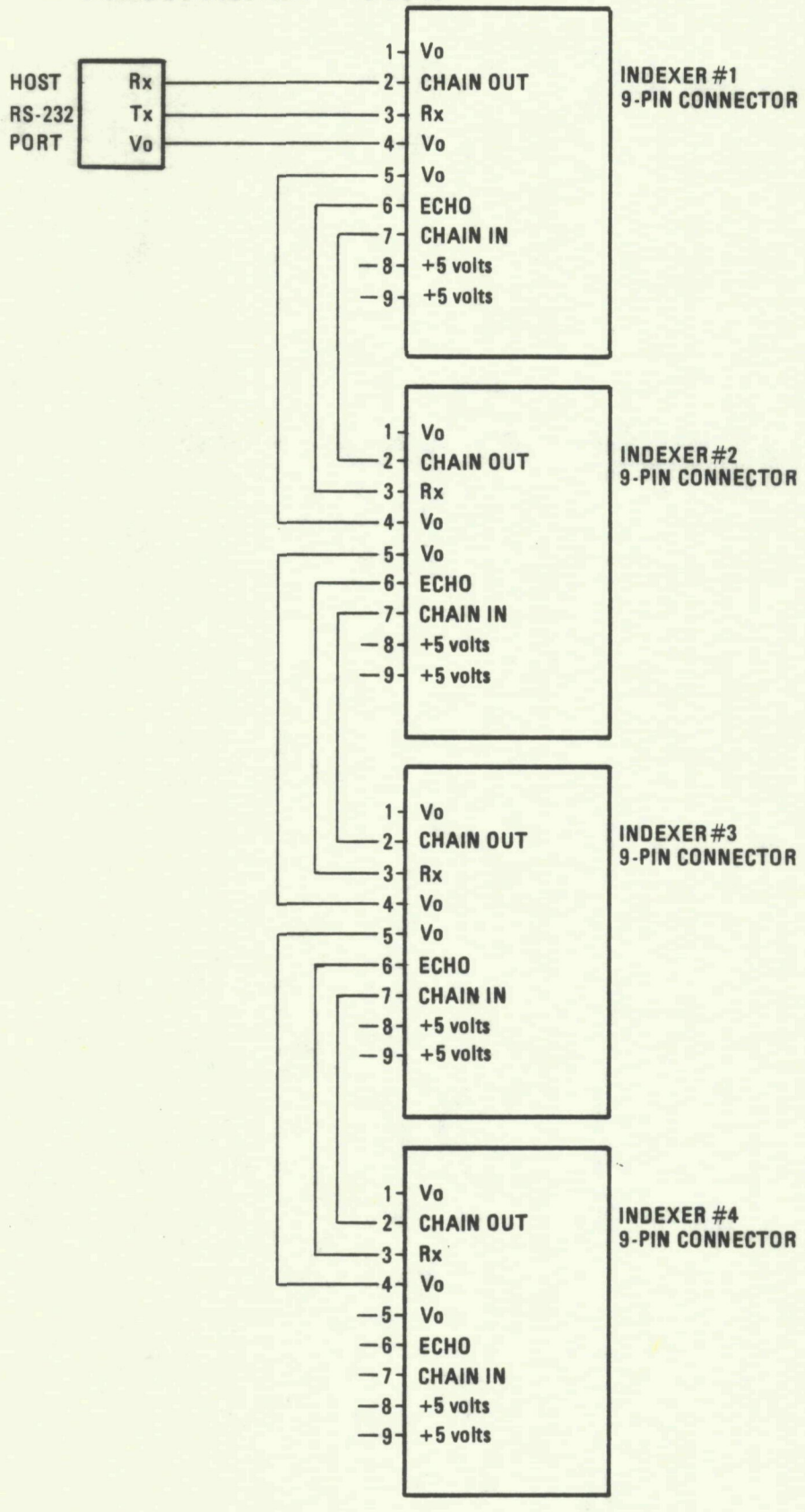


Fig. 3.2 Daisy chain configuration for stepper motor controllers

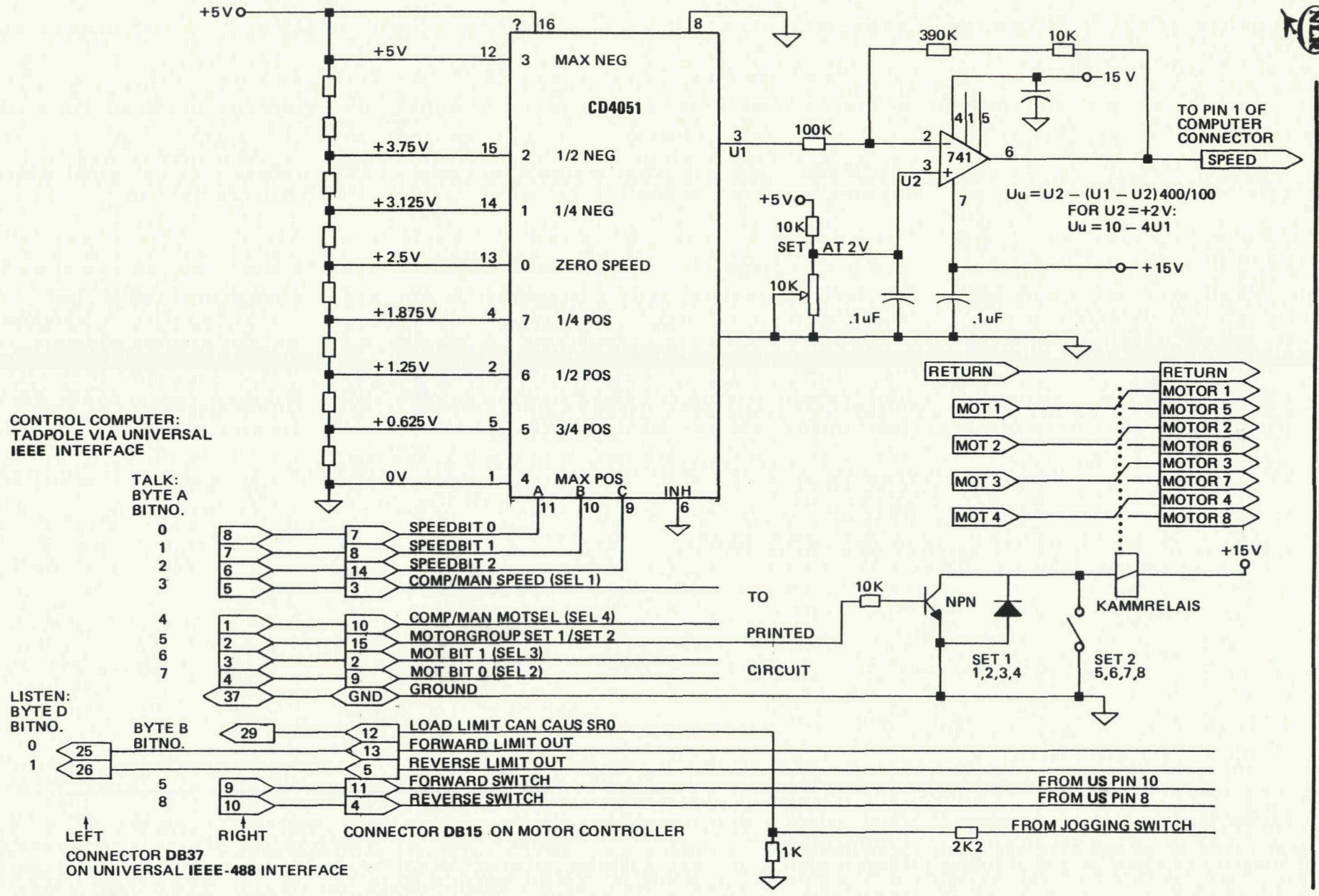


Fig. 4.1 Extension of motorised drive controller

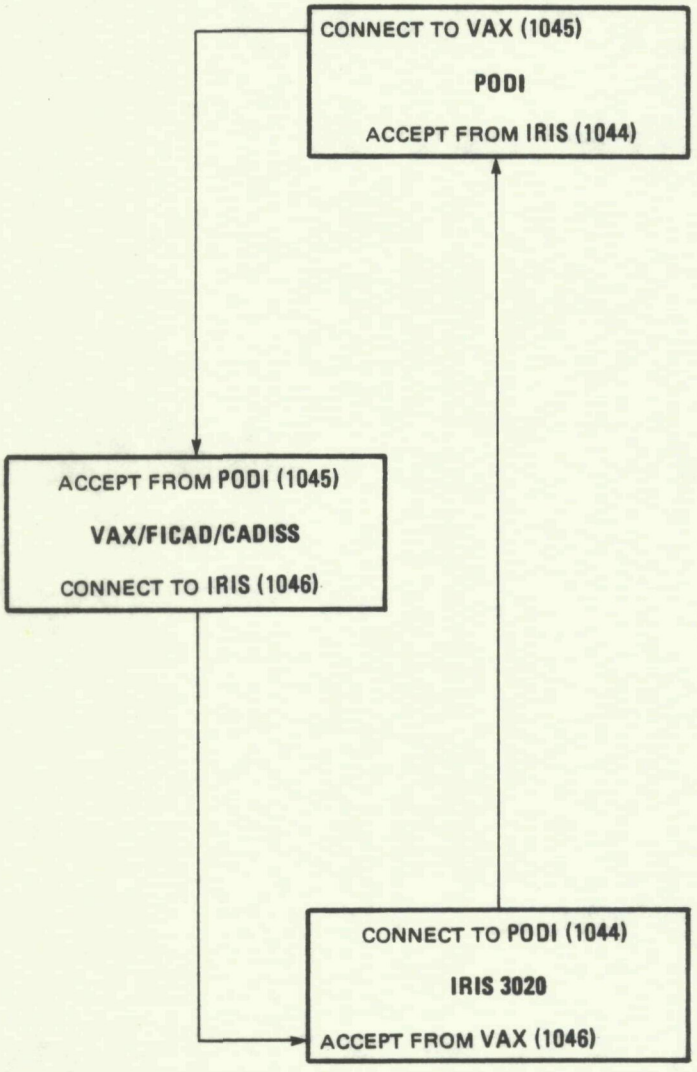
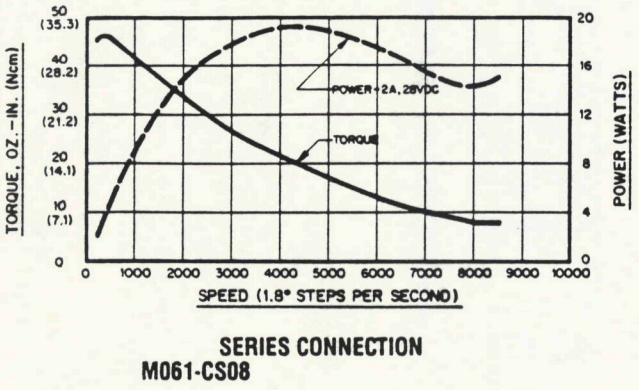


Fig. 6.1 Communication between PODI, IRIS and VAX in TELEPODI demonstration set-up at NLR

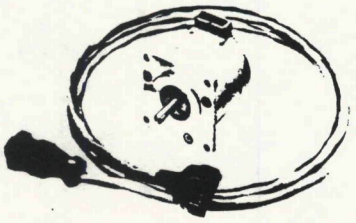
APPENDIX A

Description of stepper motors

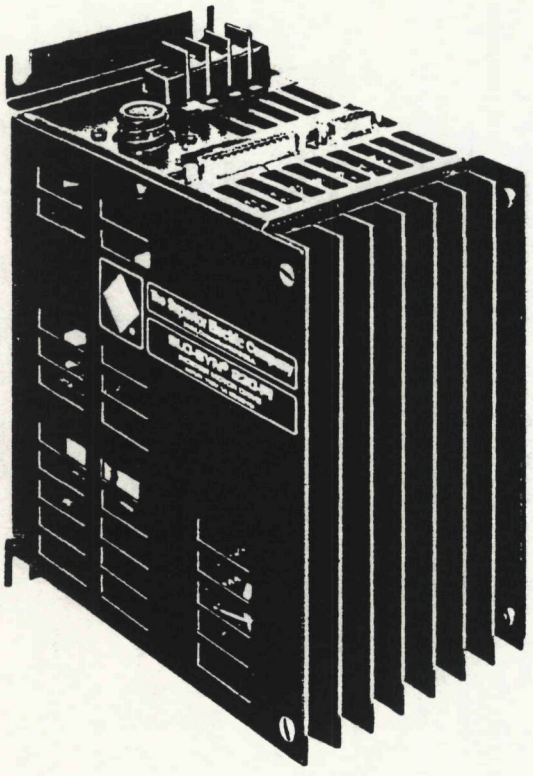
Model M061-CS08 stepper motor



SERIES CONNECTION
M061-CS08



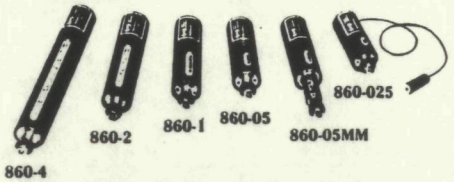
SLO-SYN micro series motion controls indexer models 230-PI



APPENDIX B
Description of motorised drives

Motorized Drives

U.S. Patent Nos. 4,467,250; 4,496,865



- **Easily interchangeable with micrometers in most Newport mounts**
- **Smooth-running, quiet DC motors**
  Continuously variable, load-independent speed
- **0.02 micron resolution**
- **>30 lb. (13 kg) load capability**
- **Up to 4 in. travel**
- **Compact, rugged construction**

860 Series Motorized Drives provide high-resolution linear motion for automated position control in micropositioning and measurement system applications. These micrometer replacement units are compatible with most Newport English and metric mounts and stages. They are powered by DC motors for smooth movement, high resolution and freedom from mechanical and acoustic noise. Ruggedly constructed for long life, they easily handle axial loads exceeding 30 lbs (13 kg). Versions with travel from 0.25 to 4 in. (6 to 102 mm) are offered.

When used with **860 Series Controllers** (opposite), these drives provide continuously variable, load-independent speed with dynamic braking for fast start/stop response. For extra jam-resistance, 860 actuators have cushioned end-stops that provide a slow deceleration at the ends of travel, and 860 Series controllers incorporate a patented limit/stall sensing design that automatically backs off the actuator when an overload condition occurs.

Submicron resolution is obtained with a precision-ground and electropolished stainless steel leadscrew, driven through a low-backlash reduction gear. A patented design constrains gearhead/motor rotation while mechanically "floating" the motor to decouple slight eccentricities from the rotating spindle. Spindle position indicators are provided with English and metric scales.

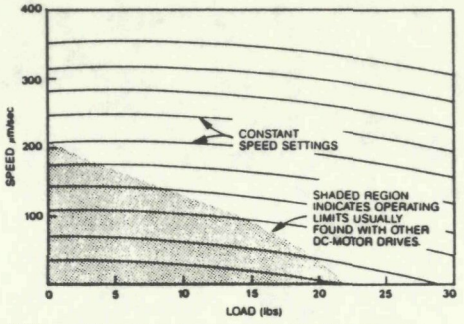
Specifications

Speed Range	40-400 micron/sec (Drives can be provided with speed ranges up to a factor of 30 faster or slower. Call Newport for pricing and delivery.
Resolution	0.02 micron
Max. Axial Load	35 lb. (15 kg)
Vacuum Compatibility	Special-order vacuum-compatible versions for operation to 10^{-6} are available. Versions for operation to 10^{-9} Torr are available on a semi-custom basis.

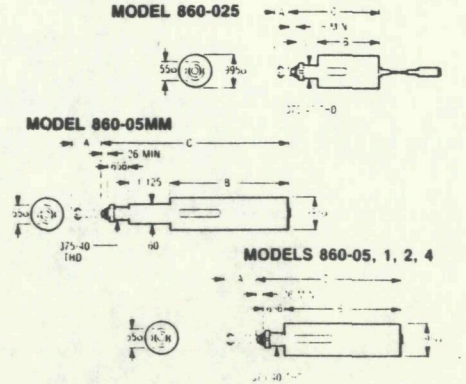
Motorized Drives

Travel (T) [in. (mm)]	Length (L) [in. (mm)]	Model
0.25 (6)	3.44 (87)	860-025
0.5 (13)	4.10 (104)	860-05
0.5 (13)	5.22 (133)	860-05MM*
1.0 (25)	4.66 (118)	860-1
2.0 (51)	5.72 (145)	860-2
4.0 (102)	7.91 (201)	860-4

* Used with Models 400 and 405 stages; 600A, 625A, 630A, 670, 675 and 605 Series Mirror Mounts; 960 and 965 beam-splitter mounts.



860 Series typical speed vs load characteristics illustrating 30 lb. load capacity over a wide speed range.

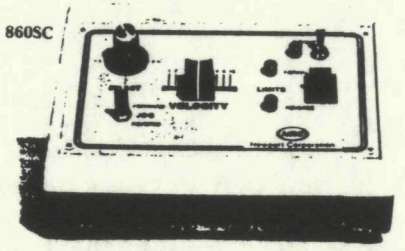


DIMENSION

MODEL	A	B	C
860-025	2.50	2.60	3.35
860-05	5.00	3.09	4.01
860-05MM	5.20	3.09	5.09
860-1	1.000	3.88	4.58
860-2	2.000	4.72	5.64
860-4	4.000	6.91	7.83

- **Continuously variable, load-independent speed control**
  Integral velocity servos
- **Patented limit/stall protection**
- **Dynamic braking for fast start/stop**
- **JOG function provides fine micro-motion control**
- **Control up to 4 drives**
- **Optional computer interface**
- **Compact; low cost**

This selection of velocity-servo controllers for **860 Series Motorized Actuators** provides continuously-variable, load-independent speed control with fast start/stop. Patented electronic limit/stall protection protects the motor from possible burnout at both extremes of travel and under stall conditions, so — unlike most DC motorized positioners — 860 Series motorized drives will not bind at an end-of-travel limit. The controllers also have circuitry which provides an automatic "kick" for a short time to ensure non-stick separation when reversing direction even at low speeds. 115 V/60 Hz power supplies are included (220 V/50 Hz versions are available on special order).



4-Drive Controller

Up to four 860 Series drives can be sequentially controlled with the **Model 860SC Servo Controller**. This hand-sized unit provides continuously variable speed control from 32 to 320 micron/sec using standard 860 Series drives. Motion is initiated by a FORWARD/STOP/REVERSE switch, enabling you to take advantage of the controller's dynamic braking capability to make fast changes of direction (or a sudden stop) without changing the velocity setting. A separate JOG switch provides a slow slew — with speed reduced to 0.1 to 1 micron/sec — allowing single-step movements approaching the drive resolution of 0.02 micron. A front-panel indicator shows end-of-travel or stall conditions.

Computer Interface Option: **Model 850SC-C** adds a computer interface connector to allow access to signals necessary for computer or remote electronic control, including direction, speed, limit indication and drive selection.

APPENDIX C
Programmers library for stake system

The following sections contain a description of the library for the stake system (version 12 April 1989).

C.1 High Level Primitives

FUNCTION
int center_stake ();

DESCRIPTION
This function moves the stake to the absolute center position. This is the position given by center_x_max, center_y_max. The default values are (10000, 10000).

The unit is 0.001 cm on rulers of stake (see Fig. 3.1). The new position becomes the home position. The maximum and minimum allowed displacements are updated. This function should be called before exiting the user program, in order to set the stake in a well defined position for future use. The function returns 1 on success or 0 on failure.

FUNCTION
int close_stake ();

DESCRIPTION
The channel ttyb, which was opened for read and write by open_ttyb in initialise_ stake, is closed. The function returns 0 on success and -1 on failures.

FUNCTION
int initialize_stake ();

DESCRIPTION

This function sends "L26 1" to both indexers giving EOT after all transmissions to the host. Both indexers are set to electrical home and in jog mode. All parameters are set to there default values in both indexers. The function returns 1 on success and 0 on failure.

FUNCTION

```
int move_stake_X (new_x);
int new_x;
```

DESCRIPTION

This function moves the stake in X direction. The new position becomes the home position. The maximum and minimum allowed displacements in X direction are updated. The distance from the center of the field of view is updated. The function returns 1 on success and 0 on failure.

FUNCTION

```
int move_stake_Y (new_y);
int new_y
```

DESCRIPTION

This function moves the stake in Y direction. The new position becomes the home position. The maximum and minimum allowed displacements in Y direction are updated. The distance from the center of the field of view is updated. The function returns 1 on success and 0 on failure.

FUNCTION

```
int reset_origin ();
```

DESCRIPTION

Set origin to current absolute position. Update maximum and minimum allowed displacements in X and Y direction.

FUNCTION

```
int set_origin (new_x, new_y, choice);
int new_x, new_y, choice;
```

DESCRIPTION

The stake is moved to the required origin in the center or in the lower left corner of the field of view. The maximum and minimum allowed X and Y displacements are updated. The default origin is at (10000, 10000). See figure 3.1. The function returns 1 on success and 0 on failure.

FUNCTION

```
int set_stake_X (new_x);
int new_x;
```

DESCRIPTION

This function sets the stake to the requested absolute X position with respect to the current origin. The new position becomes the home position. The maximum and minimum allowed displacements in X direction are updated. The distance from the center of the field of view is updated. The function returns 1 on success or 0 on failure.

FUNCTION

```
int set_stake_Y (new_y);
int new_y
```

DESCRIPTION

This function sets the stake to the requested absolute Y position with respect to the current origin. The new position becomes the home position. The maximum and minimum allowed displacements in Y direction are updated. The distance from the center of the field of view is updated. The function returns 1 on success and 0 on failure.

C.2 Keyboard I/O for test and initialisation

FUNCTION

```
int make_origin ();
```

DESCRIPTION

The user is asked to give the position of the stake as given by the rulers on the stake system (see Fig. 3.1). Then the user can choose the position of the origin in the center or in the lower left corner of the field of view. The routine calls set_origin to update the origin. The function returns 1 on success and 0 on failure.

FUNCTION

```
void print_stake ();
```

DESCRIPTION

Print the comment associated with the last given command. Print the contents of the stake_buffer. If the first character of a buffer line is 'L', then add the proper L comment.

FUNCTION

```
void print_position (row, col);
int row, col;
```

DESCRIPTION

Print absolute X and Y position with respect to current origin on screen at row "row" and column "col".

FUNCTION

```
void read_new_position (row, col);
int row, col;
```

DESCRIPTION

Ask for new X position at row "row" and column "col" and for new Y position at row "row + 2" and column "col". Get new position, within current limits, from keyboard.

FUNCTION

```
void read_new_displacement (row, col); int row, col;
```

DESCRIPTION
Ask for X displacement at row "row" and column "col" and for Y displacement at row "row + 2" and column "col". Get displacements, within current limits, from keyboard.

C.3 Low level primitives for the stake system

These functions are called from the user functions. They require global parameters from the include file stakelib.h and should not be called by user written routines.

FUNCTION
```
#include stakelib.h
int move_stake (motor, new_position, current_position);
char motor[];
int new_position;
int current_position;
```

DESCRIPTION
For this motor, calculate the required displacement from the new and the current position. Dependant on the sign of the required displacement, set a new clockwise (L18) or counterclockwise (L19) travel limit and start the motor in clockwise resp counterclockwise direction. If necessary go step by step to the new position. Read the new motor position and save this position in stake_buffer. The function returns 1 on success and 0 on failure.

FUNCTION
```
#include stakelib.h
int open_stake ();
```

DESCRIPTION
Open ttyb for read and write, with 9600 BAUD, echo enabled and XON/XOFF protocol. If open fails, send a message. The function returns 1 on success and 0 on failure.

FUNCTION

#include stakelib.h

int read_stake ();

DESCRIPTION

Until EOF received, read the indexer response. Ignore ':', '=', XON and XOFF. If a carriage return is received increment the output buffer pointer. Stop reading if the input buffer is empty. The function returns 1 on success and 0 on failure.

FUNCTION

#include stakelib.h

int set_ttyb ();

DESCRIPTION

Set tty terminal to enable start/stop in and output control. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. The system will transmit START/STOP (XON/XOFF) characters when the input queue is nearly empty/full. Echo is enabled. The baud rate is 9600. A character contains 8 bits. The function returns 1 on success and 0 on failure.

FUNCTION

#include stakelib.h

int stake_command (motor, instruction);

char motor[];

char instruction[];

DESCRIPTION

Combine the motor number and the instruction into the indexer command, add a carriage return and a linefeed. Use the write function to send the indexer command to ttyb. If the number of bytes returned by write is not the number of bytes send, give an error message. Wait a few milliseconds before continuing, to allow the indexer to get ready to acknowledge. Read the output of the indexer and save the result in the buffer "stake_buffer" defined in stakelib.h.

APPENDIX D

Programmers library for control via IEEE-488

The following functions are available in the version of 12 April 1989.

D.1 Control of IEEE-488 interface

FUNCTION
void clear_GPIB ();

DESCRIPTION
Send IFC to clear the GPIB.

FUNCTION
void initialize_GPIB ();

DESCRIPTION
Initialise the interface to the motorised drives. Set the interface in on-line mode. Clear the interface.

FUNCTION
void terminate_GPIB ();

DESCRIPTION
terminate GPIB activity by clearing the GPIB. Set the GPIB in off line mode and disable the interface.

The following functions are the basis for message communication via GPIB. A full description can be found on the source code accompanying the GPIP-1014 P board.

FUNCTIONS
int ibrd(buf,cnt)
char buf[];
int cnt;

```
int ibwrt(buf,cnt)
char buf[];
int cnt;

int ibcmd(buf,cnt)
char buf[];
int cnt;
```

DESCRIPTION

The function "ibrd" is used to read up to cnt bytes of data from the GBIP into buf.

The function "ibwrt" is used to write cnt command bytes from buf to the GPIB.

The function "ibcmd" is used to write command bytes from buf to the GPIB. This function needs to be called for addressing before using the functions "ibrd" and "ibwrt".

D.2 Control of motorised drives

FUNCTION

```
#include table.h
char motor_limit ()
```

DESCRIPTION

The routine reads the motor status which is returned in 4 bytes. The third byte gives the status of the motor limit. If the motor is at the forward limit position the status = 1 and in reverse limit position the status = 2. Else the function returns 0.

FUNCTION

```
#include table.h
void set_motor (motor, speed);
```

```
int motor;
int speed;
```

DESCRIPTION

The user can call this function with the motor number (1 - 8) and the speed number (1 - 8). The function finds the bit combination for the required motor from the array MOTOR[] (most significant nibble) and performs a binary OR with the bit combination for the requested speed found from the array SPEED[] (least significant nibble). The result is saved in character array command[]. The direction of motion is derived from bit 7 (least significant bit) of the speed. If this bit is 1 then the direction is forward which is represented by setting bit 3 of command[1]. If bit 7 is 0 the direction is reverse and bit 2 of command[1] is set. The byte array is send via the uni_bus with ibwrt. The motors are assigned by:

	NR	BIN	DEC	HEX
MOTOR[0]	1	00010000	16	10
MOTOR[1]	2	10010000	144	90
MOTOR[2]	3	01010000	80	50
MOTOR[3]	4	11010000	208	D0
MOTOR[4]	5	00110000	48	30
MOTOR[5]	6	10110000	176	B0
MOTOR[6]	7	10110000	112	70
MOTOR[7]	8	11110000	240	F0

```
              \ /|
               |  ----- computer control bit.
                ------- motor selection bits.
```

The speeds are:

	NR	BIN	DEC	HEX	DESCRIPTION
SPEED[0]	1	00001101	13	0D	full negative speed.
SPEED[1]	2	00000101	5	05	1/2 full negative.
SPEED[2]	3	00001001	9	09	1/4 full negative.
SPEED[3]	4	00000001	1	01	zero speed.
SPEED[4]	5	00001111	15	0F	1/4 full positive.
SPEED[5]	6	00000111	7	07	1/2 full positive.

```
SPEED[6] = 7 = 00001011 = 11 = 0B  3/4 full positive.
SPEED[7] = 8 = 00000011 =  3 = 03  full positive speed.
                        | ||
                        | | ----- computer control bit.
                        |  ------ sign bit.
                        \ /
                         |
                          ------- speed selection bits.
```

The byte obtained for speed is combined using an bitwise-or operation to obtain the first byte which is sent to the IEEE-488 interface. A second byte will contain derived direction information as discussed above.

In the include file table.h the speeds are defined as:

FULL_REVERSE	1
HALF_REVERSE	2
QUAT_REVERSE	3
STOP	4
QUAT_FORWARD	5
HALF_FORWARD	6
THRE_QUA_FRW	7
FULL_FORWARD	8

See figure 4.1.

FUNCTION

```
void tset_motor (motor, speed, time);
int     motor, speed, time;
```

DESCRIPTION

This routine gets the current processor time using the function "getetime". Then starts the required motor with the requested speed. The time is read until the demanded time has elapsed. Then the motor is stopped.

FUNCTION

```
long getetime ();
```

DESCRIPTION

This routine returns the current elapsed real time in milliseconds. The accuracy is 1/16 s.

D.3 Control of plume

For the plume experiment the following function is proposed. This function is not implemented.

FUNCTION

```
#include table.h
void plume (condition, msec);
int condition;
int msec;
```

DESCRIPTION

Calling this routine will cause a puls of "msec" milliseconds to be send to the plume. If condition is 1 a high amplitude puls will be sent, else if condition is 0 a low amplitude puls is sent.

APPENDIX E

Library for terminal control

The routines in this library perform terminal independent i/o functions based on the "termcap" capabilities in UNIX. I.e. Clear the screen, goto a certain line and column, and goto a line and column and clear to the end of the line. The following routines are available in the version of 12 April 1989.

FUNCTION
```
void clear_screen ();
```

DESCRIPTION
The screen is cleared.

FUNCTION
```
void clear_eol (row, col);
int row, col;
```

DESCRIPTION
Goto row "row" and column "col" and clear to end of line. Row 0 and column 0 is in the upper left corner of the screen.

FUNCTION
```
void goto_row_col (row, col);
int row, col;
```

DESCRIPTION
Set the cursor to row "row" and column "col". Row 0 and column 0 is the upper left corner of the screen.

multihouse

Appendix to TCL-IMAGE Product Description.

TCL-IMAGE FUNCTIONS

Below an overview is given of the present (October 1988) functions within TCL-IMAGE V 4.3. This list will be extended on a regular basis, please call us for the latest version.

multihouse

TCL-IMAGE OVERVIEW OF FUNCTIONS
Version 4.3 dated 29 september 1988

multihouse

multihouse

multihouse

TCL-IMAGE OVERVIEW OF FUNCTIONS
Version 4.3 dated 29 september 1988

multihouse

APPENDIX G

Description of demonstration program using pseudo-language

G.1 Program on PODI

The PODI program running on the PODI workstation is an intermediate between the TELEPODI setup and the IRIS computer. On the IRIS computer runs also a program (source name teleiris) which is used by the telescientist to give commands to and to receive pictures from the PODI computer. As the transfer rate between PODI and IRIS is only approximately 10 Kbytes/sec while the transfer rate PODI/VAX and VAX/IRIS is much higher, the pictures are sent to the IRIS computer via the VAX. The program running on the VAX (source televax) only receives and sends 8 blocks of 1024 bytes. Optional another program can be used which uses the capabilities of CADISS for compression and decompression of images.

The program on the PODI computer performs the following actions (between parentheses the C-functions are given):

```
Accept a connection to a remote host with (accept_IRIS)
and wait for anyone to connect.
  Then try to connect to the VAX (connect_VAX).
  If the connection is granted
    Initialise the table motors (initialise_table)
    Initialise the stake systems (initialise_stake)
    Initialise the video system (init_video)
    Set the origin of the stake in the center (set_origin)
    Repeat the next instructions
      Receive a command from IRIS (get_IRIS_podi_command)
      Perform the requested function (handle_podi_command)
      These function can be:

        Stake commands:
          Make an absolute move in X and/or Y direction
          Reset stake to center position

        Interferometer commands:
```

Move motor 1 during a given time at a given speed
Move motor 2 during a given time at a given speed

Whole schlieren commands:
Move motors 3, 4, 5, 6, 7 or 8 during a given time
at a given speed

Video commands:
Send a picture from camera 1, 2 or 3 to the IRIS

End command:
Exit Repeat Until Loop

Until an end command is received
Set the stake system in the center position with "center_stake"
Send an end instruction to the VAX
Close the connections with the VAX and the IRIS
Exit the program

G.2 Program on VAX

The program on the VAX computer performs the following actions (not using CADISS) :

Accept a connection to a remote host (net_listen)
and wait for anyone to connect.
Then try to connect to the IRIS (net_connect)
If the connection is granted
Repeat the next instructions
Read 8 blocks of 1024 bytes from PODI (net_read)
Write 8 blocks of 1024 bytes to the IRIS (net_write)
If less than 8 * 1024 bytes are received
send them anyway in order to keep the communication running
Until an end instruction is received
Close the connections with PODI and the IRIS

Exit the program

G.3 Program on IRIS 3020

This program on the IRIS computer performs the following actions:

Try to connect to PODI workstation (connect_PODI)
Then accept a connection to a remote host (accept_VAX)
and wait for anyone to connect.
If the connection with the VAX is established
Initialise a window of 512 * 512 pixels on the graphics screen
Set double buffer mode
Make a colour map
Send a command to PODI to send a picture from camera 1

Using the menu possibilities in the graphics library
the functions as discussed in sections G.1 are selected.
If the user wants to select a new position for the stake, a cross wire is drawn in the image which can be moved using the mouse. If the desired position is selected, the position of the cross on the screen is converted to a command for the stake. For the implementation of this feature double buffering is used.