An automated polarimeter and its use in the study of active galaxies

Stockdale, D.P.

An Automated Polarimeter and its use in the study of active galaxies.

D.P.Stockdale B.Sc. (Applied Computing)

A thesis submitted to the University of Durham for the degree of Doctor of Philosophy

The copyright of this thesis rests with the author. No quotation from it should be published without his prior written consent and information derived from it should be acknowledged.

Department of Physics

September 1996.

In memory of my

father.

"forever with me"

Abstract

In this thesis I present the design and development of an automated polarimeter for use in mapping the percentage levels and position angles of linearly polarized light from extended astronomical objects. The polarimeter is controlled from a personal computer that is running a UNIX operating system and controls not only the instrument, but the CCD camera as well.

The second chapter of the thesis consists of a description of how the polarimeter works, the principles behind the optics, the mechanics and the electronics. The third chapter describes the software that controls the functional units within the polarimeter to the required accuracy demanded of a scientific application.

The fourth and fifth chapters of the thesis address some of the scientific issues that the polarimeter has been used to clarify. There is a brief presentation of the phenomena of starburst galaxies and the generation of galactic-scale winds, often called superwinds. Polarization results and their interpretation for three starburst galaxies are presented.

Contents

Table of Figures

Preface

The contents of this thesis describe the work carried out by the author between 1989 and 1996, under the supervision of Dr. S. M. Scarrott. This period included four years as a part-time student while working as an electronics technician in the Electronics Workshop and one year as a part-time student while working as Experimental Officer in the Durham Polarimetry Group.

The work is chiefly that of the author with the exception of the section on polarization of starburst galaxies which is in collaboration with other members of the polarimetry group. All of the data for this section was collected by the author along with other members of the group using the Mark IV polarimeter described in this thesis. The data have been published by the author in collaboration with Dr. S. M. Scarrott, Dr. P. W. Draper, Dr. R. D. Wolstencroft and Dr. C. Done

None of the material contained within this thesis has been previously submitted for a degree in this or any other university.

1. Introduction

Electromagnetic radiation emitted from astronomical objects can be used to measure many different astrophysical processes that occur in stars, nebulae. We have the ability to measure radiation ranging from γ-rays through to radio waves using either telescopes based on the earth, or for wavelengths that lie outside the earth's atmospheric window, telescopes mounted on satellites in space. Using data collected from all possible sources, astronomers attempt to piece together this information to give a better insight into how stars and galaxies are formed, evolve and end, as well as more fundamental questions such as how the universe began. The Durham polarimeter is an instrument that can be mounted to a ground-based telescope and is used to provide polarmetric data that helps us discover certain information that cannot easily be gleaned from other studies.

The Durham polarimeter operates in the optical wavelengths of the electromagnetic spectrum and is used to measure the polarization properties of the light that is emitted from extended astronomical objects. The light can polarized for various reasons (section 1.2) and the polarization information provides us with a method of locating illuminators (directly visible or hidden), determining the dust distributions and geometry of various objects and mapping the magnetic field in systems ranging from star-forming regions to galaxies.

There have been a total of four Durham polarimeters to-date (see section 1.1) and the main reason for building the current polarimeter was a desire to integrate the three main observing functions into an easy-to-use system that would give optimum results in the most efficient way possible. This latest polarimeter would be controlled from a

single workstation computer that would also operate the CCD camera collecting data, and reduce the data so that it could be analysed at the observing site.

The purpose of this thesis is to describe the work carried out in writing the new control system software for the polarimeter and the integration of the control system with the other subsystems. Results for polarmetric observations for a selection of starburst galaxies are presented and discussed.

1.1 History of the Durham polarimeter

The first polarimeter (Mark I) was based on a small optical bench mounted within a wooden case and required the observer to open the case side and rotate a half wave plate by hand to change the analyser position. This first version was used to test the principles at the RGO and operated between the years 1973 - 1974. The second polarimeter (Mark II) was more engineered and the observer could select analyser positions using a notched thumb wheel on the side of the instrument case rather than open up the side of a wooden box. This version of the Durham polarimeter was used on many different telescopes between the years 1975 - 1987 and again provided a lot of useful data that went into many publications.

The first automated polarimeter (Mark III) was the third instrument of its kind to be manufactured in the workshops of the Physics Department at the University Durham and allowed the observer to change the analyser position and other settings such as the filter, using an Apple II computer. This polarimeter was operated between the years 1987 - 1992 and was far more efficient at collecting data due to the fact that everything was controlled from the telescope control room. The main problem with this polarimeter was the use of multiple computers each of which controlled a separate function. Since there was no method of interlock between the two computers, there was nothing to prevent the observer from accidentally moving one of the functional units within the polarimeter while taking an exposure. The Mark IV polarimeter

described here, was constructed in 1991 and was commissioned in 1992, and was built to address these earlier problems. Since the commissioning run it has been used on twelve observing trips without any major problems.

1.2 Polarization Mechanisms

The Durham polarimeter, as previously mentioned, is an instrument for measuring linearly polarized light from astronomical objects. There are three main polarization mechanism that give rise to linear polarized light, scattering, extinction and synchrotron radiation.

1.2.1 Scattering

The process of scattering occurs when an electromagnetic wave impinges on electrons, atoms, molecules or dust grains and interacts with the bound electron cloud, imparting energy and setting it into vibration (see Figure 1-1). The level of polarization is a function of the scattering angle with the maximum values of polarization occurring

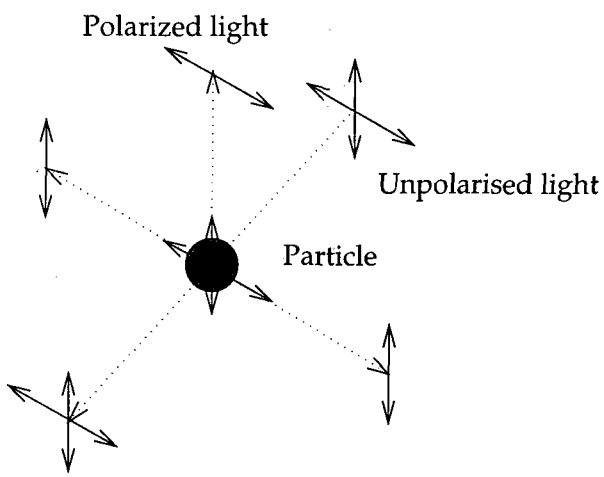


Figure 1-1 - Scattering

with angles near to 90°. The orientation of the polarization is perpendicular to the plane defined by the incident and scattered light rays.

1.2.2 Dichroic Extinction by Aligned Non-Spherical Grains

Dust grains in the interstellar medium are elongated and consist of some form of paramagnetic material. In a magnetic field the grains spin and align themselves with the field in accordance to the theory proposed by Davis and Greenstein (1951). The grains align with the long axis perpendicular to the direction of the magnetic field due to the interaction between the dipole moment of the grain and the magnetic field (see Figure 1-2). There are certain prerequisites for this to occur such as the strength of the magnetic field and grain temperature. It is important that the gas temperature of the ISM is much greater than the temperature of the grain so that the magnetic oscillations that provide the alignment torque dominate over the thermal fluctuations of the dipole. When initially unpolarized light passes through an area containing aligned grains, it is found that the transmitted light has a net polarization with its orientation parallel to the component of the magnetic field perpendicular to the line of site. The polarization occurs due to the grain preferentially extinguishing the vibrations of the incident light

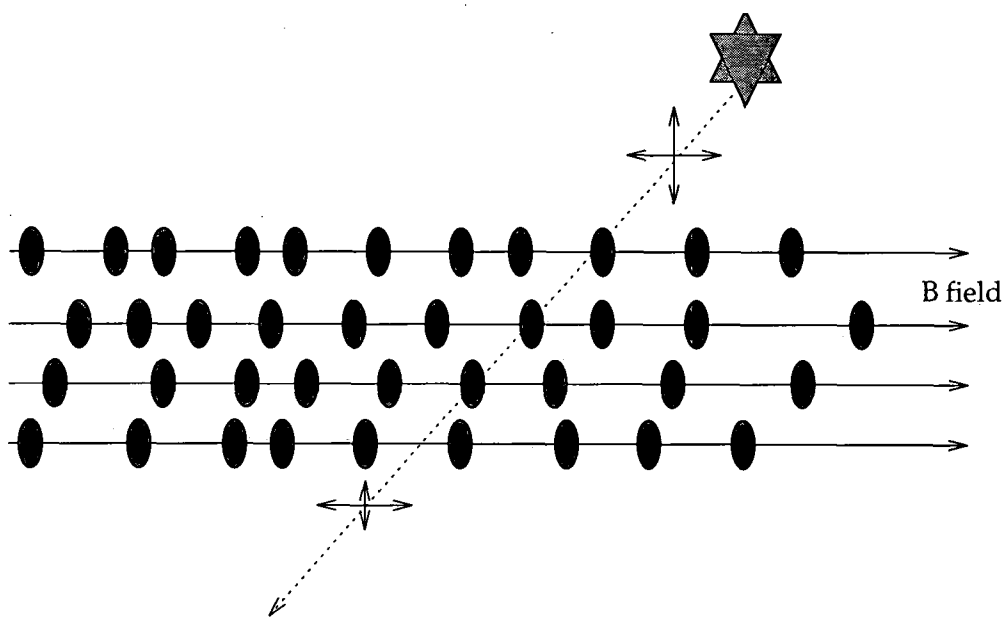


Figure 1-2 - Extinction

in a direction parallel to the long axis of the aligned grain.

1.2.3 Synchrotron Radiation

When high energy charged particle interact with a magnetic field their trajectory becomes helical about the field lines. As these particle gyrate they emit electromagnetic radiation known as synchrotron radiation where the E-field is parallel to the direction of acceleration and therefore perpendicular to the direction of the magnetic field (see Figure 1-3). Synchrotron emission is normally seen in the radio bands due to the energy levels involved, although occasionally emission in the visible wavelengths occur from high energy sources such as the Crab Nebula and the jets of 3C373 and M87.

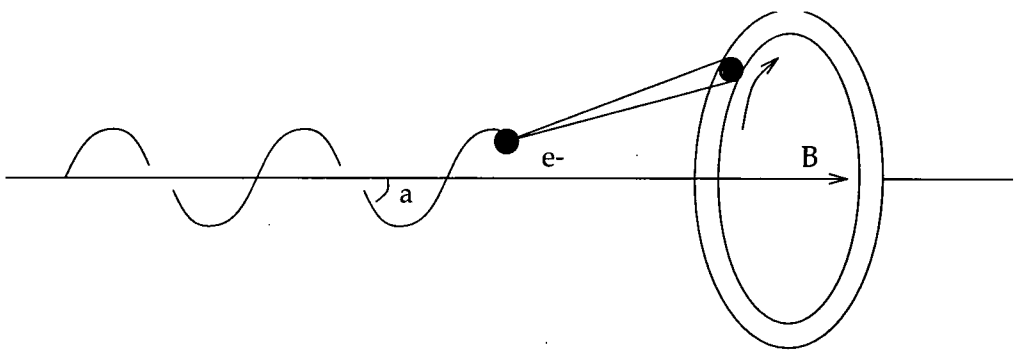


Figure 1-3 - Synchrotron Radiation

2. New Durham polarimeter

2.1 Introduction

The Durham polarimeter is an instrument designed to be mounted onto telescopes to spatially map the polarization of light from astronomical objects. The instrument acts as a polarization analyser and images of the objects in various polarization states are recorded onto some electronic medium so that they can be interpreted at a later time. The original Durham polarimeters are well documented and described in detail by Pallister(1976), Axon(1977) and Warren-Smith(1979), and have changed very little with respect to the principles of operation since then. Scarrott et. al. (1983) describe the workings of the polarimeter using an electronographic camera as the detector, whereas Draper(1988) describes the charge coupled device (CCD) detector. A complete description of the current system is included to give a completeness to the authors work.

This chapter describes the principles of how the Mark IV polarimeter functions both in terms of the optics and the mechanics, as well as the design of the electronics. The final section of the chapter describes various system configurations that have been used and talks about them with respect to the advantages and disadvantages to the observer.

2.2 Optical Design

The Durham polarimeter optical system is based on the ideas put forward by Pickering(1873) and later by Ohman(1939). The Durham polarimeter uses the dual channel method of operation whereby two orthogonal polarised states of light are measured simultaneously. By using this method, variations in the sky brightness and transparency need not concern the observer as the effects are cancelled out during the data reduction process. An optical diagram of the Durham polarimeter is shown in Figure 2-1.

At the heart of the polarimeter lies the analyser which in the case of the Durham polarimeter is a Wollaston prism (Figure 2-2). The prism is constructed of two optically contacted wedge shape quartz crystals with their optical axes at 90° to each other. Light which is incident on the prism is split into two beams diverging at a nominal angle of 1°, one beam being polarised parallel to the plane of divergence and the other perpendicular to it. To calculate polarisation levels and positions at least two sets of measurements are required, one set at some nominal angle and another set at a position angle varying from the first by 22.5°. This is achieved by rotating the plane of polarisation of the light entering the instrument using an optical retarder (Figure 2-3 - Half Wave Plate). The retarder introduces a phase difference of 180° between the parallel and perpendicular components of the incident light and is therefore known as the half wave plate. The retarder rotates the plane of polarisation of the linearly polarised light by an angle 2θ where θ is the angle subtended by the incident polarisation relative to the optical axis of the retarder. The retarder is constructed from three pairs of optically active quartz and magnesium fluoride plates and is superachromatic to 1% over a wavelength of 310 - 11000nm

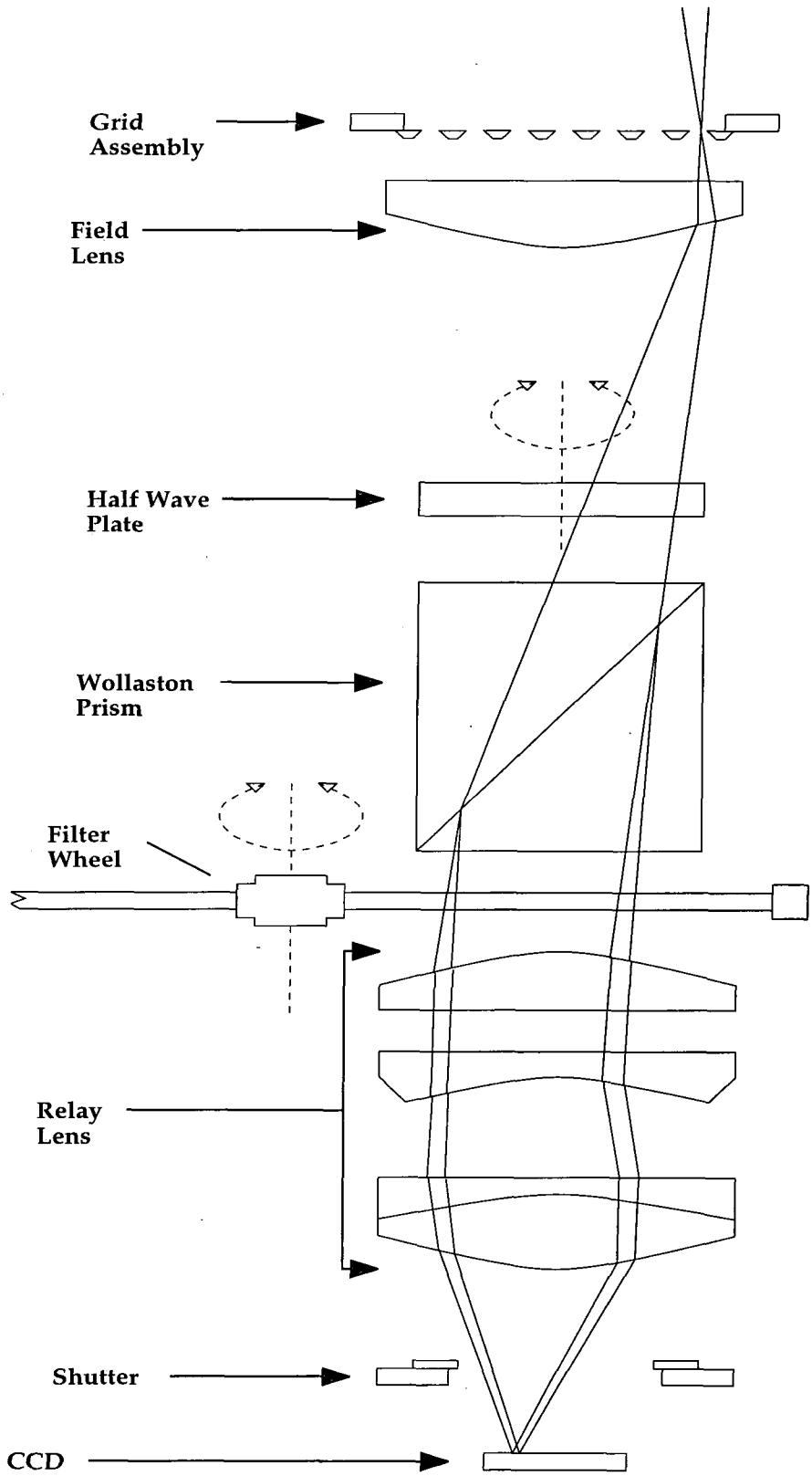


Figure 2-1 - Optical Diagram of Polarimeter

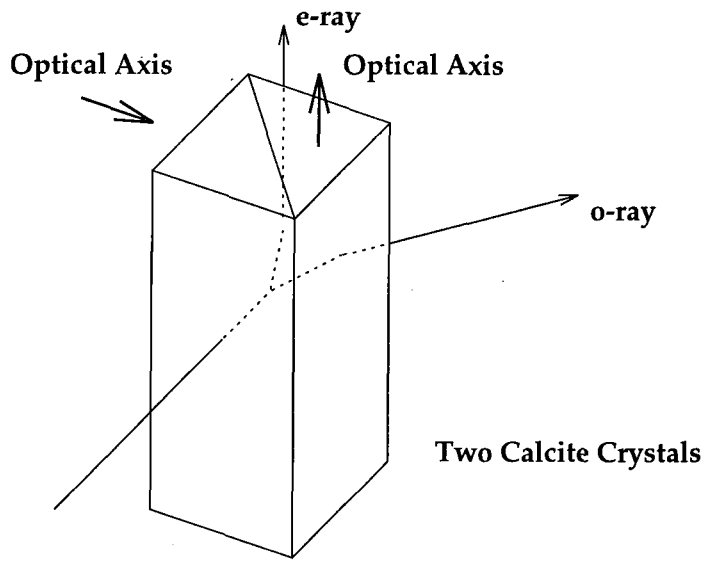


Figure 2-2 - Wollaston Prism

Each plate pair is achromatic in its own right and the two outer pairs have their optical axis parallel to each other, with the optical axis of the middle pair slightly offset. The half wave plate is mounted directly in front of the Wollaston prism and is rotated by 22.5° between exposures thereby rotating the plane of polarisation by 45°. The first exposure records position angles of 0° & 90° followed by the second exposure recording position angles of 45° & 135°. In practice, four measurements are made in all, the other two measuring 90° & 0° and 135° & 45° so that any systematic errors may be cancelled out when analysing the data.

To prevent double images on the detector from the two diverging beams of light, a system of parallel obscuring bars is placed at the focal plane of the telescope. This blocks part of the image from the telescope (Figure 2-4 - View through grid mask) but allows the two subsequent images to interleaf on the detector where one strip represents the light polarised at say 0° and the other strip representing light polarised at 90° (Figure 2-5 - View through prism & mask).

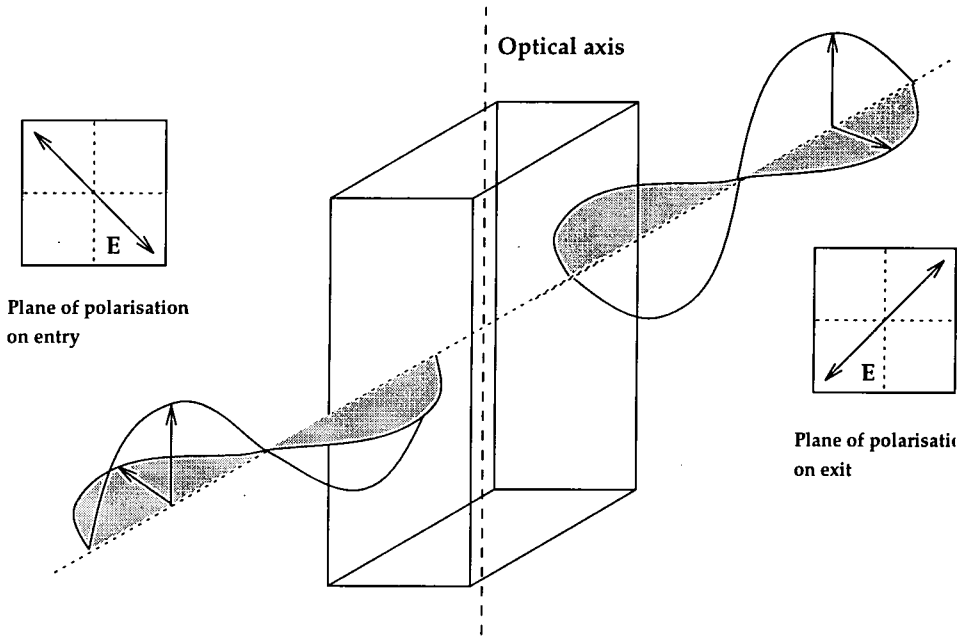


Figure 2-3 - Half Wave Plate

The grid mask, as it is known, is constructed from strips of glass fibre board, milled to an optimal size and sprayed matt black to prevent unwanted reflections. The size of each grid and the spacing between them is accurately set to match the optical configuration so as to prevent any overlap of images or loss of data, and then mounted into a carrier so as they are unable to move. The grids are made from a non-metallic material to avoid the possibility of polarisation of the incoming light by the metallic

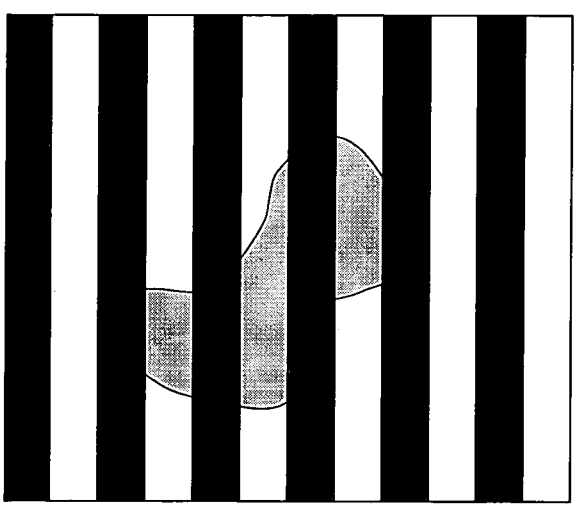


Figure 2-4 - View through grid mask

edges and are bevelled so as not to present a knife edge.

During the observing session it is sometimes necessary to replace the grid mask with a set of grid spots so that the optimal focus setting of the polarimeter may be acquired. The grid spot mask is a metal plate with a matrix of small holes approximately 1mm in diameter drilled through it, which can be placed in the light path of the polarimeter. The effect is to produce an image on the CCD detector of these holes, which if they are circular as opposed to elliptical, indicates that the polarimeter is in focus. A secondary use of the grid mask is to take images of the grid spot for use at the data analysis phase. Each row of light spots represents light that has passed through the prism, one row representing the parallel polarisation plane and the other the perpendicular plane. These images can then be used to accurately calculate the divergence of the prism and the left-right transformation coefficient which is needed during the analysis of the data.

The half wave plate and Wollaston prism arrangement are mounted between two lenses, one at the front of the polarimeter and one at the back, which constitute a focal plane reducer. The image is reduced by a factor of approximately 4 which significantly reduces the integration time, a useful feature when observing faint objects.

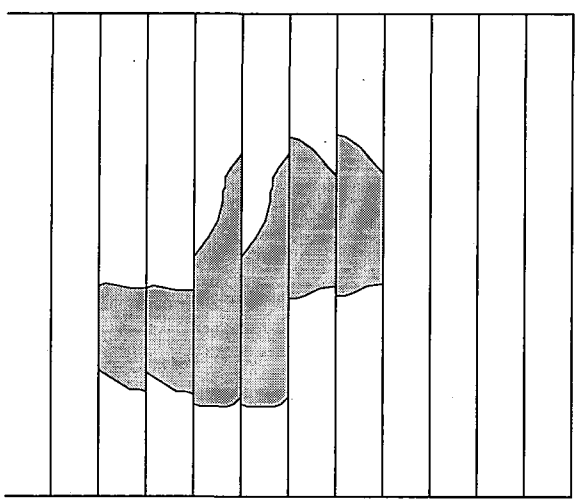


Figure 2-5 - View through prism & mask

As light enters the polarimeter it passes through a planar convex field lens with a focal length of 220mm, which reduces the beam width so that it can pass through the analyser stage unvignetted. The field lens lies close to the focal plane and just behind the grid mask and is mounted in such a way as it may easily be removed for cleaning. Lying so close to the focal plane of the telescope, any dirt that is present on this lens will be double imaged onto the detector and degrade the quality of the data. After the light has passed through the Wollaston prism it is re-imaged onto the detector using a relay lens. This lens is a Nikon 50mm f/1.2 camera lens with standard coated optics to prevent unwanted reflection.

The final optical component of the polarimeter is the filter wheel which is mounted between the Wollaston prism and the relay lens and allows one of eight filters to be positioned in the path of the beam. The filter wheel accepts circular filters of 51cm diameter in six positions and two square filters, 51cm square. The filter wheel is loaded with a standard set of filters whose bandpass parameters are given in

Table 2-1, while the square holes are normally used for narrow band interference filters which may be required for specific objects.

Table 2-1 - Filter Bandpass

Filter	λ_{max} (nm)	λ_{mean} (nm)	F.W.H.M.
B	440	440	96
V	525	544	86
R	595	663	157
I	808	883	297
Z	>1100	>1026	>>120

2.3 Mechanical Design

With the introduction of CCD detectors there was less of a need for the observer to be located in the actual dome and so the first automated polarimeter was designed and implemented by Dr S. M. Scarrott and Dr R. F. Warren-Smith. This Mark IV polarimeter used motors to rotate the half wave plate to specific angles, select filters, focus the instrument onto the detector and move the grid mask in and out of the light path. The polarimeter which will be described by the author is known as the Automated Polarimeter Mark II or more generally, the Mark IV polarimeter and is based upon the design of the previous one, but using smaller and faster motors and has a more sophisticated control system.

2.3.1 Material Specification

The polarimeter is constructed mainly out of HE30TF aluminium alloy which allows it to be rigid while not being too heavy. The main frame consists of two 18mm thick rectangular plates, which lie at the top and bottom of the polarimeter, connected together by four 76mm square bars positioned at each corner. The top plate is connected to the telescope mounting plate while internally the grid mask carrier and field lens are fixed to it. The telescope mounting plate is constructed from a sheet of alloy 18mm thick and is mounted off the polarimeter using eight spacer bars. The length of these spacer bars is determined by the particular telescope the instrument is to be mounted to, so that the focal plane of the telescope lies in the plane of the grid mask. The plate itself has multiple sets of mounting holes for the different telescopes and is easily removed to enable a new set of holes to be drilled should there be a need. A cylindrical tube fixed between the mounting plate and the polarimeter prevents stray light from entering the polarimeter. The bottom plate of the polarimeter holds the CCD mounting assembly and internally has the relay lens mechanism and filter assembly attached.

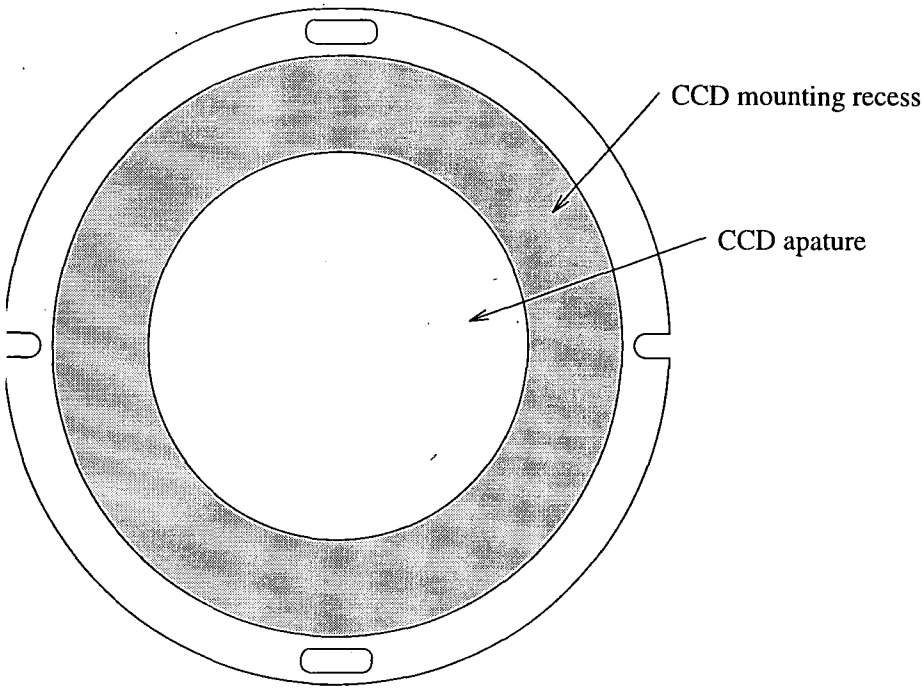


Figure 2-6 - CCD X-movement Plate

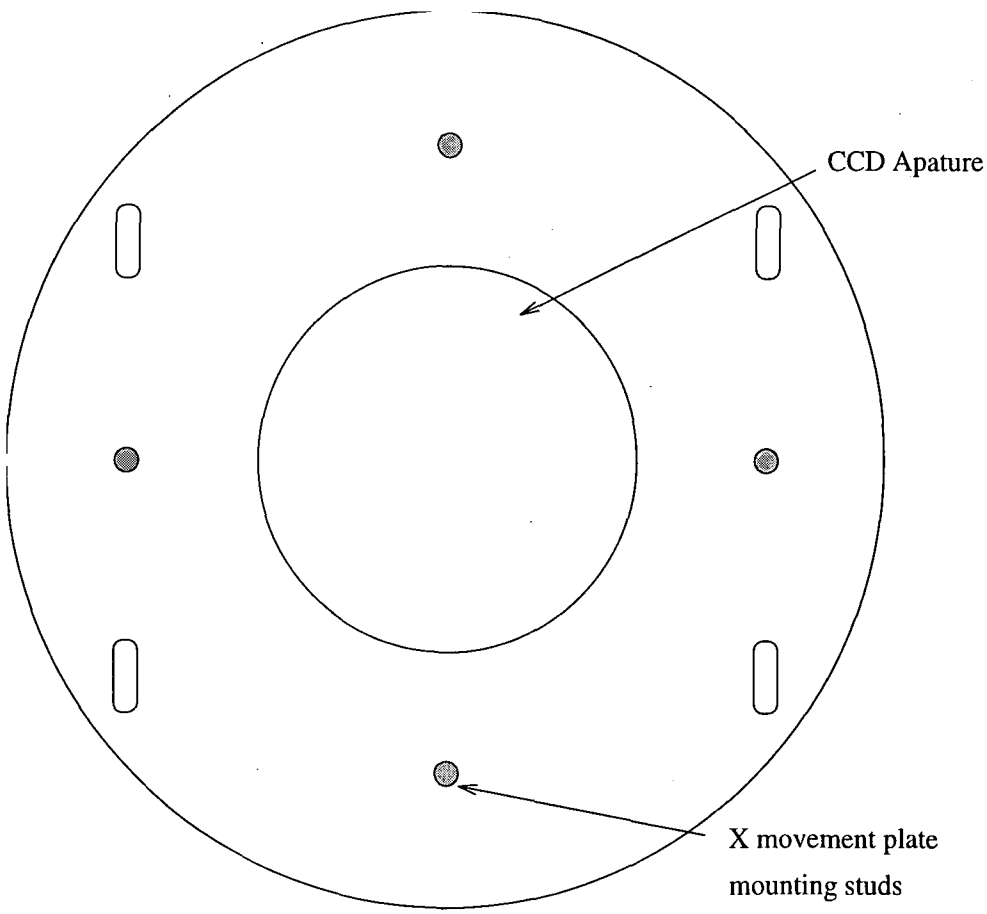


Figure 2-7 - CCD Y-movement Plate

The CCD is mounted onto the polarimeter by placing it into a circular recess slightly larger than its own base plate, then attaching with cap-head bolts, two semicircular collars to prevent it from moving. It is then possible to align the CCD dewar in the x and y planes by means of two plates with slotted mounting holes (Figure 2-6 - CCD X-movement Plate & Figure 2-7 - CCD Y-movement Plate). The dewar can also be rotated slightly before tightening the bolts so as to align the CCD chip with the grid mask.

Within the polarimeter there are four circular stainless steel bars centro-symetrically positioned running between the top and bottom plates, mounted to this via linear bearings is a plate carrying the Wollaston prism and half wave plate assembly. This plate can be locked into position using clamps but may easily be moved to allow for the cleaning of the filters in the filter wheel.

The polarimeter is made light tight by four side panels of which two may easily be removed to allow access to the optics. All the internal metal surfaces with the exception of the four circular rods are either anodised or painted matt black to cut down on internal reflections due to stray light.

The mechanical gearing assemblies which drive the various units within the polarimeter are commercially cut and bought in to ensure accuracy within known tolerances. The worm-wheels are made of phosphor-bronze and the worm drives from hardened steel, and are mounted to the polarimeter body using bearings to ensure that there is no movement in any unwanted direction.

2.3.2 Half Wave Plate

The half wave plate is mounted in a circular aluminium carrier surrounded by a bearing and gear wheel which meshes into the drive worm. This sub-assembly is then mounted to the aluminium plate held by the four circular rods passing from the front to the back plate of the polarimeter.

Photograph 2-1 - Half Wave Plate Assembly

The worm and wheel gear mechanism has a ratio of 60:1 so that for one complete revolution of the worm drive, the wheel will rotate by 1/60 of a revolution. Using the type of motors described in the next section, for each step of the motor, the HWP rotates by 0.03°. The aluminium sensor strip is mounted to the carrier and has sixteen 0.5mm wide slots cut into it and one hole to represent the home position. Both Photograph 2-1 - Half Wave Plate Assembly & Figure 2-8 show the half wave plate assembly of the Mark IV polarimeter.

2.3.3 Filter

The filter wheel is mounted via a central bearing onto an aluminium shaft which is attached to the bottom plate of the polarimeter offset from the optical axis. As the filter wheel rotates the centre of each filter socket comes into alignment with the optical axis.

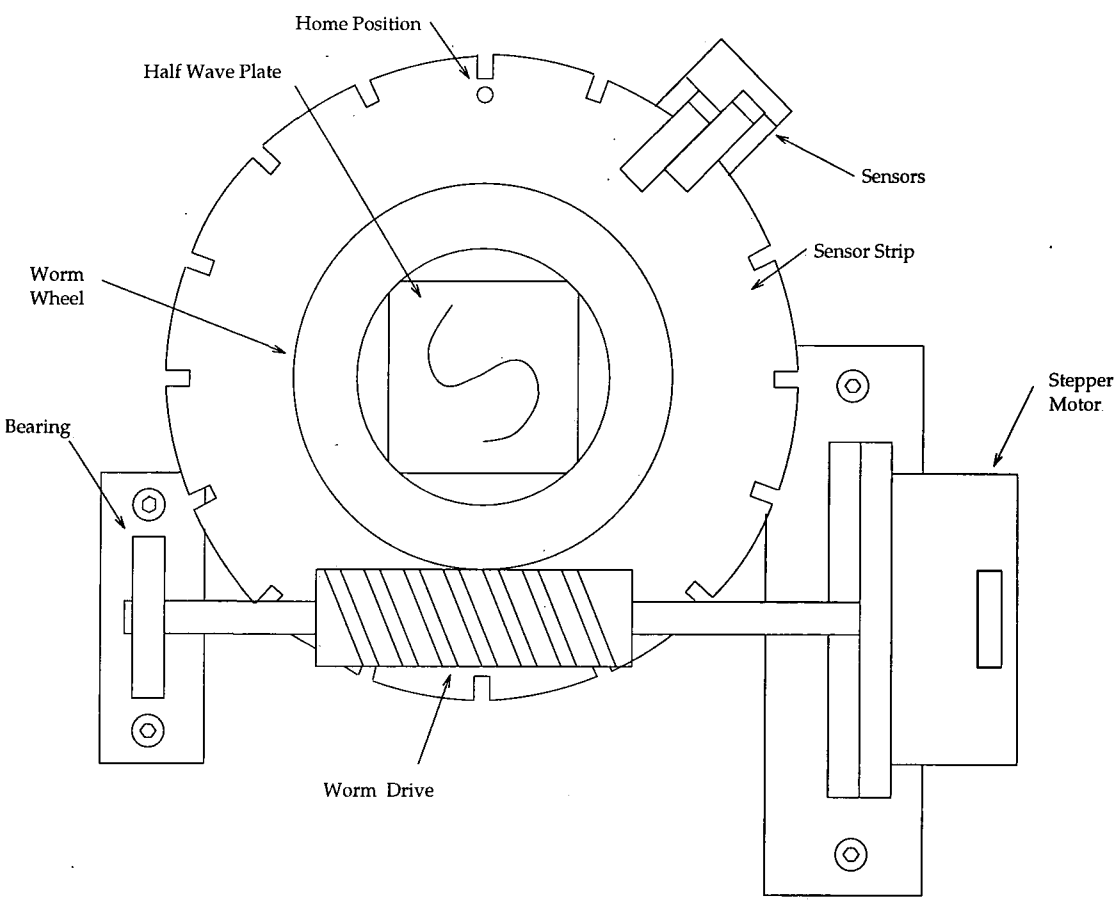


Figure 2-8 - Half Wave Plate Assembly

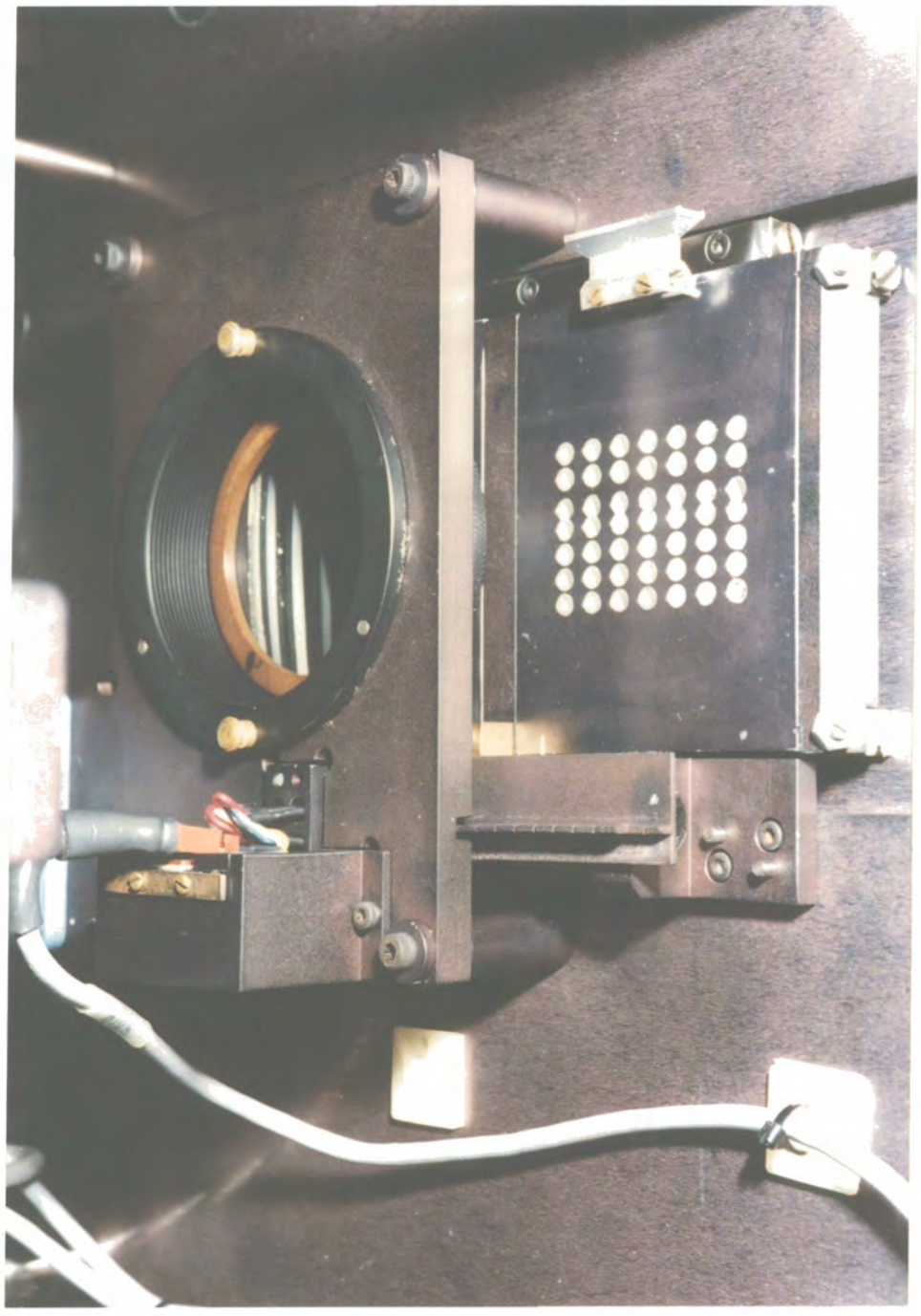
Around the filter wheel shaft is mounted the gearing which drives the filter wheel by meshing into the worm drive; the motor and worm driving mountings are also secured to the polarimeter bottom plate. The gearing ratio of the filter wheel drive assembly is 70:1 which translates a single step of the motor to 0.026° rotation of the filter wheel. The outer edge of the filter wheel acts as the sensor strip into which is cut eight slots and one deeper slot which acts as the home position.

2.3.4 Focus

The focus assembly is constructed around a standard 50mm Nikon camera lens and is semi-rotational i.e. it has limited movement and definite stops at each end of its travel. The lens is mounted into an aluminium collar which attaches to the polarimeter bottom plate via a bearing. The sensor strip and gear wheel are attached to the collar and to prevent the whole lens from rotating, the body of the lens is keyed to the bottom plate. The focus assembly is operated via a worm and wheel with a gear ratio of 60:1 and motor step angle of 1.8°. The sensor strip has nine slots cut into it with a hole to represent the home position. At either end of the strip there are extended cut outs so that both detectors operate at the same time to indicate the limit of travel (see section - Electrical Design)

2.3.5 Grid Mask

The grid mask assembly is mounted to the top plate of the polarimeter using two linear bearings and is driven using a screw thread drive shaft. The motor and drive shaft are mounted to the polarimeter top plate, the end of the drive shaft being held in position with a bearing. The drive shaft thread is cut at 40 threads per inch so that one revolution of the motor will move the grid mask 634.75μm and one step will translate to 3.17μm in linear movement. This enables the grid mask to be aligned to the same place on the CCD detector each time the dewar is remounted to the polarimeter, an important feature should there be any defects on the detector. Photograph 2-2 shows the field lens and the grid mask assembly lying behind field lens.



Photograph 2-2 - Grid Assembly and Field Lens

2.4 Electrical Design

Each of the four mechanically functional units (half wave plate, filter wheel, polarimeter focus and grid mask) are driven by stepper motors which are controlled by commercial stepper motor drivers and intelligent microprocessor based control cards. These cards are housed in a 19" rack and wired up following the author's design. The control system is then linked to a computer which communicates using a generic command language to move the units.

2.4.1 Motors

Stepper motors move by a fixed angle with great accuracy when energised by electrical pulses and are therefore ideal for this particular application where accuracy is required, and also have the ability to be driven at high speeds with reasonably high torque outputs. The motors chosen for this application are Slow-Syn M061-CE08 with a step angle of 1.8° which, with the associated gearing ratio, allows each of the functional units to be positioned with a much greater degree of accuracy than required. Each motor is driven by a dedicated bipolar stepper motor driver card which generates the sequence of electrical pulses to each phase of the motor to ensure accurate steps. The card also allows the motor to be placed into a holding state where the shaft is locked and cannot rotate, this useful feature is exploited for stopping the other units from drifting while a particular unit is being driven.

The motors are driven in a bipolar configuration by pulses of 1.25volts with a current rating of up to 3.8amps. The bipolar driving arrangement gives the motors greater torque so that they do not slip when starting up and also means that there is no need to employ power dissipating voltage dropper resistors. Because the motors dissipate significant amounts of heat when in use they are only switched into holding mode when necessary so as to prevent unwanted warm air thermals being generated within the polarimeter which would distort the light beam.

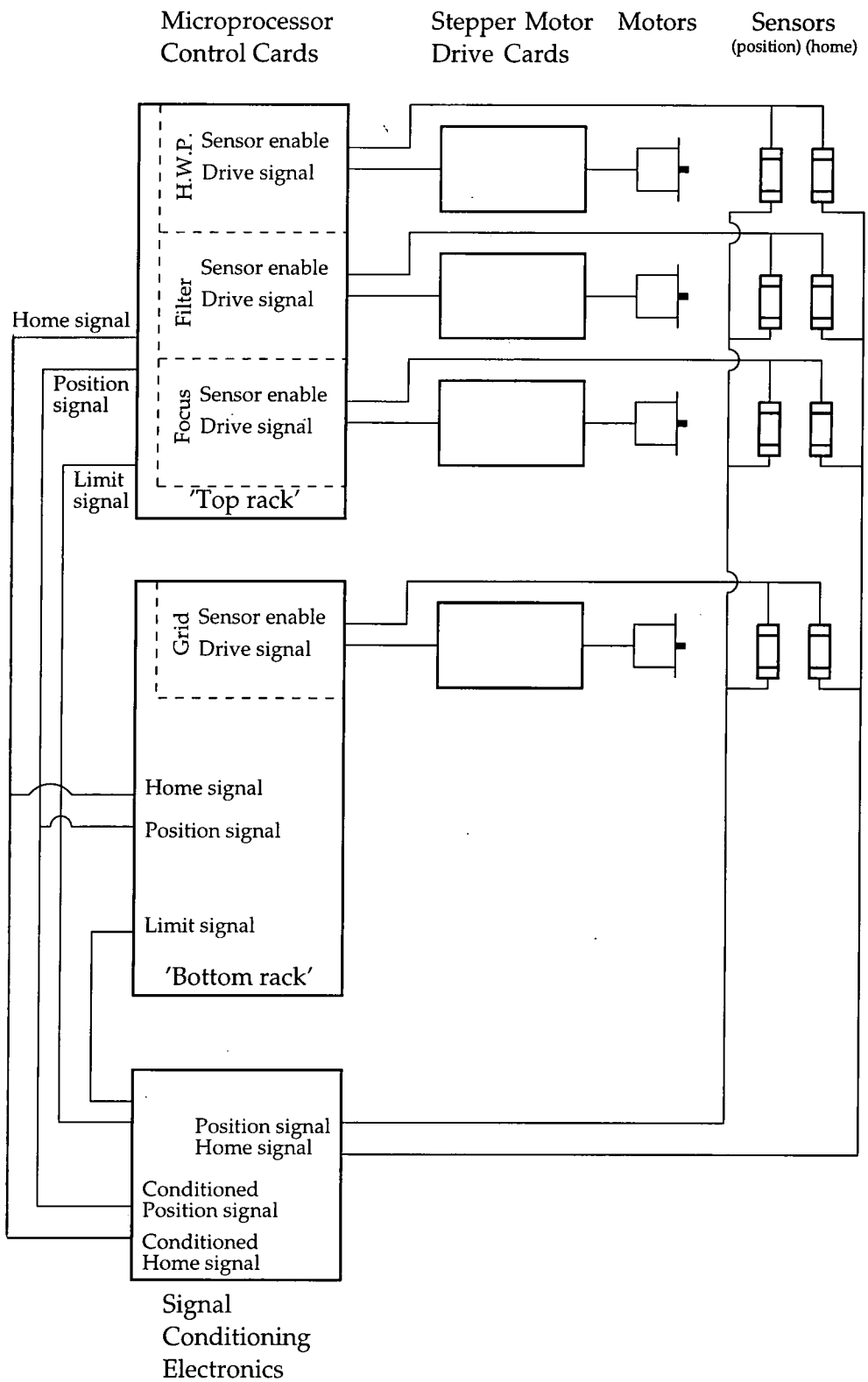


Figure 2-9 - Electrical Overview

2.4.2 Control System Wiring

Each motor is wired up to the connector panel on the side of the polarimeter with its own 4 wire lead which is interchangeable with any other motor lead should a problem occur. One of the main criteria to be met with this version of the polarimeter was interchangability so that any faults that occurred during the observing schedule could easily be worked around until a more suitable time for repair transpired. The connector panel is mounted with four military grade six pin sockets of which four pins are used for the motor and the remaining two pins are the power feed to the sensor assembly associated with that motor. The stepper motor driver cards are controlled using a dedicated microprocessor card which accepts simple commands via an RS232C serial interface and sends the appropriate pulses to the card to drive the motor. Each microprocessor card can control three motor cards and so there are two control cards as the polarimeter has four motors requiring control. The two microprocessor cards along with the four motor drive cards are mounted into a 3U high 19in eurorack along with the appropriate power supplies. In addition to commanding the motors, the microprocessor cards have a number of logical inputs and outputs which the user can exploit for control purposes. In the case of the polarimeter these inputs and outputs are used to power and read the sensors so as to determine if the functional units are in position, with the exception of one output which is used to enable the motors.

One of the main advantages of using stepper motors is the ability to move to a required position with a reasonable amount of reliability. With respect to the polarimeter, accuracy is important at a repeatable level but not necessary to an absolute level i.e. it is important to return the half wave plate to the exact position is was at, but the absolute position is arbitrary and only determined by the first time the polarimeter is used and calibrated. The system therefore does not rely on just the stepper motors to position the units, but uses sensors to check and correct the position. The sensors used in the polarimeter are standard slotted optoswitches (Figure 2-10 - Sensor Assembly) which contain an infrared light emitting diode (L.E.D.) and photodetector within the

same package so that when an object obscures the source from the detector, the voltage level changes and the system circuitry reacts accordingly.

Each of the four units have a double sensor assemble so that they can detect set positions as well as a home position for referencing all the other positions. The set positions (e.g. with respect to the filter wheel - filter 1, filter 2 etc.) are set by slots cut into a sensor strip attached to the moving assembly, that passes through the beam of the sensor so that when in a set position, the beam of the sensor falls on the detector but when the unit is not in a set position, the sensor strip obscures the beam. The two optosensors are mounted onto a purpose made bracket which slides into position so that the sensor strip is located between the diode and detector. The bracket assembly is held in place by a magnetic strip and wired up via a small plug and socket arrangement so that it may easily be removed and replaced should one of the sensors go faulty. The L.E.D.s of a particular sensor assembly are wired up in series and powered from the associated socket on the connector panel, while the outputs from the detectors are wired up to a military grade three pin plug on the same connector panel. Similar members of each sensor pair are all wired up in parallel so that all the home sensors are paralleled and all the position sensors are paralleled (see Schematic 2-1). This does not cause any conflict as only the L.E.D.s associated with the moving unit are activated when movement is initiated.

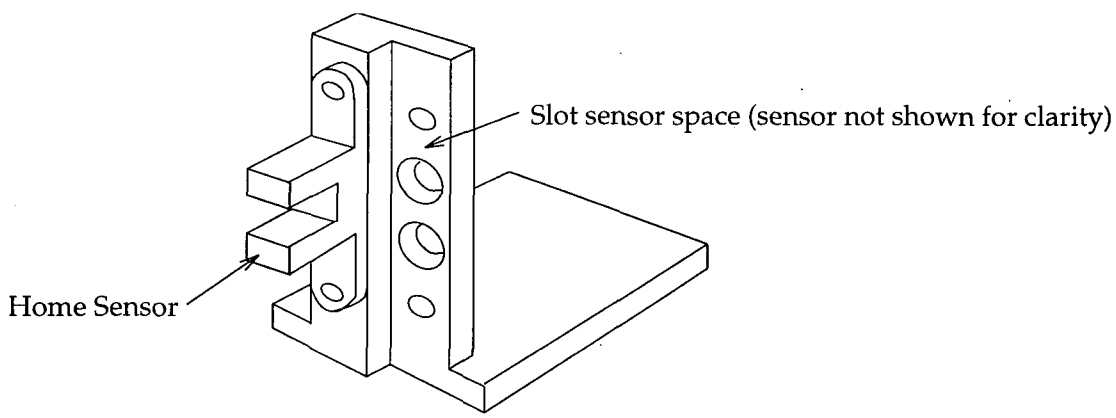


Figure 2-10 - Sensor Assembly

Two of the units (i.e. H.W.P. and filter wheel) are rotational in their movement and therefore will continue to go round continuously if the motor is left running, but the other two (i.e. focus and grid mask) have semi-rotational or linear movements which reach physical limits if left running. With these latter two units it is important to prevent them from reaching these limits or physical damage may be caused to the mechanics and so limit sensors are employed. Rather than use more sensors, the system looks for a particular code from the double-sensor assembly to determine if the unit is near the limit of its travel, the outputs are represented in a table of logic (Table 2-2 - Sensor Logic). The outputs from all the sensors are fed into an electronic circuit where they are conditioned and at the same time the output from the focus and grid mask sensors can be monitored to detect the limit condition. The circuit, on detecting such an event, will send a signal to the system which immediately stops the motor and prevents the unit been driven to its physical limit.

Table 2-2 - Sensor Logic

Status	Home Sensor	Slot Sensor
Between slots	0	0
Over slot	0	1
Over home	1	0
At limit	1	1

2.4.3 Circuit Description

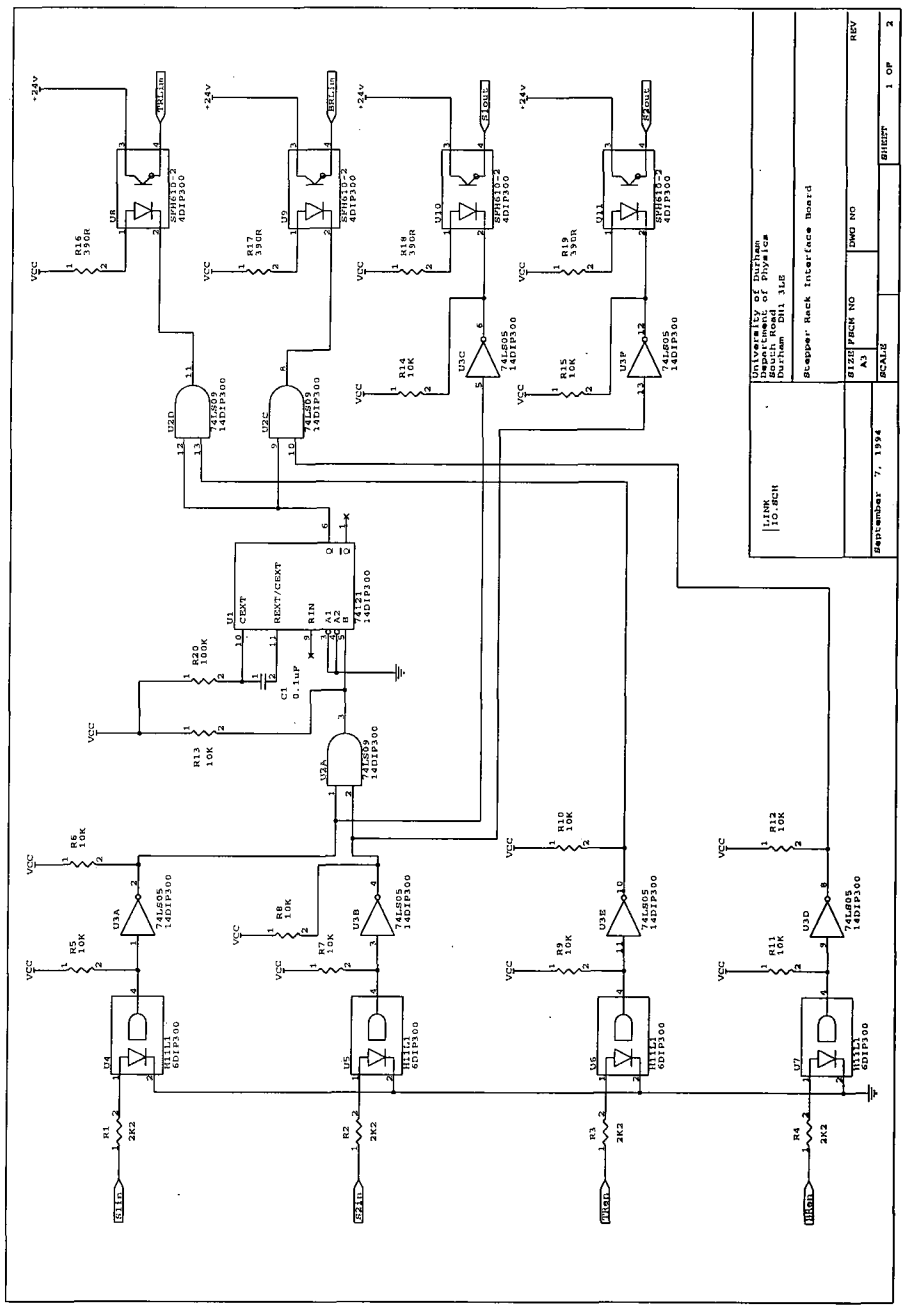
The conditioning circuit is constructed on a double sided plated through hole printed circuit board which was designed, laid out and assembled in-house by the author. The main circuitry is shown in Schematic 2-1. The integrated circuits U4 - U7 are optoisolators with TTL schmitt trigger outputs, these give the circuit an element of protection but are mainly to drop the input signal down to the TTL level of 5volts from 24volts, as the sensors and the stepper motor control cards all operate at this higher voltage. The resistors R1 - R4 are there only to limit the current to approximately

10mA through the internal light emitting diodes of the optoisolators. The integrated circuit U3 is an inverter which inverts the signals from the optoisolators, and the resistors are there as voltage pull-up's due to the nature of the output circuitry of the isolators and inverter. The signals are represented by tags on the far left hand side of Schematic 2-2 indicating there source, S1in is the input from the position sensor, S2in is the input from the home sensor, TRen is the top rack enable signal and BRen is the bottom rack enable. The conditioned and inverted sensor signals are then connected to an AND gate U2A which looks for the coincidence of the two as explained earlier, and are also re-inverted (U3C and U3F) and routed into two more optoisolators where they are reconditioned into 24V signals so that they may be connected to the stepper motor control panel. The output of the AND gate is fed into a retriggerable monostable which produces a fixed length pulse of 1ms indicating that the driving unit has gone into a limit condition. The pulse width is determined by the resistor/capacitor pair R20/C1 and the formula

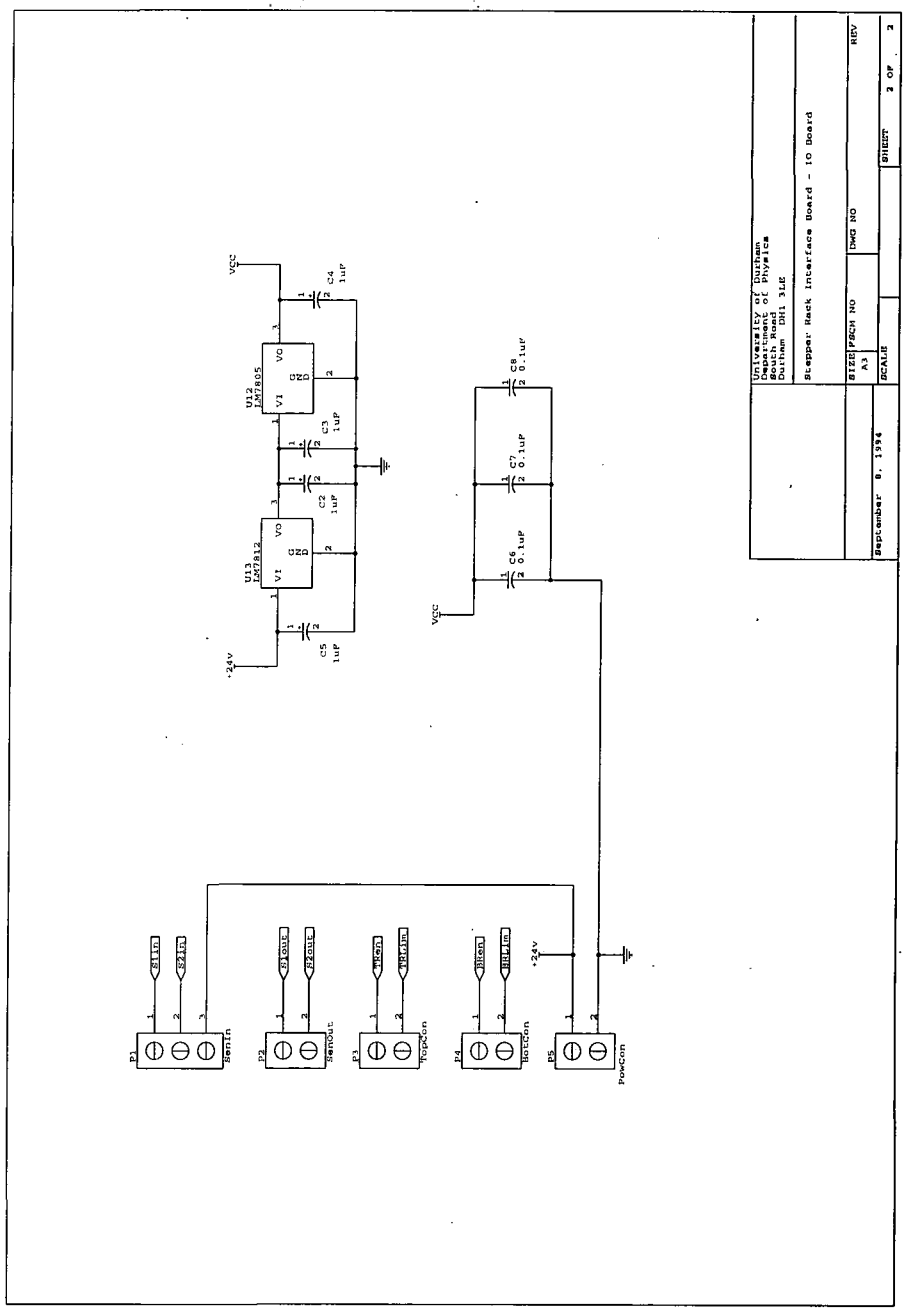
$$\mathbf{T = 0.1RC}$$

The output from the monostable is fed into two AND gates U2D and U2C which are also fed from the rack enable signals TRen and BRen respectfully. This ensures that only the rack which is driving the unit that is moved into a limit condition is signalled, as only the relevant enable line will be activated at run time. These two signals lines are then connected to optoisolators to be conditioned to the necessary 24V.

The circuit is powered from a 24volt power supply obtained from the stepper motor control card and is regulated down to 5volts using two regulators U12 and U13. This two stage regulation circuit is used as the head room (difference between the input voltage and output voltage) is too great for a single regulator which would produce too much dissipated power for the required current of the circuit.



Schematic 2-1



Schematic 2-2

Also shown on the same schematic (Schematic 2-2) are the decoupling capacitors which prevent high frequency spikes from affecting the TTL integrated circuits, these 0.1μF capacitors are connected close to the positive supply pin of the integrated circuit and to a good ground. The connectors P1 - P4 connect the P.C.B. to the stepper motor control board and sensors and are pluggable so that if the circuit board should fail for any reason, the connecting leads can be easily be removed and the board replaced with a spare.

2.5 System Set-up

The main control of both the polarimeter and the CCD camera is via a personal computer or computer workstation running the multi-tasking UNIX operating system. The UNIX operating system allows both control tasks to run concurrently as well as any other tasks which may need to be performed during the observing period, such as dumping the collected data onto a backup device e.g. a tape streamer. There have been a number of different set-ups tried and tested on the way to rationalising down to the most efficient system which is easy to use, offers the least weight and greatest productivity. Three such configurations are outlined in the next section.

2.5.1 Standalone P.C.

This set-up represents the base configuration and is the minimum needed to run the Durham polarimeter system. From this set-up all the other set-ups are derived. The personal computer (P.C.) is an A.T. based machine with an Intel 80286 microprocessor and is supplied by Elonex. The machine consists of the main computer with colour E.G.A. graphics, 100Mbyte internal winchester disk drive, 5.25" floppy disk drive and runs the Microport System V AT version of the UNIX operating system. In addition to this standard P.C. configuration there is also the CCD camera interface card, a high resolution graphics display card, a tape streamer interface card, an ethernet network

interface card and a multi-port communication card giving a total of four RS232C interface lines. The original setting up of the personal computer was carried out by Chris Rolph (1990) but has subsequently been changed by the author when the network and communication cards were added. These changes involved patching the kernel of the operating system so that the system could operate using multiple communication ports to the Polarimeter control hardware and network connections to allow remote access to the system.

The high resolution graphics card is used to display the images taken by the CCD camera and has a resolution of 800 by 600 pixels. This resolution was extremely good when the system was purchased but by present day standards is quite low, though is well matched to the EEV P86000 CCD chip in the camera. The images are displayed onto a 14" colour multi-sync monitor which is used solely for this purpose as there is a second 14" EGA monitor for use as the operator console.

Inside the P.C. A.T. there is an interface card to the CCD camera system. This card is effectively a digital parallel input/output card operating at TTL levels (0v/5v) and connects to a parallel-to-serial communications box which represents part of the CCD camera system. The communications box talks with the remote CCD controller down a 50Ω co-axial cable and uses Manchester encoding to ensure that the signals in either direction, whether commands or data, reach the destination in tact. The interface card, communications box and the CCD camera were all supplied by Astromed of Cambridge.

The multi-port RS232C communication card that is fitted to the P.C. A.T. replaces the standard two port card and provides a total of four serial interface ports. There is a requirement for three of these ports, two of which communicate with the polarimeter control hardware and the third to communicate with a graphics tablet used by the observer to select command procedures. The graphics tablet is an easy way for non-experienced observers to master the observing system without having to memorise lots of new commands and the syntax of there use. On top of the tablet is a printed table

with all the commonly used commands and all the observer has to do is move the puck over the relevant command and click the button to activate the particular function e.g. take an exposure, move the half wave plate etc. . When the button of the puck is pressed, the system executes a number of lower-level commands which are hidden from the user.

The storage device used with this system is a Cipher 1/2" tape streamer connected to the computer using the generic interface card mentioned previously. The tapes store the information at 1800bpi (bits per inch) allowing approximately 64 data frames to be stored on each tape, which compared to more uptodate devices is poor in terms of weight to storage ratio. The size, weight and storage capacity of the actual tape drive were the main reasons for moving away from this set-up to the alternatives mentioned in the next sections. The data frames are stored on the tape in the FITS format which has become an unofficial standard in the astronomical community and allows the data to be read by many different image display tools and reduction packages.

2.5.2 VAX Workstation

The VAX workstation configuration is based upon the P.C. setup and a Digital Equipment Corporation (DEC) VAXstation 3100 workstation. The VAX is a high powered multi-user system with 24Mbytes of memory, 100Mbyte and 500Mbyte winchester disks and high resolution 8bit colour graphics display. The operating system used on this workstation is VMS and its primary role is to reduce and analyse the data collected with the CCD camera and polarimeter system.

The idea behind using the workstation was to provide the observing team with the means by which to do a "quick" analysis at the observing site, to determine the potential of the chosen candidate. With this facility, if a potentially good candidate that had been selected for the observing program turned out to show little or no intrinsic polarisation, this would quickly be spotted with the consequence of no more time being spent observing the object.

As mentioned above, this configuration is based on the P.C. system but without the cipher tape storage unit. The P.C. and the workstation communicate using the ethernet network connections of the two machines, which allows multiply and differing connections to co-exist between the computers. In this way, the user can log into the workstation and then open a new window on the screen which will act as a connection to the P.C. UNIX system. At the same time, the P.C. may be making a separate connection back to the workstation to transfer the latest image, ready for analysis. All this is carried out using a networking protocol known as TCP/IP (transport connection protocol/internet protocol) which allows terminal sessions, file transfer sessions etc. to run concurrently. This is the basis upon which this system is designed, the P.C. acts as a slave device which is controlled remotely from the workstation and as CCD images are read into the P.C., they are transferred automatically to the workstation. The workstation then stores the image onto disk and automatically converts it to a format that it can be process by the analysis software. At the end of the night, all the downloaded images are transferred to a TK50 tape drive for save keeping and transport back home. The tape drive is connected to the VAX workstation via the SCSI port and is capable of storing 50Mbtyes of data on each tape, which translates to approximately 100 CCD images. Both the tapes and the tape drive are much more compact than the cipher 1/2" tape system.

2.5.3 Apple Laptop

The Apple laptop computer setup, like that of the VAX workstation, is based upon the P.C. setup described earlier in section 2.5.1. The primary functions of the laptop computer are data storage and terminal access to the P.C. based UNIX control system. The Apple laptop computer used in this setup is a Powerbook 170 with 8Mbytes of memory and a 120Mbyte internal winchester disk drive, the display is provided by a 9 inch monochrome L.C.D. screen with a 640 x 480 pixel resolution. Data storage is on magneto-optical disks using an external Panasonic disk drive connected to the laptop

on the local SCSI interface. Each disk can hold up to 120Mbytes of data which is equivalent to approximately 240 CCD frames.

The laptop and the P.C. system are connected together by a thin ethernet network which transfers the data from the P.C. to the optical disk, and allows the apple to connect to the P.C. as a terminal.

The advantages of this system are in the weight and portability, both the computer and optical disk drive can be carried in hand luggage by the observer, which means they are not out of use for long periods of time during transportation and observing. The other advantage with this setup is simplicity, connecting the system together is a lot more straight forward than the VAX workstation and there is no need for the observer to have complex system administration skills that are required for VAX VMS operating systems.

3. Polarimeter Control Software

3.1 Introduction

The polarimeter control system (PCS) software is a suite of programs written in the C programming language. The software is run on the control computer and issues commands to the relevant stepper motor controllers to operate each of the four functional units, the half wave plate, the filter wheel, the polarimeter focus and the grid mask. These commands are issued as strings of alphanumeric characters down an RS232C serial interface to the microprocessor based controllers which interpret them and generate the correct sequence of pulses to move the motors in the desired manner. The control software also interrogates the controllers to determine the position of the motors and the status of the position sensors described in the previous chapter.

The software was written originally on an Elonex PC AT computer based on the Intel 80286 microprocessor. The computer ran a copy of Microport's Unix "System V/AT" operating system and the PCS software was compiled using the standard non-ANSI C compiler supplied with the operating system. The software used no special libraries other than those supplied with the compiler. The compiler produced a number of standalone executable programs that could be called from within shell scripts or at the command line interface of the operating system.

This chapter will deal with the command language of the stepper motor controllers, the principle of operation of the PCS software as well as a 'walk through' of the major

software functions, as well as a description of the utility programs required for 'day-to-day' operation of the system.

3.2 Stepper Motor Command Language

The stepper motor command language is proprietary to the Digiplan stepper motor controllers. The language consists of simple single letter commands followed by the required arguments and terminated with a carriage return. This allows the user of the command language to construct command strings within a high level program that can be transmitted down an RS232C serial port to the controller to make the motors move in a desired manner. These single letter commands may be concatenated to form more complex command strings which embed a complete function in terms of motor movement.

The card is controlled via its RS232C port which must be initialised after power-up. This is achieved by transmitting a single parenthesis character "(" and from this the controller calculates the baud-rate of the controlling computer. The computer then transmits a three character string which informs the controller of the number of data bits, stop bits and parity of the serial interface as well as whether the controller will 'echo-back' each received character. Once this is completed, the controller is initialised and ready to receive commands to control the motors and access the input and output ports.

The motors can be operated in two different modes depending on the commands issued from the host computer. The first mode is absolute where the command is followed by an argument representing the number of steps the motor should move, this type of command is useful for moving a functional unit to a desired position when the number of steps of required movement is calculable e.g. moving the half wave plate to position

14. The command is constructed first with the letter representing the motor[1], then the argument representing the number of steps and finally the execute command e.g. move the z motor 1000 steps in the negative direction ;

Z -1000 $

The second type of motor command is the run-mode command, this tells the motor to move in a certain direction until a command to stop is issued. This command is useful when commands are concatenated together to form more comprehensive command strings as will be presented further on in this section. The run-mode command takes the form of motor letter, followed by the optional direction argument, followed by the go command and then the execute command e.g. to start the x motor moving in the positive direction ;

X G $

In the case of both of these commands it is possible to add an argument within the command to give control over the motors rotational speed, e.g. to make the motor move at 2000 steps per second in the previous two examples the commands would be ;

Z - 1000 @ 2000 $ and **X @ 2000 G $**

The other main function of stepper motor controller card is to read the input ports and write to the output ports. These ports have various functions as discussed in the previous chapter, including enabling the stepper motors and reading the position sensors. Again, like the motor commands, the commands to write to the output ports are simple single letters commands with a single argument followed by the execute command. The command to take an output port to a logical high is "S" for set and to take it to a logical low it is "R" for reset, e.g. to switch output four high (enabling the stepper motors in the case of the polarimeter control system) the command string would be ;

---

[1] The relation between motor letters and functional units will be discussed further on in the chapter

S 4 $

The input ports may be interpreted in two ways, the first by requesting a status value which will return the status of all three ports, and the second by embedding the desired input port status into other commands such as run-mode motor commands. This is a useful way of building higher level commands which perform an operation on a functional unit within the polarimeter e.g. searching for the home position. To perform such an operation the motor needs to move at a slow speed until the two position sensors are in a desired state indicating that they are over the home position. The motor function then needs to abort the sequence. The command string for this type of operation would look like ;

: Y - @ 100 G : L1H2 : A : $

In this example the Y motor moves in the negative direction at 100 steps per second until input one is low and input two is high at which time it aborts. This is also a useful example of how the simple command set can be concatenated to create more complex commands.

One aspect of motor control that is important is acceleration and deceleration which is dependent upon what the motors are driving. This makes the motor ramp up to its maximum speed at a rate that will avoid motor stall thereby causing loss of steps and accuracy. As the motor approaches the required position, the controller ramps down the speed of the motor until it is at rest having completed the correct number of steps. In the case of the example above where the motor moves until a certain condition is met, the controller will start the deceleration upon the condition being true. The stepper motor control board allows this value to be set either within the motor commands or independently making it the default. In the case of the polarimeter, all the functional units represent similar mechanical loads and therefore there is never a need to change the default setting.

The stepper motor controller card also allows various status to be read as briefly mentioned earlier. Probably the most useful of these is the motion status which gives information about the motors and what they are doing at any one moment in time. The command to interrogate the controller is the single character "K" which causes the controller to respond with a status number between 0 and 255.

Table 3-1

Bit	Value	Condition
0	1	In motion
1	2	At Programmed Speed
2	4	Accelerating
3	8	Decelerating
4	16	Drive Fault
5	32	Emergency Stop
6	64	Limit
7	128	Communications Fault

If this number is interpreted as an eight bit binary number then each bit represents a condition and the return value can be interpreted as the sum of the individual condition values e.g. if the controller returns code 5 in response to the "K" command, then one of the motors is "In Motion" and "Accelerating". See Table 3-1. There are various other status functions available to pass back information about communication faults, input status (as discussed above) and rack status. One other useful status report is the position status that returns the number of steps the motor has moved, either due to the last completed motor command or if the motor is still in motion, the present motor command. This command is issued by transmitting the single character "N" which causes the controller to respond with a number between 1 and 4×10^9 steps.

These are the main commands that are used by the polarimeter control system software to control the four functional units and position them as the user requires. A full list of the command set can be found in Appendix A.

3.3 Principles of Operation

The polarimeter control system software, as mentioned previously, consists of a suite of individual programs to control each of the four functional units. The two programs that control the functional units that have full rotational movement (half wave plate and filter wheel) are logically similar, as are the two programs that control the movement of the semi-rotational and linear units (polarimeter focus and grid mask). All four programs are based upon the same principles of operation and only differ in values such as steps between slots and direction of travel. A schematic representation of the main program in the form of a flow diagram is presented in Figure 3-1 & Figure 3-2. A key to all the flow diagrams can be found in Appendix B.

Each of the four programs uses a common configuration file that holds the information about which stepper motor control card is connected to each of the functional units, and at what speed they should be driven. When any one of the four programs is executed, the first operation it performs is to look for the configuration file in the default directory, should it not exist the program exits as it is impossible to continue any further without this information. A flow diagram of the configuration routine can be found in Figure 3-3. If the file is found, the program opens and reads it until it comes across the relevant record for its particular function. An example of the configuration file is shown;

```
HWP A 1000 0100
FLT B 3000 0100
FOC C 2000 0100
GRD D 4000 0500
```

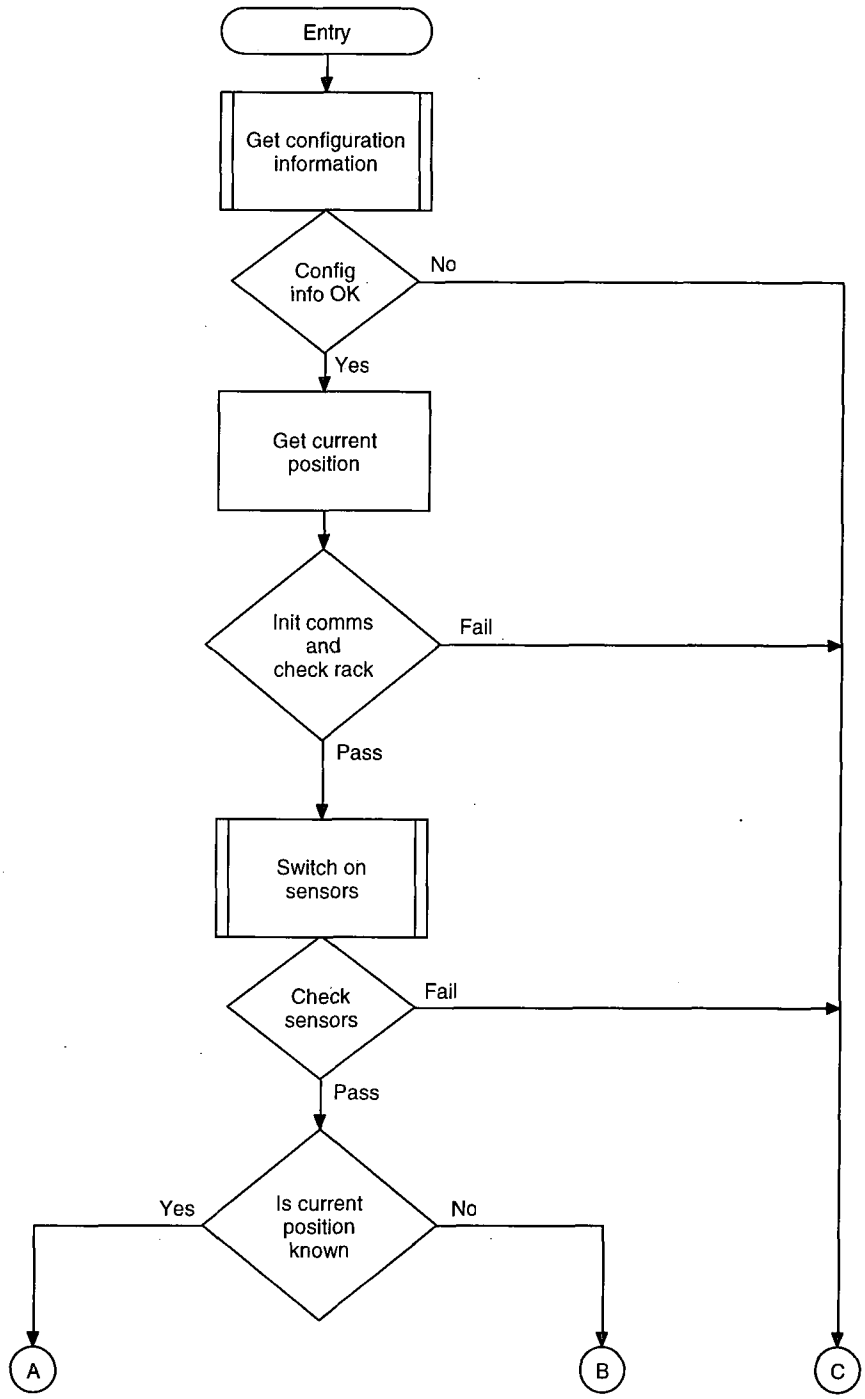


Figure 3-1 - Main Program Flow Diagram(Part A)

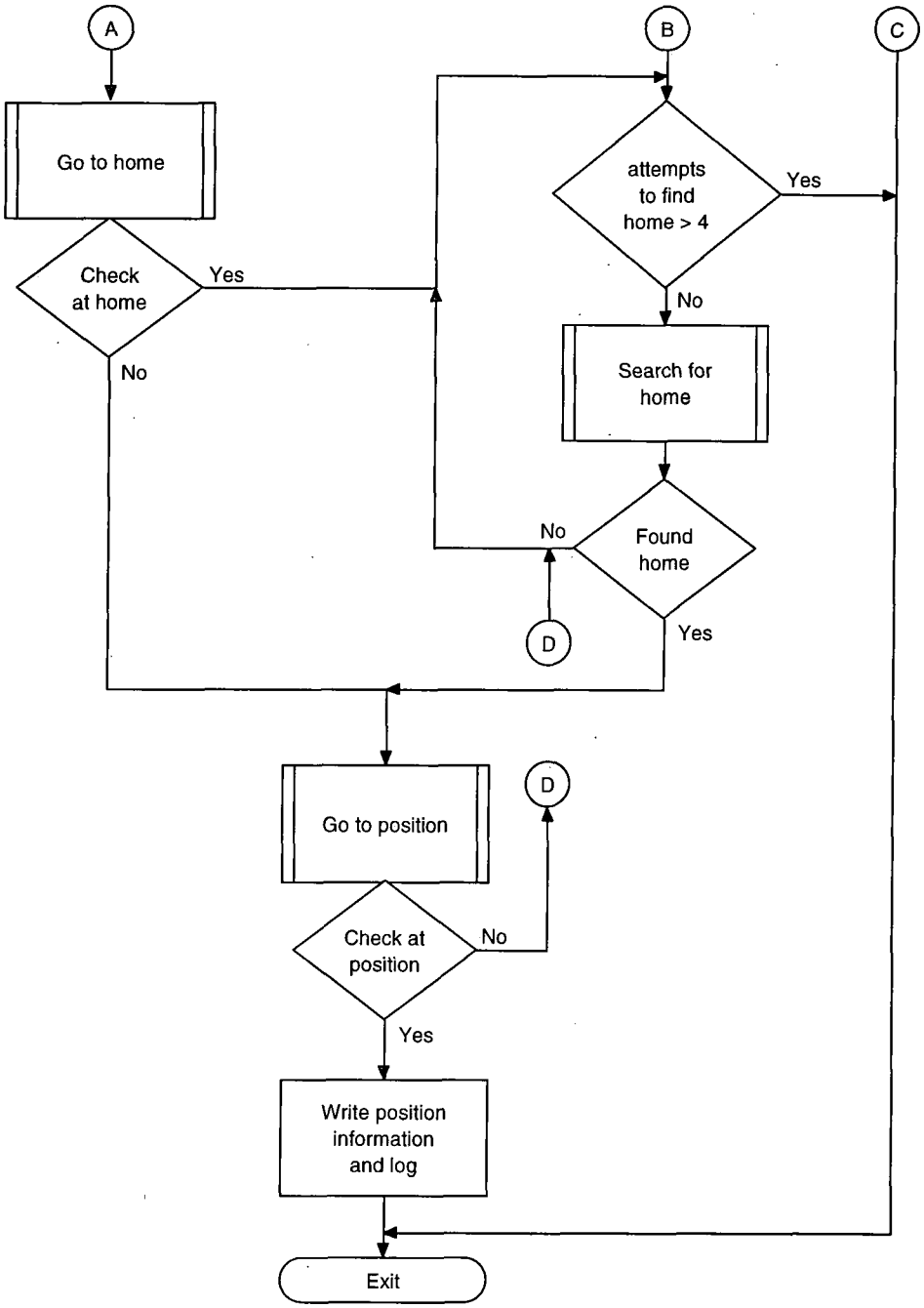


Figure 3-2 - Main Program Flow Diagram(Part B)

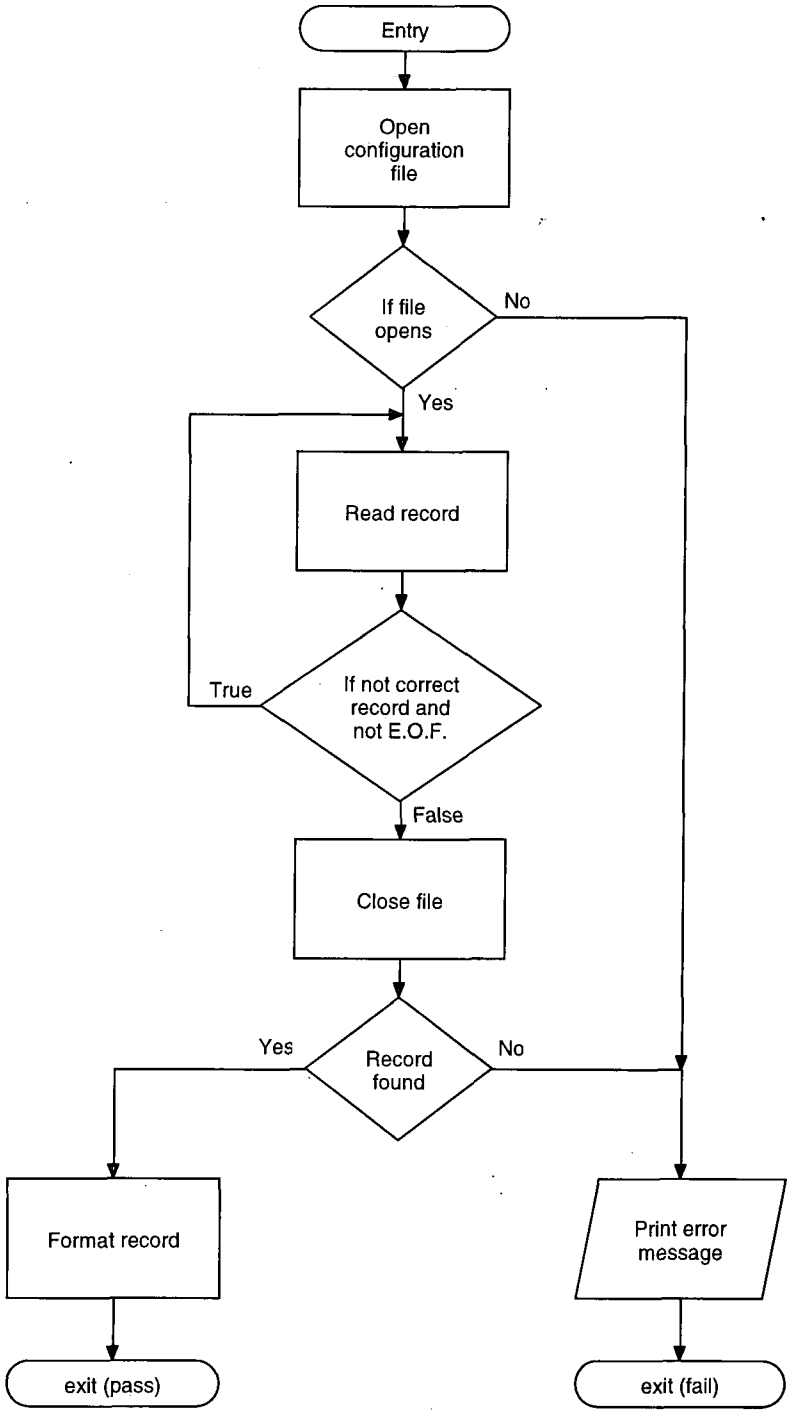


Figure 3-3 - Configuration Flow Diagram

The configuration file consists of four records, each with four fields. The first field is the function name, the second field the logical unit to which the functional unit is connected and the third and fourth fields are the fast and slow speeds of travel. The logical units correspond to the physical rack units as shown in Table 3-2.

Table 3-2

Logical Unit	Rack	Drive
A	top	Z
B	top	Y
C	top	X
D	bottom	X

The reason behind using logical names to describe which logical unit connects to which functional unit is to allow interchangability as stated in the previous chapter. If one of the stepper motor driver cards were to fail, the observer would only need to swap the cable of the affected unit to a different driver output socket and then alter the configuration file to reflect the change and the system would be in use once again. This level of flexibility gives the best guarantee against failure should a fault occur during an observing run.

After the program has read the configuration information and authenticated the data within the file, it then initialises the relevant communications port. A flow diagram of the rack initialisation routine can be found in Figure 3-4 - Initialise Rack Flow Diagram. Because there are two microprocessor control boards (referred to in Table 3-2 as top rack and bottom rack) the system uses two communication ports. The program initialises the relevant port for the functional unit depending upon the logical unit information in the configuration file e.g. if the functional unit is connected to logical unit A, then it will initialise the communications port connected to the top rack.

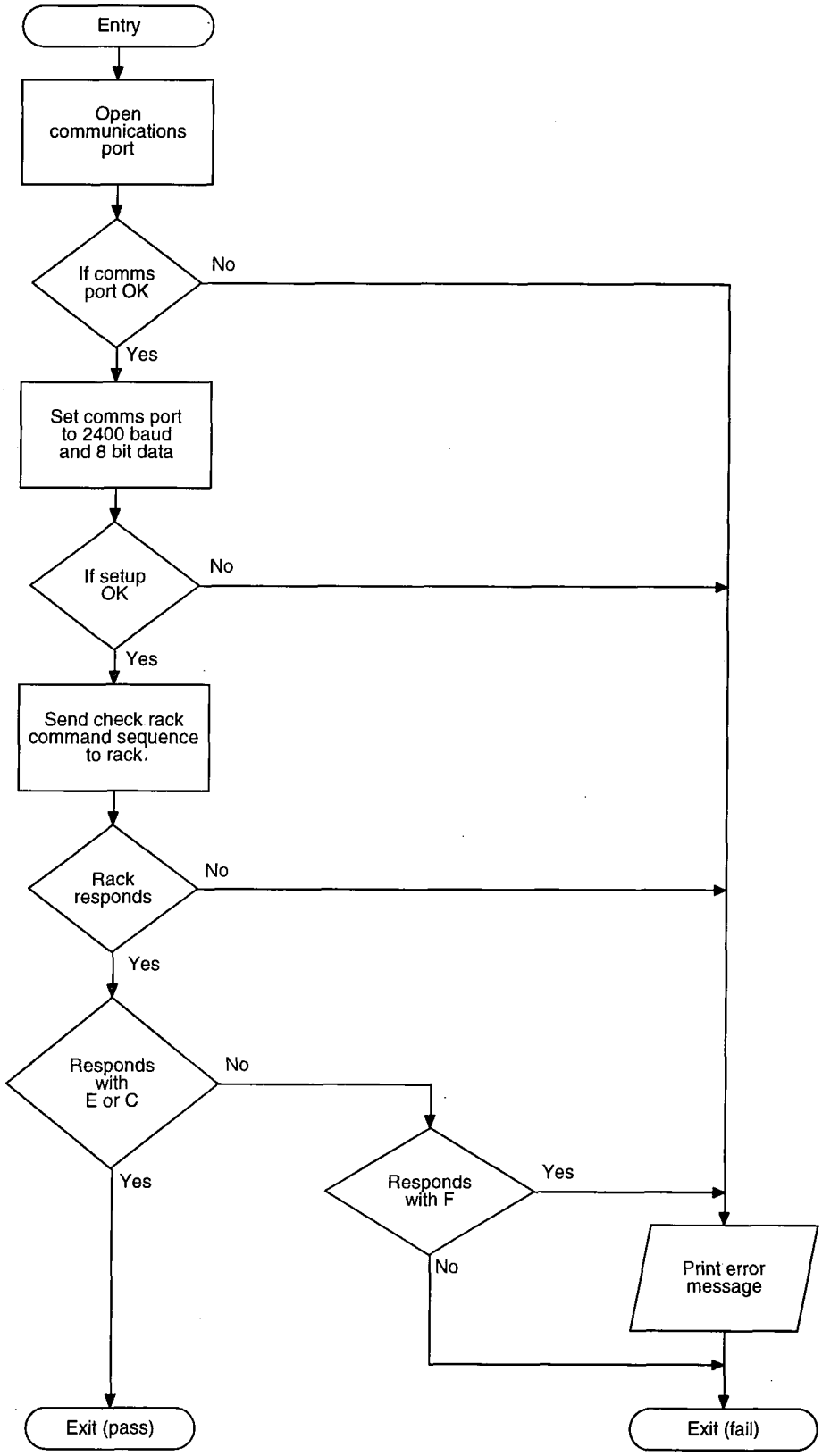


Figure 3-4 - Initialise Rack Flow Diagram

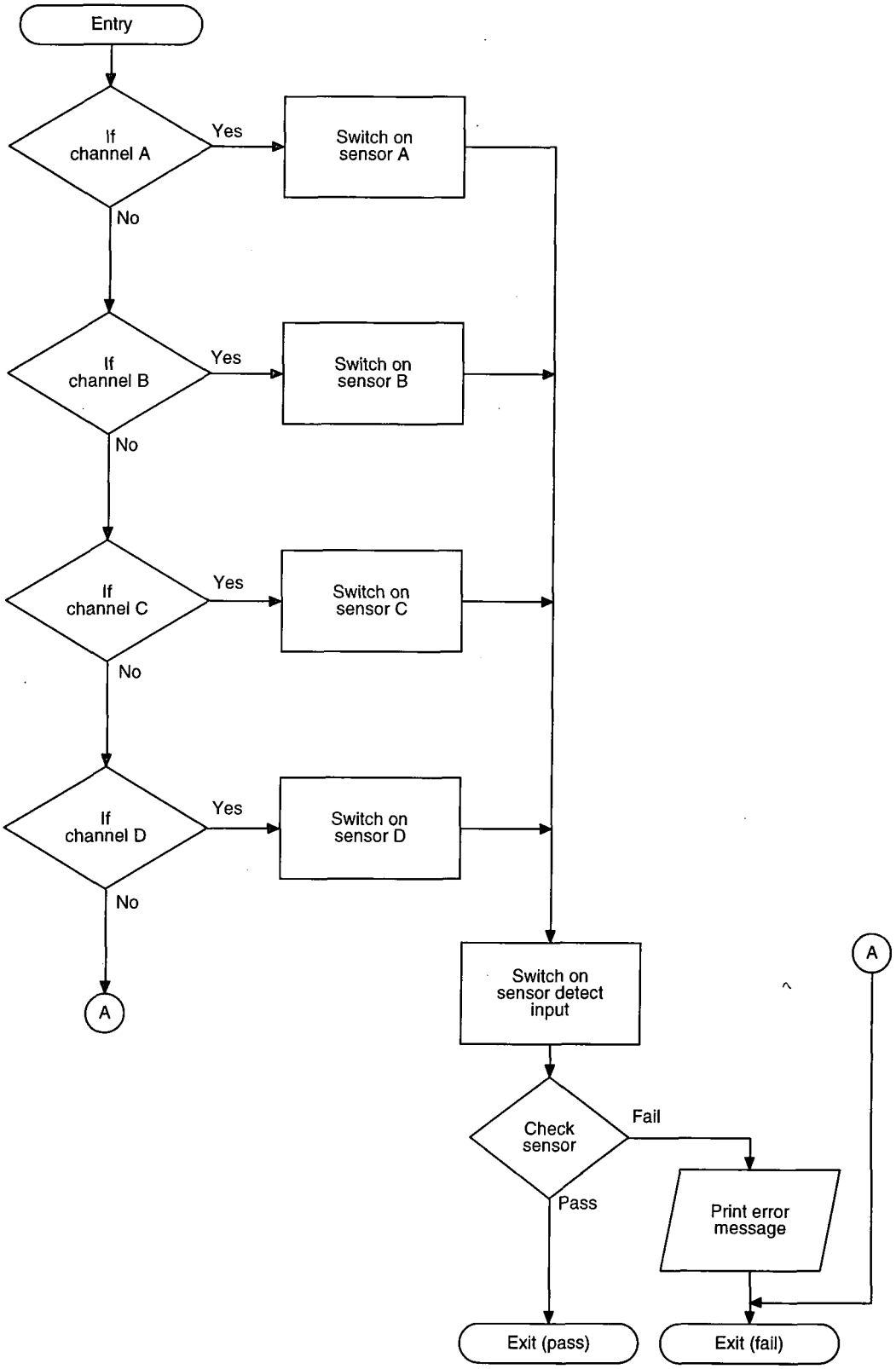


Figure 3-5 - Initialise Sensor Flow Diagram

If the communications port configures without any problems, the program proceeds to check that the microprocessor control board it initialised and ready to accept commands. At this stage of the program the relevant sensor assembly is energised. The flow diagram in Figure 3-5 describes this section of the code. Again, like the communication port, the particular sensor assembly energised depends upon the information read in from the configuration file. The program then checks to see that the two light emitting diodes of the sensor assembly are consuming current (and therefore functioning), by reading the relevant input to the microprocessor control board to which they are attached (see previous chapter). The program has no way of checking the detector parts of the sensors but this is still a useful test as it functions as a check to make sure the cables are connected correctly. If the sensor assembly check passes, the program then initialises the motors into there holding state where they will hold each of the functional units still while the relevant one is moved.

If at any time one of the above checks fails, the program will inform the observer of what has failed via a message on the console and then abort the program and exit, returning an exit-code to the calling program or operating system. Assuming all the checks have completed correctly the program continues by reading the current position of the functional unit from a file in the default directory. There is a position file for each of the four functional units and each file contains a character string representing the current position of the unit in the form;

5.78

The information is read into the program and converted into two numbers, one representing the major position value and the other number representing the offset. If the file does not exist, or if the data is corrupt in some way (e.g. the major position value is outside the valid range for the functional unit), then the program will inform the observer of this and then proceed to search for the home position so that it can realign the unit. The flow diagram in Figure 3-6 describes the logic applied when searching for the home position. The program transmits the command sequence to

start the motor moving in the direction of home at the slow speed specified in the configuration file. The motor continues to move until the home sensor reads true or, in the case of the polarimeter focus and the grid mask, a limit is reached (this only occurs if the unit was positioned at a more negative location than the home position).

Once the home position is found, the program then initiates a sequence of commands that align the sensor in the middle of the home position so that the program knows the alignment of the unit to within a single step the motor. The unit is then ready to be moved to the new position specified on the command line or by the reply given to a prompt at the start of the program by the observer.

If the position file was valid, then there is no need for the program to initiate the search for home. In this case, the program uses the current position value to go to the home position and check its positional integrity. Figure 3-7 describes the logic used by the control computer in moving the functional unit to home. The program first calculates the number of steps required to move to slot zero and issues the command to move the motor the required number of steps. The program then checks the position sensor to make sure it is over a slot and if this is true issues the motor command to move the unit to the home position where it then checks the home sensor. If this sequence is carried out correctly then the program assumes the unit was at the correct position and carries on to move to the new position. If for any reason the system fails to find a position slot or home, it then initiates the search for the home position as described earlier. By checking the home position at the beginning of each move the program accomplishes two useful tasks. The first is the positional integrity is checked for each new positioning of the unit thereby informing the observer of any problems as soon as they may arise, and second, the unit always moves into position from the same direction of travel thereby eliminating any backlash there may be in the gearing of the functional units.

Assuming that the program has found home either by searching or from a previously known location, it then proceeds to move the unit to the new specified position. As mentioned earlier, this new position can either be supplied on the command line, or if it is not supplied on the command line, the program prompts the observer for the new position. The flow diagrams in Figure 3-8 and Figure 3-9 describes the logic used in moving the functional unit to the required position. The program first calculates the number of steps the motor is required to step to move from its current home position to the slot specified by the major position value of the new location. The relevant command is then issued by the program to start the motor stepping.

Once the motor has stopped and the unit has reached its desired position, the number of steps executed is checked to see that it corresponds to the number of steps required. If the result of the check is true, the program continues but if it is false, the program returns to searching for the units home position as described earlier. Assuming that the number of steps executed is correct, the program then checks the position sensor to ensure that it is over a slot. If the sensor is not over a slot, again it will initiate a search for home and start over again. If for any reason the program does fail at any of these stages, it will attempt to find home and reposition itself a maximum of three times before aborting and exiting, informing the observer via a console message and returning an appropriate exit code to the operating system. If the sensor is over a slot, the program can continue by issuing the appropriate command string to move the unit by the desired offset.

At this point the program has completed its main task and all there remains to do is return the system to a stable state. First the motors are de-energised so that they do not warm up and create heat as mentioned in the previous chapter. Next , the new position of the functional unit is written into the appropriate position file so that it can be recalled when the unit is moved again. The program then writes a log entry into the log held on the disk so that each movement of all the functional units can be checked at a later date.

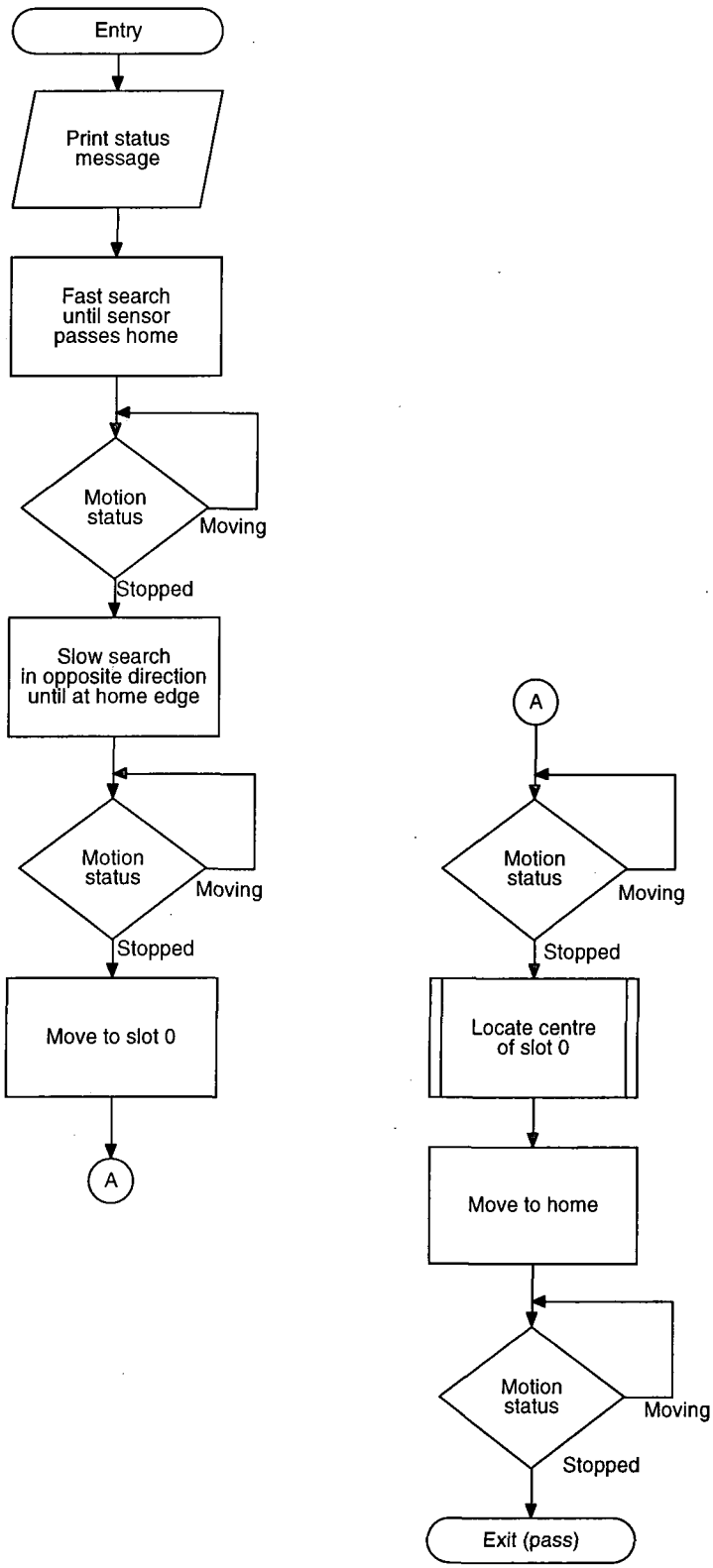


Figure 3-6 - Search for Home Flow Diagram

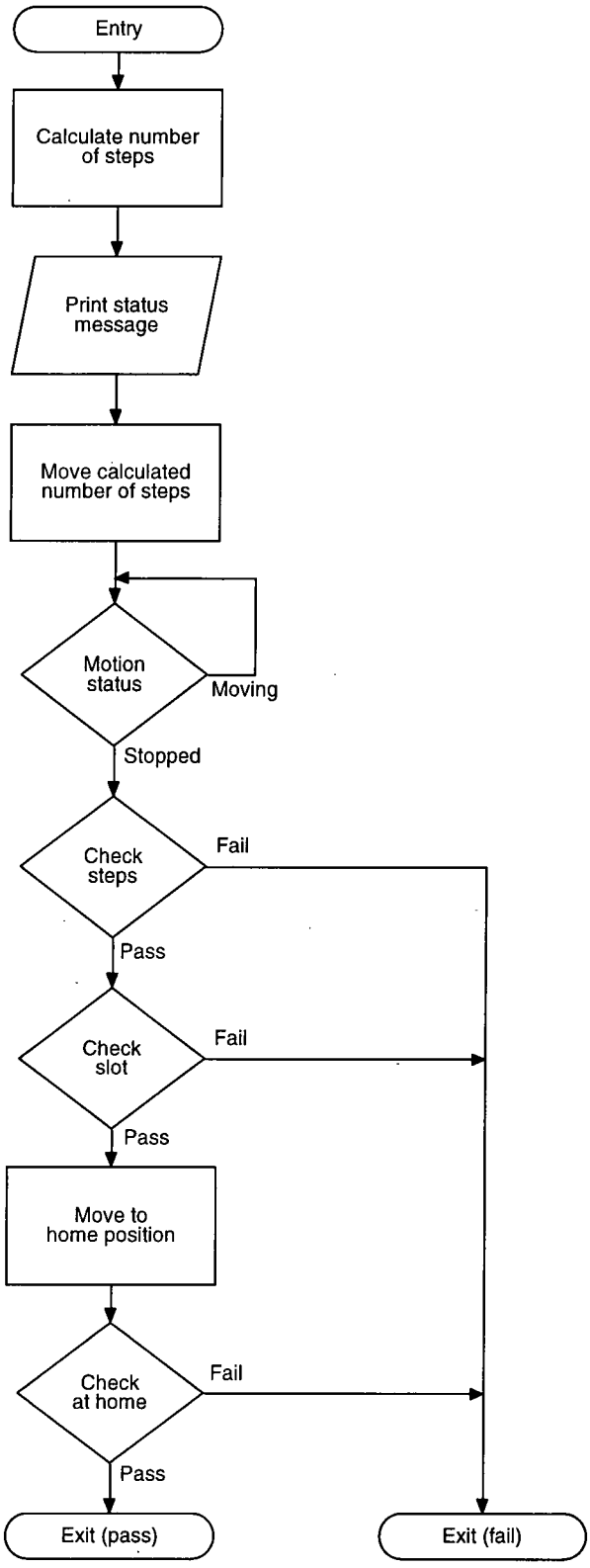


Figure 3-7 - Go to Home Flow Diagram

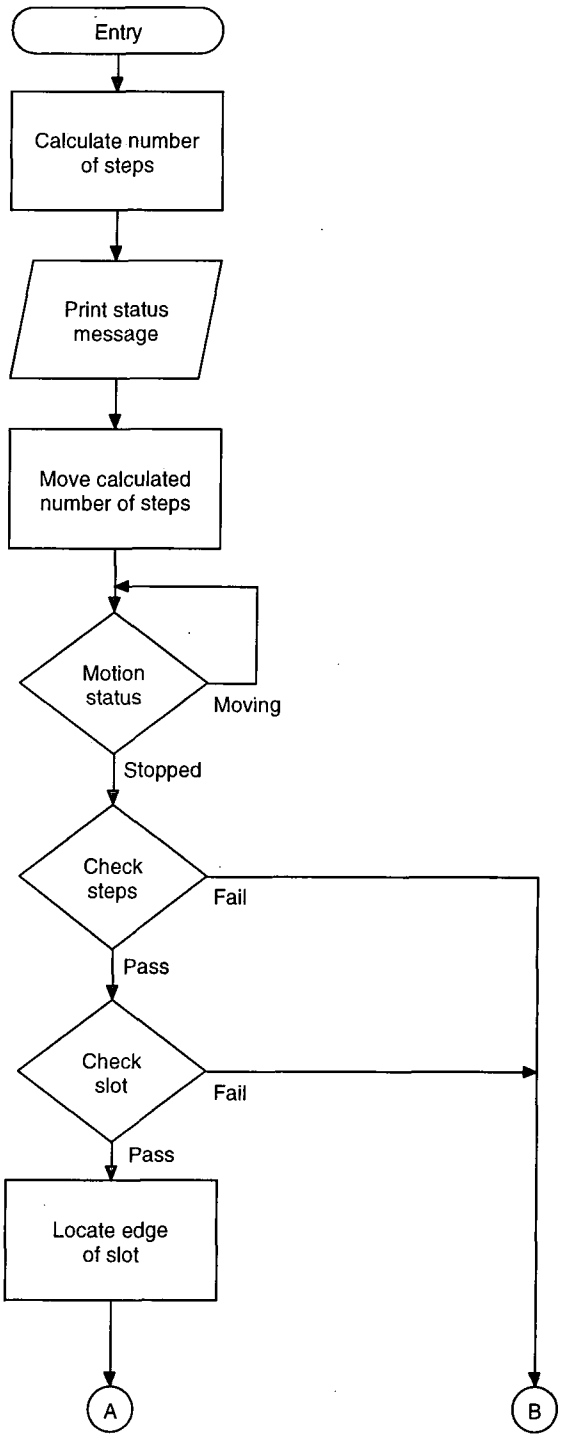


Figure 3-8 - Go to Position Flow Diagram (Part A)

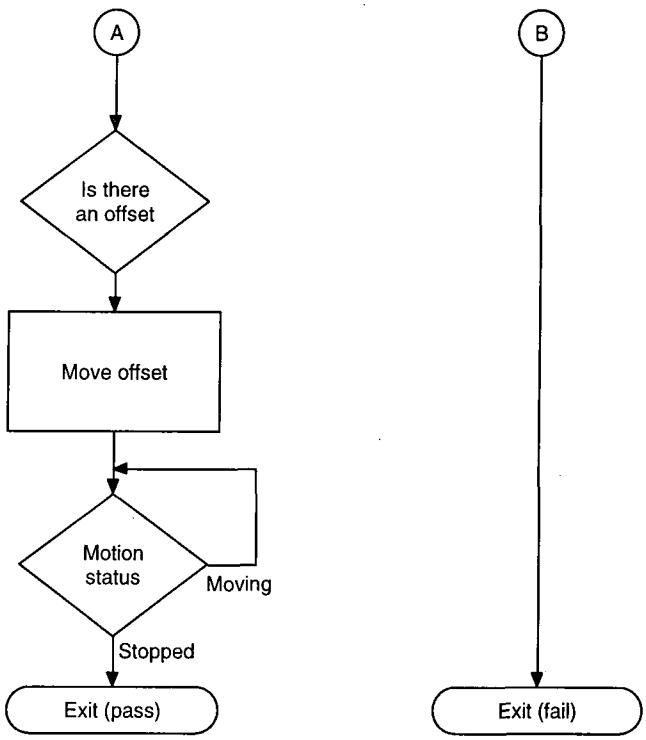


Figure 3-9 - Go to Position Flow Diagram (Part B)

The entry in the log records the functional unit name, the time and date and the new position that it was moved to. If for any reason it should fail, this is recorded also. After the log entry has been written, the sensor assembly is de-energised and the communications port closed, the program then exits returning the appropriate exit code. The program not only accepts the new position for the functional unit on the command line, but also two optional switches. The purpose of the switches is to provide debugging information to the observer / programmer about what precisely the program is doing at any one moment in time. The switches are -v to put the system into verbose mode and write more detailed information to the console, and -e which produces a file of all the command sequences both sent and received from the program to the microprocessor control cards.

As mentioned previously in this section, the program returns a code that describes the condition of the system upon exiting. This information can be used by the command procedure, if the program is being called by one, to make decisions about progressing through the procedure. The exit codes are tabulated in Table 3-3

Table 3-3

Code	Status
0	OK, program completed successfully
1	Failed to find new position
2	Failed to find home position
3	Sensor failure
4	Failed to initialise communication port or rack failed to respond
5	Failed to read configuration information

3.4 Utility Functions

Apart from the main four main programs to control the functional units, there are a number of other smaller and less complex programs that are needed to make the system operate successfully. These utility programs are a mixture of C programs and shell scripts which call C programs to carry out more complex tasks.

3.4.1 Initialise Program

When the polarimeter control rack is powered on, there is a need to initialise the microprocessor control cards as described in section 3.2 (Stepper Motor Command Language). This is achieved by calling a Bourne shell script which in turn calls a C program to send the appropriate command sequence to the cards. The script initialises the two microprocessor control cards by calling the executable version of initial.c which first initialises the top rack and then the bottom rack as described in section 3.3 (Principles of Operation).

A flow diagram of the principle of operation of initial.c can be seen in Figure 3-11. Once the racks have been initialised, the shell script then deletes all the position files thereby forcing each of the functional units to search for home when they are first used. This ensures that if the mechanics of the functional units within the polarimeter have been moved, say due to work being carried out or vibration in transport, then the system will set itself up into a known state upon initialisation. The shell script then calls the four programs that control the functional units in turn, to position each unit to position 0.00.

3.4.2 Fault Program

When any one of the four main programs are called, they each initialise the communications port that they will use, and then check the status of the rack as described in section 3.3 (Principles of Operation). If the microprocessor control card

returns a status code indicating a fault condition the program will exit immediately and the user can then run a program to determine where the fault may lie. The main use of this program is when one of the limited movement units (i.e. polarimeter focus and grid mask) has been driven to the limits of its travel. The main program will report this error but the will not always successfully de-energise the motors and sensors, so the fault program can be used to check the status and place the polarimeter in a state where the observer may open the sides of the instrument and physically move the unit back within the normal range of its travel.

The program, fault.c, initialises the communications port and then providing the communications initialised correctly, checks the status of the rack. If there is no fault present the program exits immediately after displaying the appropriate message via the console or if there is no fault but the motor is still in motion, the program will send the abort command to stop the motor and then display the message and exit. If there is a fault, the fault status will be cleared by default when the interrogation code is sent and the appropriate message will be displayed to the user depending on the error code returned. The program then de-energises the motors and the sensors before closing the communications port and exiting. A flow diagram depicting the principles of operation of fault.c can be seen in Figure 3-12.

3.4.3 Status Program

The status program is a Bourne shell script that informs the user of the current positions of the four functional units within the instrument. The user may choose to check the status of any one unit by specifying the unit name on the command line as an argument to the command, or all the units by specifying all,

status [hwp filter focus grid all]

The status program also lists the filter names within the polarimeter, the associated filter numbers and the polarimeter focus for the particular filter. This information is

held in a data file and is discussed in more detail in section 3.4.5 (Filter/Focus Program).

```
Polarimeter Status
------------------
H W P at : 0.00
FOCUS at : 0.00
FILTER at : 6.00
GRID at : 0.00

Filter Focus Settings
---------------------
R       0       9.50
V       1       9.40
B       2       9.40
CC      3       9.65
NF      4       9.50
HA      5       9.50
VV      6       9.30
I       7       9.85

Configuration Information
-------------------------
HWP A 1000 0100 FLT B 3000 0100 FOC C 2000 0100 GRD D 4000 0500
```

Figure 3-10 - Status Output Listing

Finally, the status program lists the current configuration file information that details how the functional units attach to the logical units, as described in section 3.3 (Principles of Operation). This is a useful way of obtaining the information when setting up the polarimeter, when the observer needs to know the connection order of the cables. A listing of a typical output of the status program can be seen in Figure 3-10.

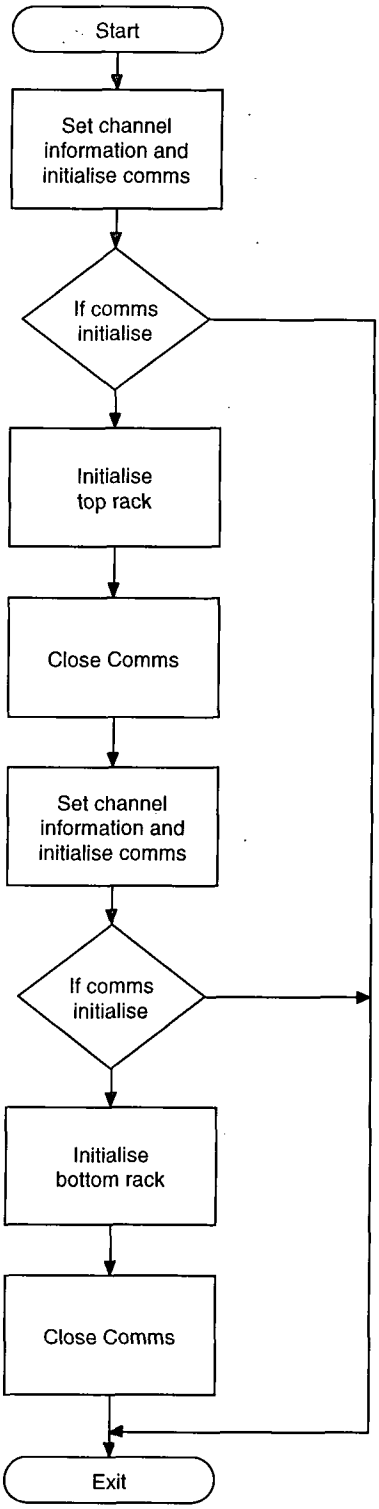


Figure 3-11 - Initialise Flow Diagram

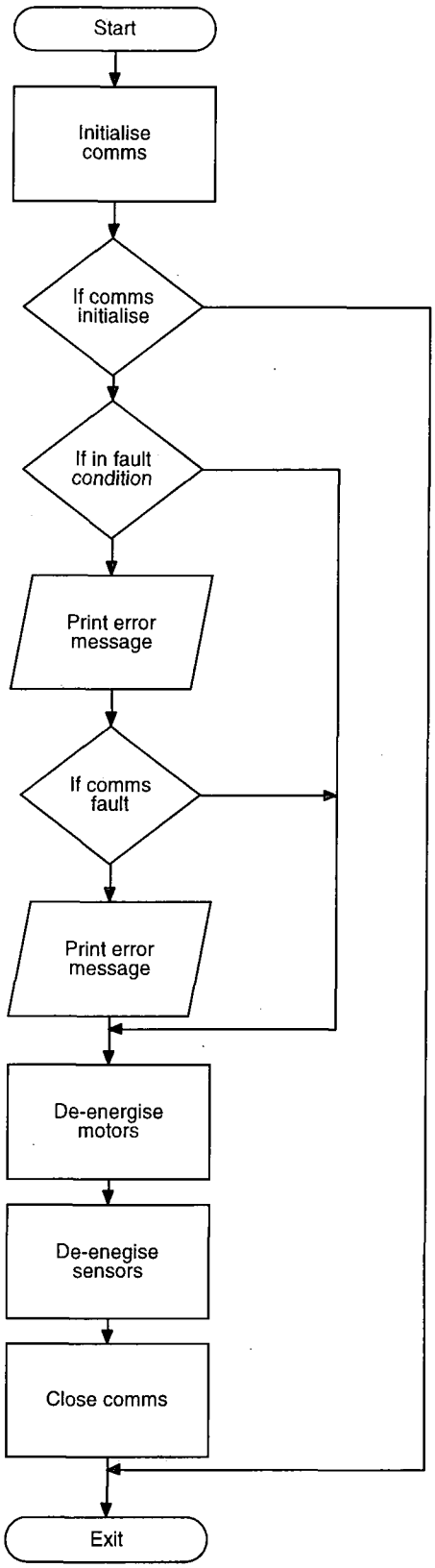


Figure 3-12 - Fault Flow Diagram

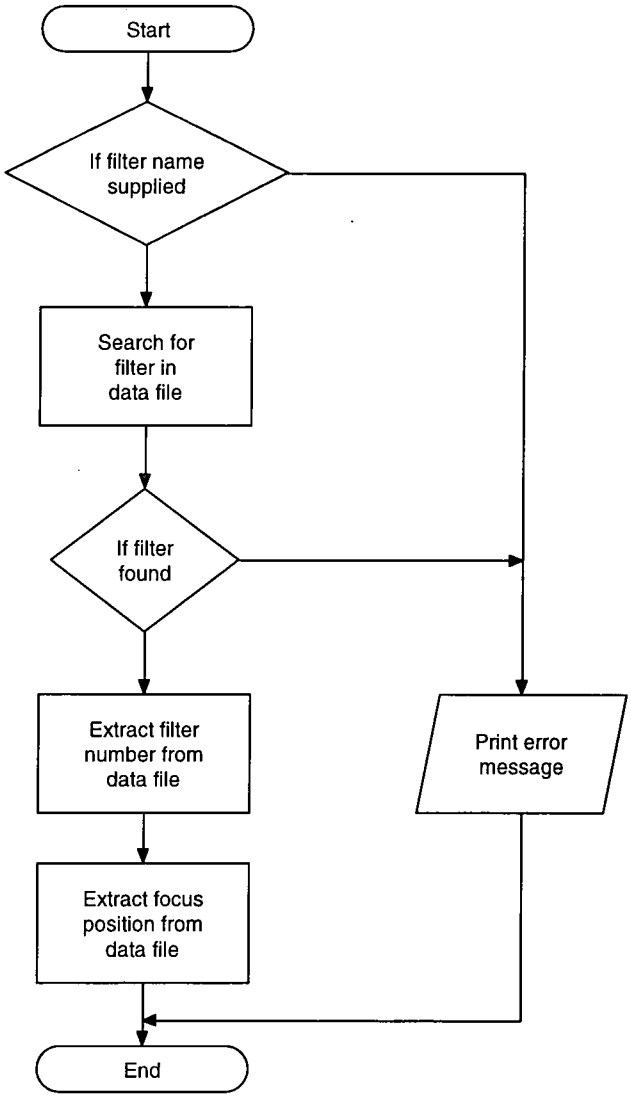


Figure 3-13 - Filter/Focus Flow Diagram

3.4.4 Verify Program

The verify program is a Bourne shell script to check whether each of the functional units are at the position specified in the relevant position files, and if not, inform the observer of the problem and move the unit to the appropriate position. The program is called in a very similar way to the status program, with a functional unit code as the argument,

verify [hwp focus filter grid all]

The script functions by taking a copy of the current position file and then piping the file into the command for the appropriate unit. First the script makes the copy, then it echoes the position to the screen and then calls the program to move the functional unit. If the program completes correctly the script will echo the appropriate message to the screen.

3.4.5 Filter/Focus Program

Due to the nature of the optics within the polarimeter, the focus of the instrument changes depending on the wavelength of the filter in the optical path and therefore with each different filter chosen by the observer, the focus needs to be adjusted. During an observing run it is easy to forget about the polarimeter focus if there are many other things that need to be controlled such as the telescope focus, telescope tracking etc., and the observer may only become aware of the problem when the first images are displayed on the system. To overcome this risk, there is a Bourne shell script to move the filter wheel to the chosen filter and then move the polarimeter focus to the appropriate setting for that filter.

There are two shell scripts used in this process, the first to set up the data file that contains the information about the filter names, the filter reference numbers and the polarimeter focus positions, and the second that extracts the information and moves the functional units to the required positions.

The first shell script is called filt_foc_set and requires the observer to enter filter names and focus positions. This information will have been compiled by the observer during the set-up of the polarimeter and before the instrument is placed on the telescope. The observer uses the grid spot mask (see section 2.2) to determine the focus of the polarimeter by making a series of exposures with different foci and then comparing the images. The image with the spots as near to being circular and with matching left and right pairs is the image with the best focus. When the script is run, it first lists the contents of the current data file and then asks the observer if he/she wishes to delete the old file, if the answer is yes the old file is deleted and a new file created. The script then asks for the filter name for filter 0 and then the focus for this particular filter, and then repeats the sequence up to and including filter 7.

The second shell script is used to extract the information from the data file and move the appropriate units to the desired positions. A flow diagram of this script can be seen in Figure 3-13 - Filter/Focus Flow Diagram. The script starts by checking to see if a filter name was given on the command line, if not, then the observer is prompted for one. The script then searches for the filter name in the data file and extracts the filter number using a UNIX command called awk and stores it in a temporary file. This information is then piped into the filter command to move the filter to the required location. Next the script extracts the focus position from the data file and stores it in the temporary file and then pipes the file into the focus command to move the polarimeter focus to the desired position. The script then returns control to the observer or to the calling program.

3.5 Porting and Future Developments

The polarimeter control software, as previously mentioned, was written on a PC AT computer running a UNIX operating system. Due to the nature of the hardware the

system is a long way out of date but is maintained so as to make use of the Astromed Imager1 package and 2000 series CCD controller. More recently there has been a need to have the software running on different hardware platforms and under different operating systems due to the fact that the research group has purchased a new Astromed 4200 series CCD controller.

The new 4200 CCD controller operates from a 486 or higher PC running the DOS operating system and Microsoft Windows graphical user interface. The controller is based upon the Inmos (now SGS-Thompson) transputer series of microprocessor chips and uses the generic high speed transputer link to communicate between the CCD controller and the PC. Inside the PC there is a dedicated card with a transputer onboard which talks to the transputer in the CCD controller at speeds ranging from five megabits per second (5Mbps) to twenty megabits per second (20Mbps). The link between the two transputers is converted to the RS422 standard so that it maybe driven down cables over 500m in length. With the purchase of the new system described above, it became essential to have the polarimetry control system software on the same platform so that the observer could use the one computer for complete instrument control.

The polarimeter control system software was ported to the DOS operating system using the Microsoft C/C++ version 7 compiler which conforms to the ANSI standard. The major difference between the DOS version of the software and the UNIX version is the way the communication ports are accessed. Under the UNIX operating system device drivers are provided that control the operation of the various hardware interfaces on the computer and allow programs to make calls to high level functions that use these interfaces. The DOS operating system does not have this level of sophistication and only allows programs to call very low level routines that may only handle single characters at a time. There is therefore a need to write a higher level library of function calls that allows the programs to communicate with the interfaces or to use a third party library. The author opted to use a third party library that gave similar function

calls to the serial interfaces as those found in the UNIX operating system. With the exception of the communication interface calls, there was very few changes to be made to the software to complete a successful port to the DOS operating system, most of the other changes were due to differing syntax between the compilers on the two PC's.

The next development with the current polarimeter is the migration towards using a Sun IPX workstation as the controlling computer instead of the PC. The main advantage of this system would be the ability to analyse the data taken during the observing run, either at the time of observing or during the next day, without incurring any overheads in equipment weight or size. With this facility there will be less time wasted observing potentially good polarised candidates that upon analysis show little or no useful results.

The port of the polarimeter control system software to the Sun platform and the SunOS 4.1 operating system was successfully completed using the standard C compiler supplied with the operating system and using no additional libraries. The changes to the code that were required, again like in the DOS port, were in the communications calls in the module comm.c. These changes were very minor and only affected the communications interface initialisation.

4. Testing and Observing.

4.1 Introduction

The function of the Durham polarimeter, as previously mentioned, is to measure the levels of polarized light from astronomical objects. No matter how well an instrument is designed and constructed it cannot be considered serviceable until it has been tested and the user knows it gives acceptable results. With respect to the Mark IV polarimeter it was already known that the optical system functioned correctly due to it being the same as the previous model and so initial testing required checking that the motors positioned the functional units in the correct position. This chapter outlines the testing of the polarimeter using software to test the repeatability of the motors, by using standard polarized stars to determine that the instrument was functioning as a polarimeter and by observing a known polarized reflection nebular to determine the mapping ability of the polarimeter.

The polarimeter is used on many different telescopes ranging from the large to the small in both the northern and southern hemisphere and in the final section of the chapter there is a brief description of the individual characteristics of different telescopes upon which the Durham polarimeter is regularly mounted.

4.2 Testing the polarimeter

4.2.1 Software tests

The first stage of testing the completed Mark IV polarimeter was to test the motors and make sure that they would move each of the functional units to the correct positions repeatably. A set of 'C' shell script routines were written to exercise each of the four functional units by moving them to consecutive positions in sequence, starting from 0 and working up to the limit of the units travel. The output from the program was allowed to print to the screen and the log of all the movements of the functional units was used to check for any errors that occurred during each procedure, either returning to home or going to the new position.

The results of the tests showed that each program would position the functional unit to the desired location over 99% of the time without any problems. Occasionally the unit would show an error when returning home indicating that the unit had possibly not been sent to the correct position during the previous movement. When this was investigated, it became apparent that the unit could only have been positioned incorrectly by a maximum of two steps, which in the case of the half wave plate, would translate to 0.06°. This was considered to be acceptable as the only functional unit that would be adversely affected by this problem would be the half wave plate. The observer would be aware of this when the H.W.P. was moved the next time, and discard the previous frame as a matter of precaution. The accuracy of the positioning of each functional unit is described in section 2.3 and is to one step of the stepper motors divided into the gear ratio.

4.2.2 Astronomical tests

The testing of the polarimeter using astronomical data was carried out for two purposes, first using the standard polarized stars to check that the instrument was

operating as a polarimeter, and secondly using an extended object to check the mapping ability of the instrument.

4.2.2.1 Standard polarized stars

Three stars were chosen from the paper by Hsu and Breger (1982) that had been measured for the percentage polarisation and position angle using a photomultiplier tube as the detector. The use of a photomultiplier is more suited to measuring these types of objects whereas using a CCD as the detector has limitations as it is primarily suited to extended objects. Axon (1977) determined that the earlier version of the Durham polarimeter, using an electronographic camera as the detector, was capable of measuring the polarization of standard stars to ±0.5%.

The stars were chosen from the list (Hsu and Breger 1982) to suit the position in the sky and polarization levels. All three stars have different spectral types and one is a variable (HD198478) although for the purpose of these tests, it was decided that it would still give accurate enough results. The magnitudes, in the *V* band, of the three stars ranged from 4.8 down to 7.9 and therefore required very short exposures to avoid saturating the CCD camera with charge that would take time to clear. Exposures of 0.5 seconds with the telescope defocused was enough to carry out the required measurements for each object. The defocusing of the telescope allowed the instrument to emulate the photomultiplier detector in that it did not record precise structure just the overall quantity of photons.

The data were reduced in the standard manner (Draper 1988) but rather than plot a polarization map simulated aperture polarimetry was carried out. The aperture for each of the standard stars was large enough to enclose all of the defocused star without including any unnecessary background. The results (Table 4-1) agree to the figures calculated by Hsu and Breger within the given errors of the system.

Table 4-1 - Standard Stars

Star	HD154445	HD198478	HD204827
R.A.	17 04 45.7	20 48 25.6	21 28 31.7
Dec.	-0 52 23	+46 03 28	+58 40 25
$\mathbf{m_v}$	5.63	4.83	7.93
Sp.	B1 V	B3 Ia	B0 V
Aperture (*pixels*)	10	15	10
$\mathbf{P_{max}(\%)(Hsu)}$	3.73 ±0.01	2.72 ±0.02	5.70 ±0.03
$\mathbf{P_{max}(\%)(Dur)}$	3.80 ±0.18	2.83 ±0.25	5.42 ±0.26
$\mathbf{\theta_v(1985)(Hsu)}$	90.1 ±0.1	3.2 ±0.2	59.3 ±0.1
$\mathbf{\theta_v(Dur)}$	89.9 ±1.4	4.9 ±2.6	61.3 ±1.4

4.2.2.2 Hubble reflection nebula NGC2261

The reflection nebula NGC2261 was chosen as the test object as the group had carried out previous observations on the object and therefore had plenty of data with which to compare against (Scarrott, Draper and Warren-Smith 1989).

NGC2261 is a variable reflection nebula illuminated by the star R.Mon and an example of a cometary nebula where the system is tilted so the northern lobe is towards our line-of-sight and the southern lobe is obscured by dust and gas. The optical nebula is part of the much larger system consisting of a bipolar molecular outflow driven by R.Mon and collimated by a circumstellar disc around the star.

The main polarization pattern shows the typically expected centro-symetric vectors centred on the illuminating source R.Mon. Earlier imaging polarimeter data of the head of the cometary nebula (Gething *et al.* 1982) showed polarization vectors aligned parallel and centred on the star and where the two patterns meet, null points indicating two competing polarization mechanisms at work. Gething *et al.* attributed the parallel vectors to dichroic extinction by aligned grains in the disc or torus, aligned by the Davis-Greenstien effect. Subsequent observations showed that the orientation of the

disc and the polarization angle of the star changed over short periods of time i.e. time scales of a year (Scarrott, Draper and Warren-Smith). This disproved the idea that the polarization mechanism in the disk was aligned grains as the time scales for magnetic grain alignment is too great. Even if the magnetic field was tightly coupled to the disc and the disc was in the process of re-orientating, the time scale would be too large to explain these observations.

Notni (1985) describes the idea of a polarized source based on the model by Elsasser and Staude (1978), where the source emits linearly polarized light and is subsequently scattered. This however, does not explain certain observational data such as the nebulosity immediately surrounding the star (Menard, Bastien & Roberts 1988) and uniform variations of the polarization angle of the R.Mon with wavelength. Scarrott *et al.* (1989) describe a model based on competing polarization mechanisms which behave in differing ways depending on wavelength and locality. The primary mechanism of the model is scattering from an initially polarized source with the possibility multiply scattering in the disk. The secondary polarization mechanism is extinction due to aligned grains which are described as if the object is being viewed "through a screen of aligned grains".

The data for the calibration and comparison was collected on the JKT in La Palma in November 1992 on the first observational run with the Mark IV polarimeter. The polarization map (Figure 4-2) presented in this thesis is not intended to add to the science but rather as a comparison of the Mark IV polarimeter against the older Mark III polarimeter for which a map of R.Mon is shown in Figure 4-1.

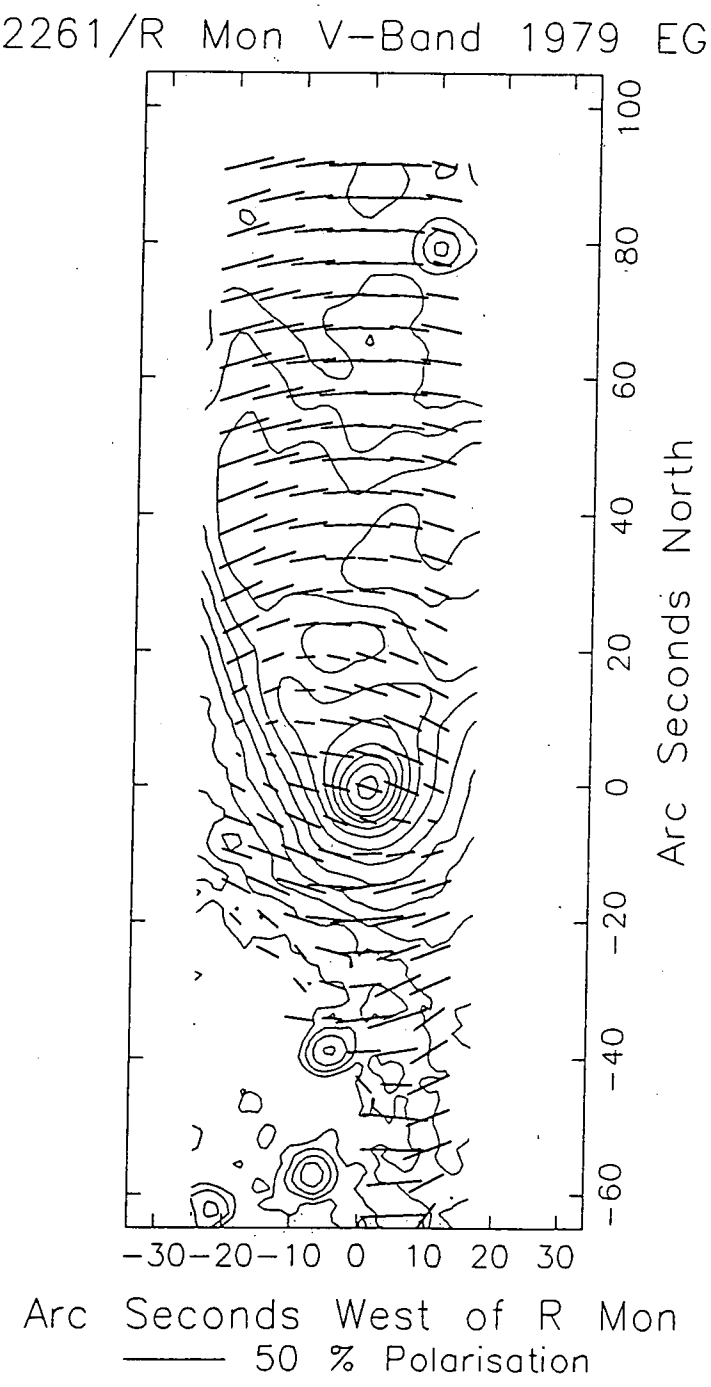


Figure 4-1 - NGC2261 (Mark III polarimeter)

(Taken from Draper 1988)

The fact that the northern lobe of the nebular shows the centro-symetric pattern commonly associated with reflection nebular made the object a useful test of the functionality of the polarimeter and also allowed the calibration of the instrument. Knowing the orientation of the polarization vectors of the object allowed the current mounting angle of the Wollaston prism to be determined. Certain aspects of the object did not make it ideal as a test due to the variations in both the percentage polarization and the polarization angles of the disk region. It was felt that even though this was a variable object, it was still adequate for checking the functionality of the polarimeter and that the variations that were apparent in the disk would be due to its variability as opposed to problems with the polarimeter.

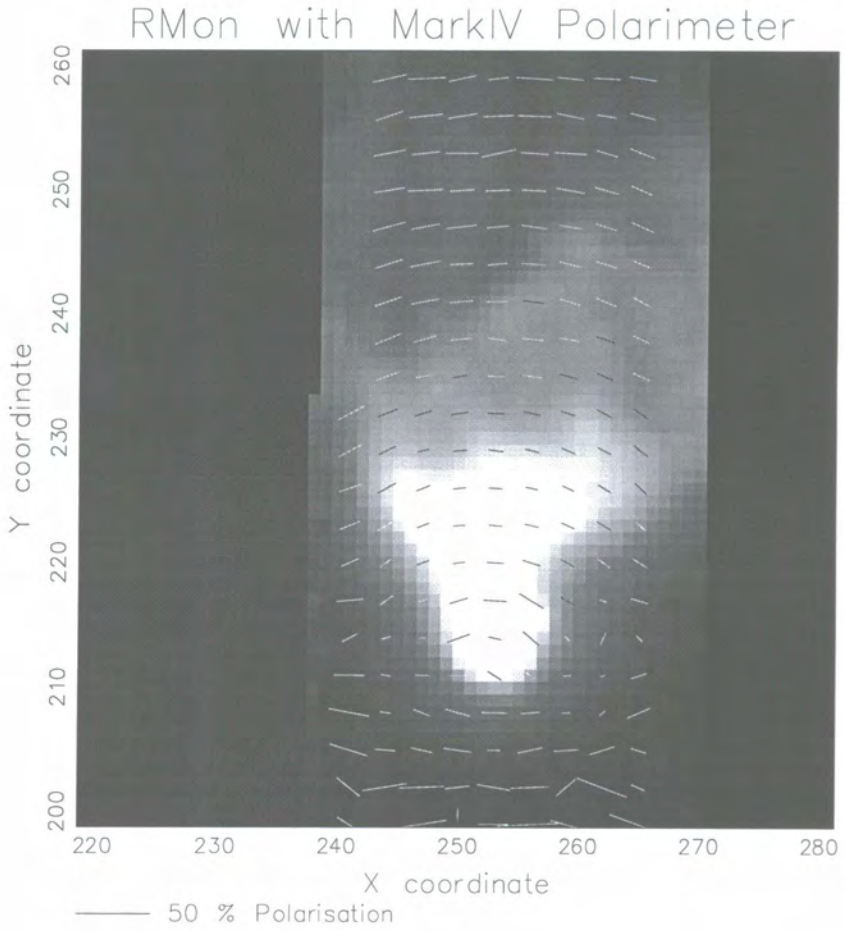


Figure 4-2 - NGC2261 (Mark IV polarimeter)

4.3 Telescopes and the Durham polarimeter

The Durham polarimeter has been used successfully on a number of telescopes around the world. One of the greatest advantages of moving the instrument to the different observatories is the ability to use a telescope that best suits the objects in a particular project. The Durham polarimeter has been used to measure the polarization of light from all types of astronomical objects from Jupiter to high redshift radio galaxies and is equally suited to all of these depending on the host telescope.

The two main factors that determine which telescope will suit a particular project are latitude of the telescope and the primary mirror size which in turn determines the resolving power and the image scale. The larger telescopes such as the William Herschel Telescope at the Observatorio Del Roque Muchachos De La Palma in the Canary Islands or the Anglo-Australian Telescope at the Siding Springs Observatory in Australia are good for doing small and/or faint objects such as radio galaxies, proto-planatary nebular and planetary nebular that have faint shells. The smaller telescopes such as the Jacobus Kapteyn Telescope in La Palma and the 40" telescope at the South African Astronomical Observatory in Sutherland, Cape Province are useful for observing larger starburst galaxies and large-scale reflection nebulae.

4.3.1 Jacobus Kapteyn Telescope

The Jacobus Kapteyn Telescope in La Palma is a 1 metre reflecting equatorially mounted telescope situated at 28° north and at an altitude of 2646 metres above sea level. The polarimeter is mounted at the Cassegrain focus with a focal ratio of f/15 which gives an image scale on the polarimeter of 1.2 arcseconds per pixel. The setup is ideal for looking at spiral galaxies, starburst galaxies and reflection nebulae down to

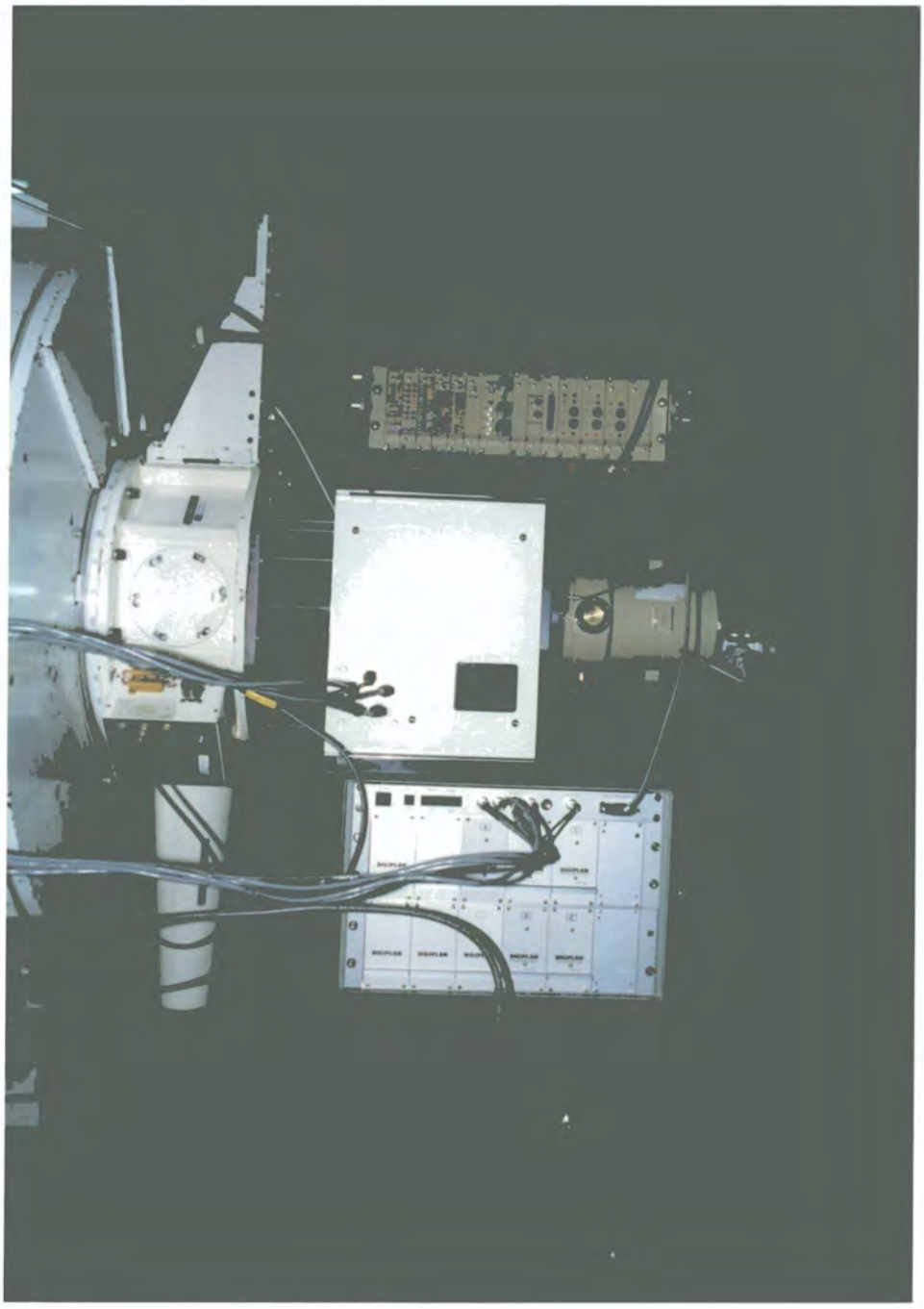
a magnitude of 15th and with an angular size of no less than 30 arcsec. It is possible to observe objects in the both the northern and southern hemispheres but are limited to objects no lower than -30°. Photograph 4-1 shows the Mark IV Durham polarimeter mounted on the J.K.T.

4.3.2 William Herschel Telescope

The William Herschel Telescope at the same site as the J.K.T. in La Palma is a 4.2 metre reflecting altitude-azimuth mounted telescope. The polarimeter is mounted to the f/11 Cassegrain focus which gives an image scale of 0.4 arcseconds per pixel. This telescope, due to the collecting power of the large mirror, makes it ideal for looking at very faint objects such as high redshift radio galaxies, quasars and the faint shells of proto-planetary nebula. The Mark III Durham polarimeter has been used on the W.H.T. to look at radio galaxies with magnitudes as faint as 21st, but at the time of writing this thesis, the current Mark IV polarimeter had not been used on this telescope.

4.3.3 Anglo-Australian Telescope

The Anglo-Australian Telescope is located at the Siding Springs Observatory in New South Wales, Australia at an altitude of approximately 800 metres above sea level and --33° latitude. It is a 3.9 metre refracting equatorially mounted telescope with a focal ratio of f/15 at the Cassegrain focus which gives an image scale of 0.3 arcseconds per pixel. The objects suited to the polarimeter on the A.A.T. are much the same as those on the W.H.T. but with a southern hemisphere bias. Using this telescope, objects with approximate declinations as far north as 28° can be observed.



Photograph 4-1 - JKT and Durham polarimeter

5. Starburst Galaxies

5.1 Introduction

Galaxies that are undergoing intense periods of star formation are known as starburst galaxies. It is believed that individual star formation events typically last for no more than 10^8 years where upon all the gas in the interstellar medium of the galaxy has been converted to stars (Rieke *et al.* 1985). The starburst activity normally occurs in central regions of the galaxy, although there are examples of starburst activity in the arms of spiral galaxies (Pildis *et al.*1994). Starburst galaxies are often considered as a subset of the much fuller group known as far-infrared galaxies (FIRG) although the starbursting phenomena is sometimes seen in other types of galaxies such as small blue irregulars and dwarf emission line. These FIRGs emit excessive radiation between 30μm and 100μm[2] and can be selected on the basis that their far-infrared luminosity is around 10^{11} - 10^{12} *Lo*, and that their far-infrared to blue luminosity ratios are between 10-100 (Soifer *et al.* 1984). With the launch of the Infrared Astronomical Satellite (IRAS) in 1983, the first all-sky infrared survey with sufficient sensitivity to detect many of these extra-galactic sources was conducted. Due to the nature of the earth's atmosphere most of the radiation (λ > 20μm) from these objects is blocked and therefore the satellite observatory was able to discover this almost unknown group.

[2] Infrared spectral regions as generally used are as follows: Near-IR 1 - 5μm, Mid-IR 5 - 30μm, Far-IR 30 - 300μm, submillimeter 300μm - 1mm

One of the main conclusions of the IRAS survey was that far-infrared emission dominates the luminosity in a significant percentage of all galaxies.

There has been a lot of work carried out in this field of astronomy and this chapter will discuss infrared emission, starburst galaxies, and the phenomena known as superwinds.

5.2 Infrared Emission

Radiation within the infrared spectral region carries information about the physical processes occurring deep within galaxies. Because dust is relatively transparent to the infrared radiation, it is often used to probe deep into star forming regions or dusty galactic halos.

Infrared emission from galaxies is generated by a number of different processes such as the heating of dust by young stars. Dust in galactic star forming regions is very efficient at converting visible radiation into infrared radiation and due to this process, the luminosity in the spectral range 5 - 300μm is often a good approximation to the bolometric luminosity of the young stars population within star forming regions. Telesco (1988) compared the infrared luminosity's of three well known IR-luminous galaxies, M82, NGC 1068 and Arp 220 and noted that even though the significance of the young star population in the generating of the infrared luminosity varied between the galaxies, the energy distribution of all three had the same general shape and had broad maxima between 50 - 200 μm which is characteristic of heated dust. In galaxies where there are very high star formation rates e.g. M82 and NGC 253, the spectra of these young stars can often dominate over the other mechanisms. Generally, luminosity at $\lambda > 30\mu m$ is emitted by dust mixed with molecular gas whereas luminosity at $\lambda < 30\mu m$ is emitted by dust mixed with ionised gas and heated by resonantly scattered Lα photons.

Armus *et al.* (1989) in a study of 53 far-infrared galaxies concludes that the young stellar population of a galaxy can typically provide at least 25% of the energy needed to power the far-infrared emission and that from models and observation of H_{II} regions/ molecular cloud complexes of our own galaxy, they note that the far-infrared radiation from the FIRG's is due primarily to heated dust grains.

Soifer *et al.* (1987) sampled over 300 IRAS extragalactic sources with 60μm flux densities greater than 5.4Jy to determine the significance of the far-infrared luminosity in these objects. All of the sample showed flux densities that could not be attributed to a stellar population alone and again, none of the sample were radio-loud where the far-infrared emission could be an extension of the radio non-thermal emission. The authors note that many of the objects are spatially extended at 60μm and therefore conclude that the emission is from heated dust grains. They calculated that to generate far-infrared luminosity's of $\sim 2 \times 10^{10}$ *Lo* would require a mass of dust $\sim 4 \times 10^{6}$ *Mo* at a temperature of ~ 35k assuming optically thin dust emission and normal dust parameters (e.g. Draine and Lee 1984). If we assume that the dust/gas mass ratio is in the same ratio as the Galaxy, then this would give a corresponding gas mass of $\sim 10^{9}$ *Mo* which is considered typical for the interstellar medium of a spiral galaxy.

There is very little direct evidence to quantify young stellar populations in far-infrared galaxies although the survey conducted by Armus *et al.* (1989) did detect a possible five galaxies that showed evidence of the broadened emission due to Wolf-Rayet stars. Wolf-Rayet stars are the post main-sequence product of massive stars which live for approximately $x10^{6}$ years and there presence usually indicates a young stellar population. These stars have been found increasingly in starburst galaxies but quite infrequently in far-infrared galaxies in general, though this is probably due to dust obscuration. The main evidence for a young star population comes indirectly from bright Hα emission (Young *et al.* 1988) caused by the gas at the centre of star forming regions being photoionised by luminous stars within energetic H_{II} regions. Armus *et al.* (1989) conclude that approximately 40% of their sample showed emission line

fluxes which are due to hot young stars whereas 45% of the sample had spectra that could possibly be attributed to active galactic nuclei (i.e. a non-stellar ionising source). The remaining objects in the sample were unclassified due to having too few detectable emission lines.

It is thought that in the more luminous galaxies, active galactic nuclei (AGN), as well as the young star population, play a part in heating the dust that radiates in the far-infrared. The Seyfert galaxy NGC 1068 is one of only a few galaxies where the roles of the AGN and OB stars to the contribution of infrared emission is known. The energy from the Seyfert nucleus peaks just below the far-infrared band between 20 - 30μm and contributes approximately half of the infrared luminosity. The remaining luminosity originates from the central few kiloparsecs of the galaxy and is due to young stars.

5.3 Starburst Galaxies

Star formation in galaxies can often have a profound effect on the evolution of that galaxy. The process of converting interstellar gas into new stars which in turn transfers energy back into the galaxies interstellar medium (ISM) to generate more stars, or in some cases causes material to be expelled out of the galaxy completely, will shape the galaxies future. Galaxies that have intense periods of star formation, as mentioned in the introduction, are known as starburst galaxies. The starbursting period is so intense that it is thought to be either episodic or short lived, lasting timescales of approximately 10^8 years when it is likely that all the star forming material is exhausted. In a study of the two starburst galaxies ARP 220 and NGC 6240, Rieke *et al.* (1985) made observations that showed that in each galaxy, there was nearly 10^{10} *Mo* of newly formed stars. The galaxy M82 is often considered to be the archetypal starburst (see Figure 5-1) but is only moderately luminous (3×10^{10} *Lo* at 5 - 300μm) with stars

Figure 5-1 - M82

This image was obtained from the Digitized Sky Survey.

The Digitized Sky Surveys were produced at the Space Telescope Science Institute under U.S. Government grant NAG W-2166. The images of these surveys are based on photographic data obtained using the Oschin Schmidt Telescope on Palomar Mountain and the UK Schmidt Telescope. The plates were processed into the present compressed digital form with the permission of these institutions.

forming at a rate of a few solar masses per year which is comparable to the whole of the Milky Way but in a volume several thousand times smaller (Telesco & Harper 1980). Even though this may be considered great, compared to the ultraluminous galaxy (luminosity > 10^{12} *Lo*) IRAS 10214+4724 the star formation rate of M82 is small. This IRAS source is considered to be forming stars at a rate several thousand times greater than the starburst in M82 and is discussed by Rowan-Robinson *et al.* (1991) and Brown & Vanden Bout (1991), though this galaxy has subsequently been shown to be gravitationally lensed (Broadhurst & Lehar 1995).

Starburst galaxies are usually observed in the optical wavelengths as having clusters of bright spots which are often associated with the starbursting region producing hot spots in the central kiloparsec. These bright areas can be attributed to variations in stellar population and density or even extinction although spectroscopic observations have shown that the hot-spots are in fact highly luminous H_{II} regions. This category of galaxy are known as Sérsic-Pastoriza galaxies and are defined as galaxies with morphologically peculiar nuclei that exhibit a change in the slope of the luminosity profile across the nucleus (Sérsic & Pastoriza 1965).

Due to the fact that the starburst periods are relatively short in terms of the Hubble time and are probably episodic, they obviously need to be initialised by some form of energy input. The general consensus of opinion is that mergers and gravitational tidal interactions between galaxies are the mechanism for starting the starbursting phase. Larson and Tinsley (1978) studied various galaxies, both interacting and non-interacting, in the U, B and V colours and concluded that tidal interactions increased the rate of star formation. Work carried out by Rieke *et al.* (1980) in modelling radio, infrared, ionising flux and X-ray properties in two well known starburst galaxies, M82 and NGC 253, showed that the starburst process could account for the large infrared excesses in galaxies. It is becoming increasingly evident that galaxies that are undergoing the strongest interactions are the ones with the highest far-infrared luminosities. Numerical simulations of galactic collisions have been carried out by

various groups (Noguchi 1991, Mihos & Hernquist 1994) which show that collisions can provide the right conditions for starbursts to occur i.e. energy and gas (there is a need for excesses of gas to be present in the starburst region which will be discussed later). The collision of the gas clouds of the galaxies are inelastic and subsequently dissipate energy in the form of shock fronts that cause the gas to collapse towards the gravitational centre where the star forming regions are located. Earlier work be Hernquist (1989) concluded that a collision between a small satellite and a more gas-rich companion would be enough to cause the required inflow of gas towards the centre of the larger body and that timescales of $\leq 10^8$ years would be enough for 35% of the material from a large spiral to enter the central 400pc of the nucleus. The result of such an interaction would be the destruction of the satellite and the distortion of the disk of the spiral galaxy.

The ISM of a galaxy consists of large gas and dust clouds, electromagnetic radiation fields, cosmic rays and probably a large amount of dark matter, and is the building area of new stars which eventually evolve and recycle their material back into the medium. It is the general consensus of opinion that far-infrared galaxies contain as much molecular gas as they do atomic gas within the interstellar medium (Sanders *et al.* 1991). Work carried out within our own galaxy into star formation has helped in the understanding of the processes within the far-infrared galaxies and has shown that there is a strong connection between regions with an abundance of molecular hydrogen gas (H_2) and star forming regions. The gas, which seems to collect in regions of enhanced gas density within the inner kiloparsecs of the galaxy, is the major ingredient required for the newly forming stars and is part of the quite complex ISM of a galaxy. Originally it was thought that the ISM consisted of two phases, a cold (100K) phase and a warm (10^4K) phase (Field *et al.* 1969) but with the discovery of diffuse soft X-ray emission and the O_{VI} absorption lines within our own galaxy, a new hot 'coronal' phase (~10^6K) was added to the model. With the discovery of this new phase, it was clear that another phase had to exist between the warm and hot phases, and this was

named the warm ionised medium. The current ideas of the ISM are based on a five phase model as summed up in Table 5-1.

Table 5-1 - Gas Phases of the ISM

Phase	Temperature [K]	Density [atoms cm^{-3}]	Filling factor	Typical gas component
Cold	20 - 80	10 - 1000	< 0.1	molecular gas
Cool	80 - 300	0.1 - 10	0.1	neutral hydrogen
Warm neutral	~ 1 - 2 x 10^3	0.05 - 0.1	0.2 - 0.5	diffuse (ionised) hydrogen
Warm ionised	~ 10^4	0.05 - 0.1	0.2 - 0.5	diffuse (ionised) hydrogen
Hot	10^6	< 0.01	0.5 - 0.7	hot 'coronal' gas

The cold neutral medium (CNM) is divided into the cold and the cool phases, with the cold phase consisting of dense molecular gas clouds and the cool phase neutral hydrogen which is usually concentrated into filaments and shells. The warm neutral medium (WNM) consists of approximately half of all the atomic hydrogen in the ISM and is found either surrounding the cold dense clouds of the CNM or partially filling the space between the clouds. The warm ionised medium (WIM) consists of diffuse ionised gas and some H_{II} regions. The ionised hydrogen is often found widely spread over the galactic disk and is heated by young, massive OB stars. The hot ionised medium (HIM) is associated with the hot 'coronal' gas which can be detected in the soft X-ray part of the spectrum and the UV absorption lines. It is thought that this gas is heated by supernovae.

As discussed above, the molecular gas collects in dense areas where the star forming regions evolve. The physical mechanisms that are in action to transfer the material to the inner areas are unclear but quite often these starburst galaxies are noted as having bars which are thought to be involved with the transportation of gas into the central regions. Bars are observed in approximately 30% of all spiral galaxies and are indeed distinctive enough to be part of the Hubble galaxy classification, although quite often there is some weaker or obscured form of bar in other types of galaxies. Often galaxies that are undergoing weak tidal interactions (non-mergers) are the ones with bars and this is thought to be due to gravitational instability, although there are examples of galaxies that are barred and do not appear to be undergoing any interactions or have any neighbours close enough to interact with. At the present time, it is unclear how these systems may come about.

The bar of the galaxy rotates rigidly about an axis normal to the plane of the disk and material is thought to move in elongated orbits aligned with the rotating major axis of the bar. At certain points along the bar, resonant orbits are set up caused by the interaction of the rotation of the bar and the stellar orbits. These points are called the inner and outer Lindbald Resonance's (ILR & OLR) and it is at these points that the gas builds up into dense regions. Gas is thought to move in various directions in the bar but a detailed discussion of the gas kinematics is beyond the scope of this thesis. It is thought that this mechanism is responsible for the manifestation of rings of star formation near to the gravitational centre of the galaxy.

In a sample of radio-bright galaxies, Condon *et al.* (1982) carried out sub-arcsecond to arcsecond imaging using the VLA and found that the emission from FIRG's was approximately 10 times greater than for normal spiral galaxies and cospatial with the starburst. The galaxies in the sample that were viewed edge-on showed double or triple emission peaks that are thought to correspond to the ring of young stars and with information about the radio continuum and the steep spectral slope, it can be assumed that an extended non-thermal source is responsible for the emission. The most likely

scenario is that star formation creates hot massive stars that heat the dust to produce far-infrared emission, these stars eventually explode as supernovae which then produce the radio emission by accelerating electrons in the magnetic field of the disk. Evidence for supernova remnants is also seen in the work carried out by Saikia *et al.* (1994) in observations of Sérsic-Pastoriza galaxies carried out using high resolution imaging on the VLA radio telescope. They observed compact, spectrally steep features breaking through the radio emission, something that is usually associated with young bright supernovae remnants.

5.4 Superwinds

The ideas behind galactic-scale winds or superwinds as they have become known have only been formulated during the last decade or so, with Chevalier and Clegg (1985) possibly being the first to analyse the connection between star formation at accelerated rates, the collective effect of supernovae, and the start of a superwind. Superwinds are not such common phenomena in galaxies as bars or warped disks and this is probably due to the large amounts of energy required to heat the gas and drive it into the galactic halo.

The galactic interstellar medium was considered to be reasonably static until the discovery of the hot gas component where upon it was viewed as being dynamic. This hot ionised gas, heated by supernovae explosions and possibly shocks from massive-star stellar winds, creates a bubble due to the difference in pressure between the starburst region and the non-starburst ISM, which starts to expand in the direction of the steepest pressure gradient. Usual the steepest pressure gradient lies along the minor axis (i.e. normal to the disk) of the galaxy, and as the bubble expands, it pushes against the ambient gas of the disk and the halo. The bubble expands out towards the halo pushing with it the cooler gas of the ISM which forms a more dense shell at the interface of the two systems, and with it an area of turbulence inside the bubble.

Eventually, the hot gas breaks out from the shell and into the halo and leaves behind fragmented cooler gas which forms into filaments.

There have been for many years, theories that show a strong connection between the gas in the disk and halo of our own galaxy due to the effect of galactic fountains (Shapiro & Field 1976) and these ideas were further developed in the model by Norman and Ikeuchi (1989) in which hot gas reaches the halo via 'chimneys' that are formed by the expanding superbubble. With the study of the nearby starburst galaxy M82 came possibly the first indications that the superwind theory was operating as a result of the starburst phenomena, and subsequently many more starburst galaxies have been discovered as having superwinds.

As the bubble of hot ionised gas breaks out into the halo, it forms two cavities in the form of a bi-polar cone although the counterlobe feature is often obscured from view due to the orientation of the galaxy. Suchcov *et al.* (1994) created 2D hydrodynamical models that confirmed that the galactic-scale wind was indeed capable of creating this bi-cone effect and that the cavities would progressively fill with the filamentary material that had been created at the bubble interface. Suchcov *et al.* also concluded that the filamentary material should be a powerful source of soft X-rays which in practice, are often conspicuous as X-ray halos expanding along the minor axis in many starburst galaxies.

The shell of shocked ambient gas at the interface of the two systems produced as the bubble expands and breaks out into the halo forms the edges of the cavity, and as mentioned is a source of soft X-rays. As well as the soft X-ray component, the shocked gas also emits strongly in the optical emission lines and is observed as Hα emission. Armus *et al.* (1990) carried out observations of 40 far-infrared galaxies and detected diffuse emission line gas in loops and filaments that extended out from the starforming regions by several tens of kiloparsecs. The galaxies in the selection that were observed edge-on showed a preference to outflows along the minor axis as would be expected due to it being the path of least resistance. The hot gas that forms the

bubble is itself a source of hard (> 1KeV) X-ray emission thought to be produced by shocks within the turbulent gas.

Eventually the superwind will sweep clean the circumnuclear regions of all obscuring gas and dust were upon the starburst phase will die due to a lack of material, although the exact process of how the starburst / superwind terminates is still unclear.

6. Polarization of Starburst Galaxies

6.1 Introduction

The study of starburst galaxies both spectroscopically and polarimetrically, has gained a great deal of information from the nearby galaxy M82, which is by many considered to be the archetypal starburst. As more information is gained about far-infrared galaxies, starbursts and superwinds, it is becoming clear that other galaxies show similar properties to those of M82. As well as similarities, there are also differences being observed which begs the question "How typical is M82?". Due to M82s proximity and size there is obviously much information to be gained from its study but there are many more starburst galaxies worth observing and the purpose of this chapter is to describe the polarization studies of three not quite so famous starburst galaxies.

This chapter will briefly describe the mechanisms that lead to light from astronomical objects being polarized, before discussing the polarization properties of some other far-infrared galaxies. The three galaxies presented, NGC 1808, NGC 3256 and NGC 2146 are all undergoing starbursts and possibly have galactic-scale winds. In the case of NGC 2146, there is an added bonus that it is a merger system which by many is thought to be the mechanism that initiates the starburst period .

6.2 NGC 1808

6.2.1 Introduction

The southern galaxy NGC 1808 (see Figure 6-1) is a high inclined spiral with a classification of Sbc pec and is located at a distance of 10.9 Mpc assuming $H_0 = 75$ Kms^{-1} Mpc^{-1} (Sandage & Tammann 1987) with an inclination of 57° (Reif *et al.* 1982). Visually the galaxy is observed to have bright 'hot-spots' and is classed as a Sérsic-Pastoriza galaxy (see 5.3). Véron-Cetty and Véron (1983) presented a real-colour image of the central region that showed that these 'hot-spots' were extremely blue and corresponded to bright H_{II} regions which are assumed to be areas of star formation. Radio observations at λ = 6cm shows that there are a number of compact radio sources in the central region of the galaxy which Saikia *et al.* (1990) attributed to supernovae remnants.

NGC 1808 falls into the category of far-infrared galaxies with a total luminosity of $L_{IR} \approx 2 \times 10^{10}$ *Lo* (Danks *et al.* 1990). Infrared emission in the IRAS 12μm band is mostly from the nuclear region i.e. 69% at 10.8μm, with less than 25% of the luminosity deriving from the suspected Seyfert 2 nucleus. Junkes *et al.* (1992) also reported that the nuclear regions of the galaxy show X-ray emission observed using ROSAT. This would be expected in a starburst galaxy where there are large numbers of supernova explosions of massive young stars.

The galaxy is well known for the dusty filaments that emerge from the central part of the disk towards the north-easterly direction. Due to the orientation of the filaments, one of the spiral arms is obscured which indicates that the filaments reach out of the disk plane. Burbidge and Burbidge (1968) found that the disk contained unusually large amounts of dust which in the NE region seemed to be running radially, and later Arp and Bertola (1970) speculated that these dust lanes could be due to "the passage of compact bodies outwards from the nucleus". Koribalski (1993) found that there was

an anomaly in the velocity of the gas / dust in the same region which indicated an outflow of neutral hydrogen gas into the halo. The idea that a superwind generated by the starburst regions in NGC 1808 is responsible for the outflow is quite likely and with the aid of imaging polarimetry, hopefully more evidence can be collected.

6.2.2 Observational Details

NGC 1808 was observed in February 1993 with the Anglo-Australian Telescope, using the Mark IV Durham Imaging Polarimeter (see also Scarrott *et* al. 1983)and the EEV P8603 CCD blue-coated detector. The seeing conditions were approximately 2 arcsec and a sequence of 16 frames, 12 exposures of a duration 600 seconds and 4 exposures of a duration 200 seconds, were taken with a V-waveband filter in the beam. The 12 exposures gave full spatial coverage of the galaxy but with a saturated nucleus and so the short exposures were used to fill in the lost data. The data were reduced in the standard manner (Draper 1988).

6.2.3 Discussion

The large scale spatial features of NGC 1808 are shown in Figure 6-3, a grey-scale image on which are superimposed an intensity contour map of the nuclear region and a polarization map of the whole galaxy. From the grey-scale image and contour map it is easy to see the optically bright 'hot spots' where it is thought that starbursts are occurring, and to the west of the major axis of the galaxy, the spiral arms with the dusty interarm regions. It is this part of the galaxy that lies closest to us and the part that is not obscured by the outflow of material along the minor axis, whereas to the north and east the spiral arm is obscured and what we see is the actual dusty filaments of the outflow.

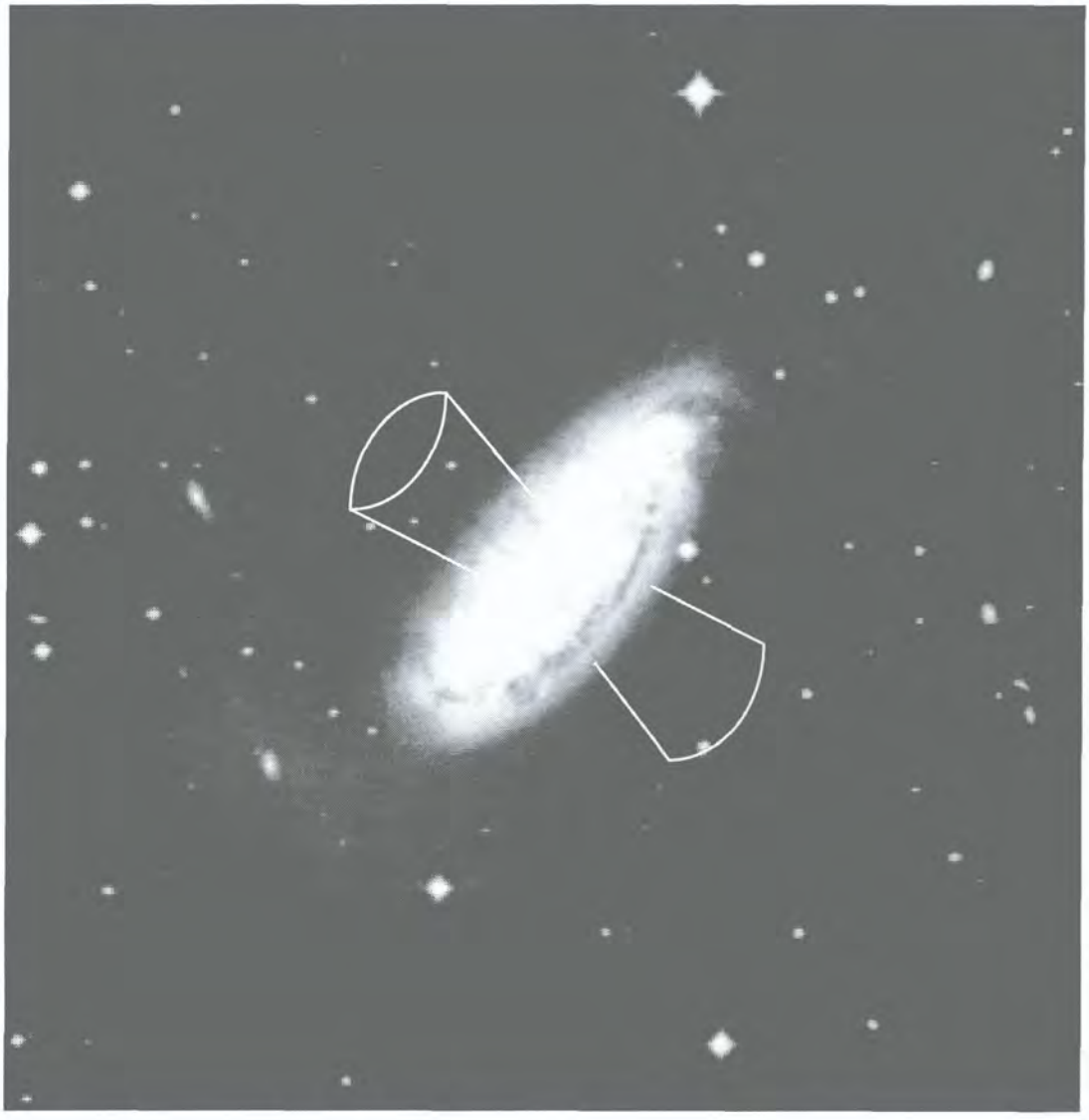


Figure 6-1- NGC 1808 with outflow overlay

This image was obtained from the Digitized Sky Survey.

The Digitized Sky Surveys were produced at the Space Telescope Science Institute under U.S. Government grant NAG W-2166. The images of these surveys are based on photographic data obtained using the Oschin Schmidt Telescope on Palomar Mountain and the UK Schmidt Telescope. The plates were processed into the present compressed digital form with the permission of these institutions.

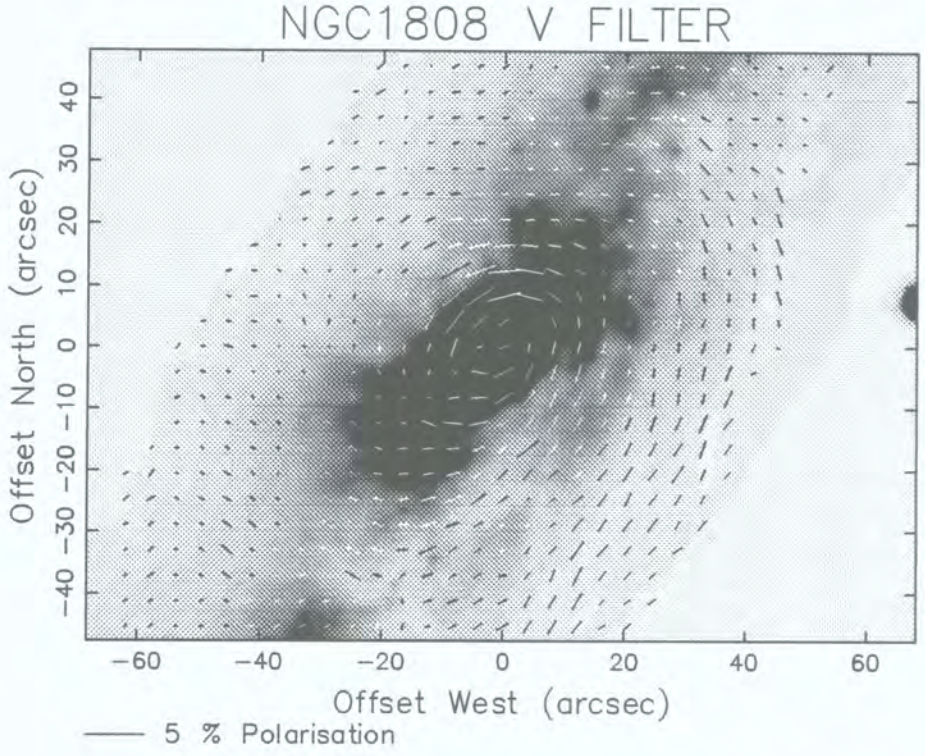


Figure 6.2 - NGC 1808

Large-scale grey-scale image overlaid with a polarization map.

The polarization map traces out the spiral arm of the galaxy at a distance > 20 arcsec from the centre of the galaxy in a south-westerly direction. The vectors are running parallel to the arms/interarm area and the author believes that this is due to dichroic extinction by paramagnetic grains in the magnetic field of the spiral arms of the galaxy. This type of feature is often seen in spiral galaxies that are viewed with small inclination (i.e. viewed near to face on) such as M51 and NGC 1068 (Scarrott, Ward-Thompson and Warren-Smith 1987, Scarrott, Rolph and Semple 1989). It is interesting to note that radio observations at λ = 6cm by Dahlem *et al.* (1990) show a large-scale magnetic field running through the spiral arms of the galaxy. In the western regions of the galaxy, the radio data and optical data overlap and show the polarization vectors to be perpendicular to each other, the expected result if a magnetic field was responsible for the effects. The reason that the vectors are perpendicular to one another is that the optical vectors are due to dichroic extinction where the vectors run parallel to the field, whereas the radio vectors are due to synchrotron radiation where the *E*-vector is perpendicular to the field.

Dahlem *et al.* (1990) were undecided about the nature of the magnetic field, whether it was toroidal and restricted to the disk or possible poloidal and bound to the halo. The optical data favours the idea that the field is bound to the disk and spiral arms as the levels of polarization are higher in the interarm regions. If the halo of the galaxy was responsible for the magnetic field polarization pattern then we would expect to see evidence of it in the north and east regions of the galaxy, those areas dominated by the outflow into the halo. In the north-east region the vectors do not follow the spiral pattern and are indeed quite small indicating that the dust and polarized light emission from this region is obscured from view due to being part of the spiral arm / interarm structure. It is for these reasons that the author believes what is seen here, on the large scale, is the effect of a bound magnetic field in the spiral arms of the galaxy.

The inner regions of the galaxy are shown in Figure 6-3 in the same format as Figure 6-2 but at a higher spatial resolution.

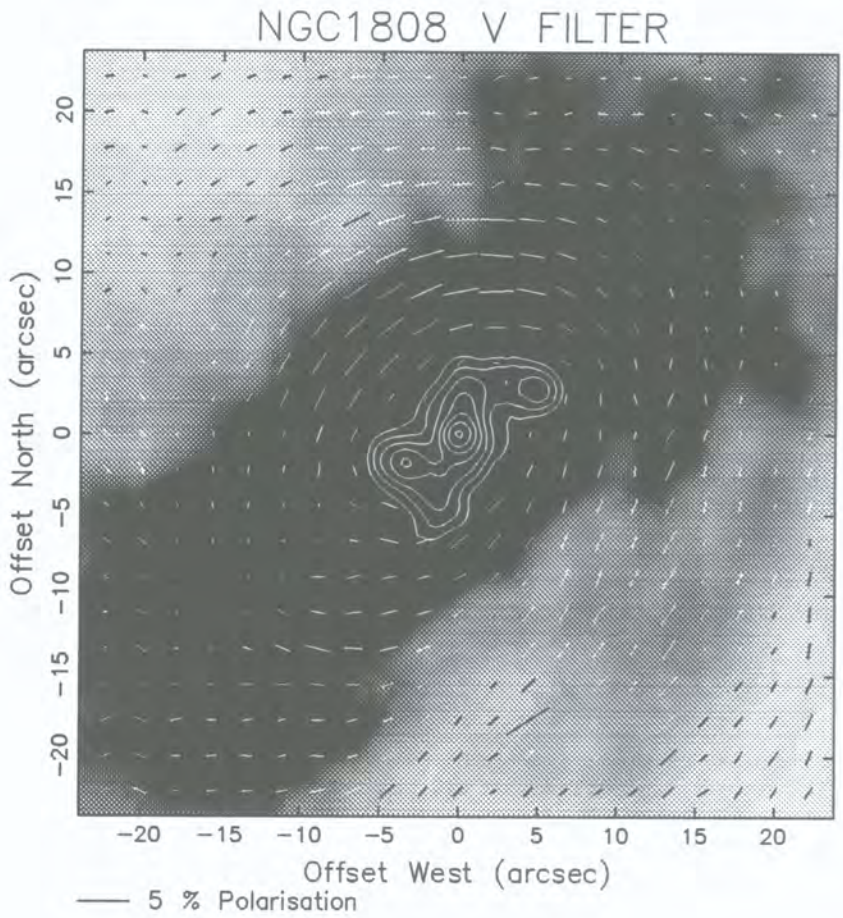


Figure 6.3 - NGC 1808 (Inner regions)

As Figure 6.2 but showing only the inner 25 arcsec.

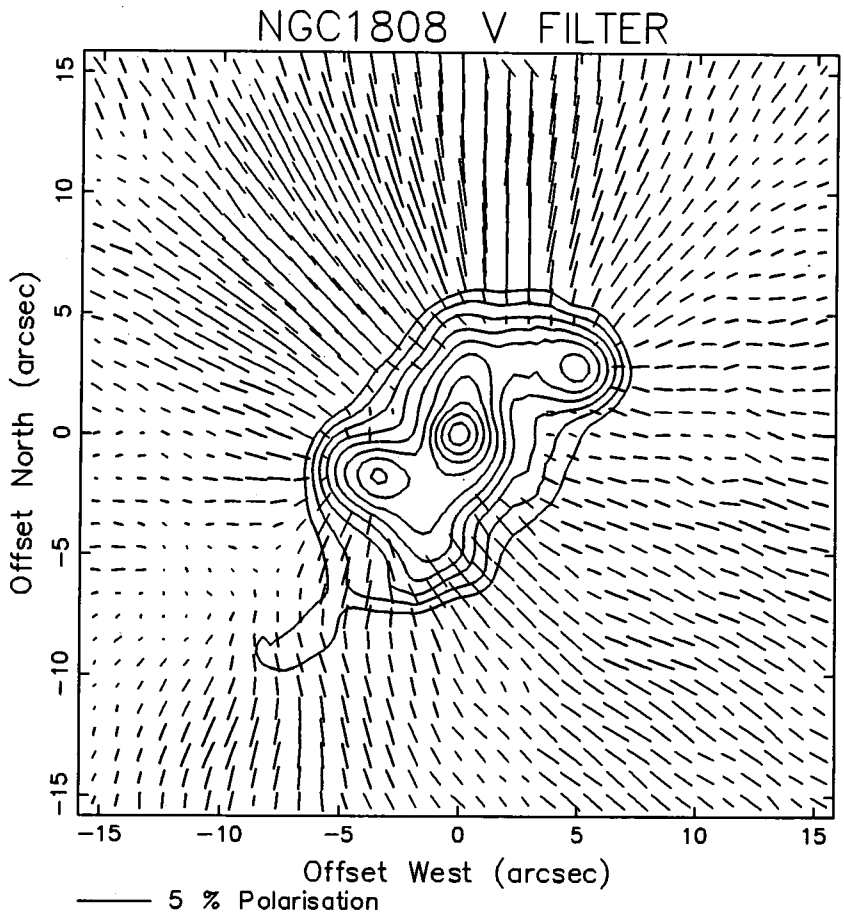


Figure 6.4 - NGC 1808 (Illuminating Sources)

The central region of NGC 1808 with the polarization vectors drawn perpendicular to the *E*-vector of the radiation.

The central optical bright spot within the inner region is assumed to be the nucleus (Saikia *et* al. 1990) and it can be seen that the polarization vectors follow an elliptical pattern within ~15 arcseconds of this point. This type of pattern is typically seen in reflection nebula that have extended illuminating sources and we can assume that in the case of the galaxy NGC 1808 the dusty halo is acting like a giant reflection nebula being illuminated by the starburst regions of the galaxy. The region towards the north-eastern quadrant shows higher levels of polarization and a more uniformly circular pattern, it is in this region that the outflow into the halo is most observable due to the orientation of the galaxy. In the region diagonally opposite i.e. the south-western quadrant, the vectors are significantly smaller and more random as they interact with the effect of the magnetic field that lies forward of the outflow. This is a similar effect to that seen in the archetypal starburst galaxy M82 (Bingham *et al.* 1976; Chesterman & Pallister 1980) but in the case of NGC 1808 the levels of polarization are smaller by a factor of between 2 to 3. The reason for this is probably due the inclination of the galaxy, M82 is seen almost in the edge-on position whereas NGC 1808 is more highly inclined to the face-on position and with it the line of sight takes in both the polarized halo and the unpolarized galactic disk that acts to dilute the effects.

To detect the illuminating sources within the central region of the galaxy, Figure 6-4 has the polarization vectors drawn perpendicular to the *E*-vector of the radiation. By doing this it should be possible to detect the illuminating source(s) as the vectors will all radiate from this point. It can be seen from the figure that all the bright 'hot spots' contribute to the illumination of the halo in one way or another, with the biggest effect coming from the nucleus. The nucleus can clearly be seen to strongly illuminate the north-eastern part of the halo and has to be considered to be the major component of the system. The bright spot to the west of the nucleus, probably the next brightest source, illuminates the north-west region but with the addition that the nucleus also plays some part. The bright spot immediately to the south and east of the nucleus is seen as illuminating the halo directly to the east and the faintest of the illuminators

Table 6-2 - Polarization of NGC 1808 Central Region

Knot	″W	″N	P(%)	θ(°)
nucleus	0.0	0.0	1.2 ±0.3	117 ±6
NW of nucleus	4.7	2.8	0.4 ±0.3	151 ±18
SE of nucleus	-3.3	-1.8	1.4 ±0.3	93 ±5
SE of nucleus	-6.5	-8.6	1.0 ±0.2	104 ±5

further down to the south and east illuminates the halo due south of itself. An interesting feature is seen in the south-east quadrant of the map, where the effects of the latter two illuminators cancel out and only a very small polarization pattern is produced.

By carrying out aperture polarimetry (using simulated software apertures) on the bright spots within the central region, it is possible to determine the intrinsic polarization of the sources. The results are shown in Table 6-2.

The first two results are a good approximation to the polarization of the bright spots but the latter two are unreliable due to the effect of either other illuminators or background polarization. Forbes *et al.* (1992) found that the nucleus was highly reddened whereas the bright spot north-west of the nucleus was the least reddened in the system. The reddening of the bright spots is a good indication to the obscuring dust and from this can be drawn the conclusion that there must be big differences in the obscuration even in the local area of the starburst and that possibly the nucleus is intrinsically polarized. This could be due to the idea that the nucleus is in fact a Seyfert (Morgan 1958, Véron-Cetty and Véron 1985) which are known to have circumnuclear torii, producing the polarized emission.

6.2.4 Conclusion

NGC 1808 has proved to be an interesting candidate for polarmetric study and has revealed interesting facts about the nature of both the large and small scale structure.

On a large scale we see polarization patterns mapping the magnetic field that is bound into the spiral arms and the interarm areas, whereas on the smaller scale we see interesting features associated with the nucleus, providing more evidence to support the idea that it is in fact a Seyfert. The polarization map of the inner area also tells us of the effects that each illuminator has on the halo and the outflow, which has the appearance of a galactic scale reflection nebula.

6.3 NGC 3256

6.3.1 Introduction

NGC 3256 is a southern hemisphere merger galaxy (see Figure 6-5) located at a distance of 37 Mpc assuming $H_0 = 75$ kms^{-1} Mpc^{-1} (Casoli, Dupraz and Combes 1992). Many have categorised the galaxy as being near to merger-completion and although this point is debatable it is known that the galaxy is a product of two equal-mass galaxies (Toomre & Toomre 1972) and is seen in the visual waveband as having two tidal tails. The collision has brought to the galaxy an estimated ~10^{10} *Mo* of molecular material that is observed as extending beyond the nuclear region (Sargent, Sanders & Phillips 1989) and is probably the fuel for the starburst (Doyon, Joseph & Wright 1994) occurring around the galactic centre. The development timescale of the merger (Schweizer 1986) is estimated to being between 10^8 and 10^9 years although it is thought that the starburst only began some 1 - 3 x 10^7 years ago (Doyon *et al.* 1994). The effect of this starburst with all the dust is to give the galaxy an infrared luminosity of ~ 5 x 10^{11} *Lo* which by some definitions would put it in the ultraluminous class and it is thought that the starburst may be powerful enough to drive a superwind from the nuclear region (Heckman, Armus & Miley 1990).

Radio observations at λ = 6cm by Smith and Kassim (1993) found four intensity maxima of which the brightest coincides with both the optical and near-infrared data. The near-infrared data shows that this cospatial peak is probably the nucleus and is emitting radiation at 2μm. Observations in the near-infrared by Zenner and Lenzen (1993) found multiple structures in the starburst region one of which they tentatively proposed was the second nucleus, presumably one from each of the merging systems. Recent high resolution (1 arcsec) radio observations at λ = 3cm & 6cm using the

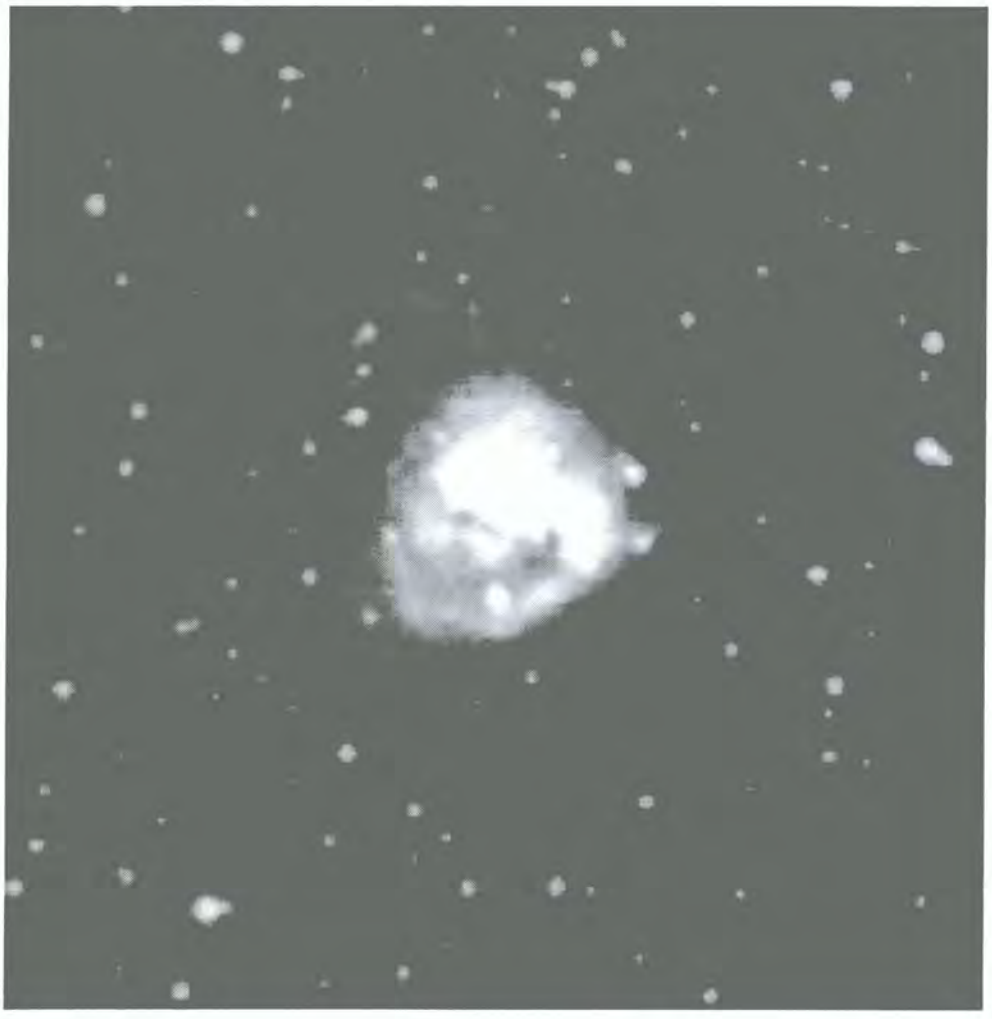


Figure 6-5 - NGC 3256

This image was obtained from the Digitized Sky Survey.

Australia Telescope Compact Array (Norris & Forbes 1995) has revealed a double nucleus, and has made astronomers reconsider the current status of the merger. It is accepted that there are strong correlation's between tidally interacting/merging galaxies, starbursts and powerful infrared luminosities, but quite how they all fully associate is still unclear. NGC 3256 falls into three of the above criteria and using optical polarization imaging it is hoped that more can be discovered as to the evolution of this system. Energy released during starburst periods often can lead to the generation of superwinds that carry material out into the halo, and this is something that hopefully imaging polarimetry can confirm.

6.3.2 Observational Details

NGC 3256 was observed in January 1994 with the Anglo-Australian Telescope, using the Mark IV Durham Imaging Polarimeter (see also Scarrott *et* al. 1983)and the EEV P8603 CCD blue-coated detector. The seeing conditions were approximately 1.5 arcsec and a sequence of 12 exposures of a duration 600 seconds were taken with a V-waveband filter in the beam. The 12 exposures gave full spatial coverage of the galaxy. The data were reduced in the standard manner (Draper 1988).

6.3.3 Discussion

A large-scale polarization map superimposed on top of a greyscale image is presented in Figure 6-7 and shows a coherent polarization pattern in the northern quadrants with levels of up to approximately 4%. The pattern is reasonably centrosymetric about the central 8 arcseconds of the galaxy and has the appearance of a reflection nebula of galactic proportions although at this resolution it is not possible to distinguish the position of the illuminating source or sources and therefore it is necessary to analyse specifically the nuclear region.

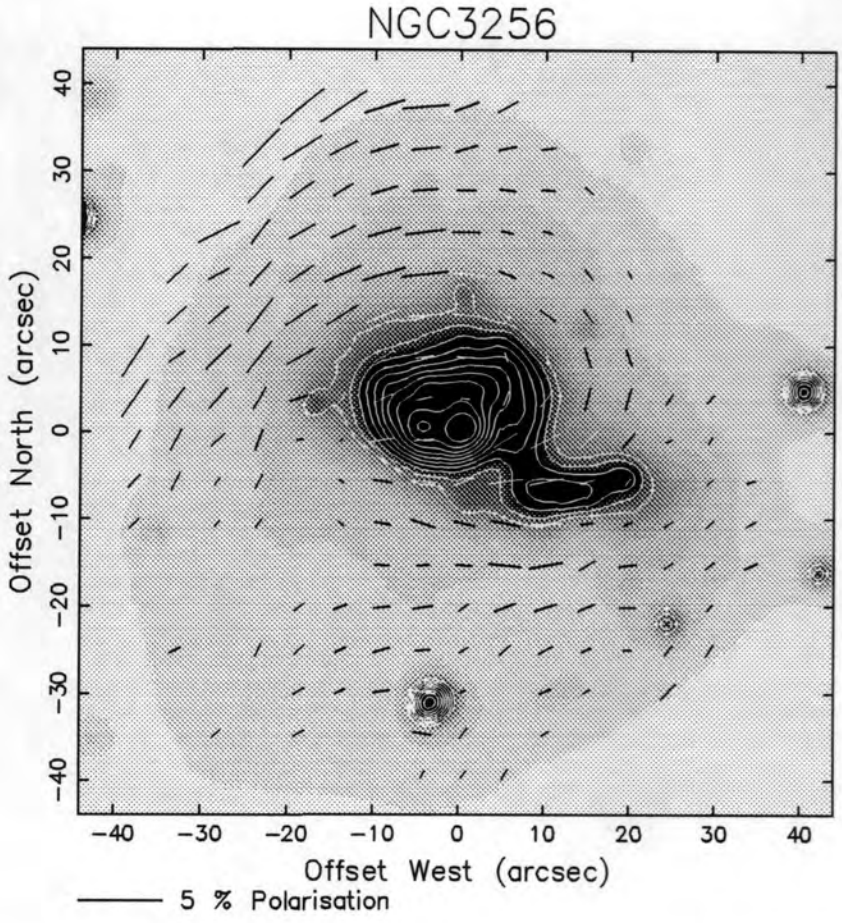


Figure 6.6 - NGC 3256

A large-scale grey-scale image of NGC 3256 with an overlaid polarization map. The inner regions are shown with intensity contours.

In this region the polarization mechanism is probably one of scattering where dust from post-main-sequence massive stars that have exploded as supernovae is carried out into the halo of the galaxy by a superwind. To detect the presence of a polarized halo it is necessary to have dust and anisotropic illumination, and so it may be assumed, as there is a polarized halo that these are true and that the illuminator of the halo is far brighter than the rest of the galactic illumination. To try and determine the exact source of the illumination it is necessary to analyse the nuclear region in more detail as mentioned previously. Figure 6-7 is a greyscale image showing the inner most 30 arcseconds of the object, overlaid with a polarization map that has the vectors turned through 90° so that they are running perpendicular to the *E*-vector of the polarized light, and therefore form a radial pattern indication the illuminating source or sources.

It can be seen that the pattern in the north and the north-east show the effect of two illuminators, the nucleus and the source directly to the east of the nucleus. The area to the west-north-west between 10 and 15 arcseconds from the nucleus appears to be illuminated by a completely different source that is obscured from our view. The source of the illumination would seem to be approximately 5 arcseconds to the north-west of the nucleus and the fact that it is not detected visually could be due to heavy obscuration from localised dusty regions.

It can be seen from Figure 6-7 that the galaxy is illuminated in the northern direction to > 7Kpc which is far larger than the illumination distance found in other starburst galaxies such as M82, the archetypal starburst with a polarized halo. The polarization levels in NGC 3256 are lower than in M82 although this could have something to do with the fact that NGC 3256 is more disturbed due to the merger, and also that we view it from a different angle. The view of M82 that the observer is afforded is toward edge-on and therefore the light from its polarizing halo will not be diluted by unpolarized starlight from the disk, a situation that we see in NGC 3256. It is widely believed that when a starburst begins, the UV radiation propagates out towards the halo at the speed of light and photoionises the halo within timescales of 10^4 - 10^5 years.

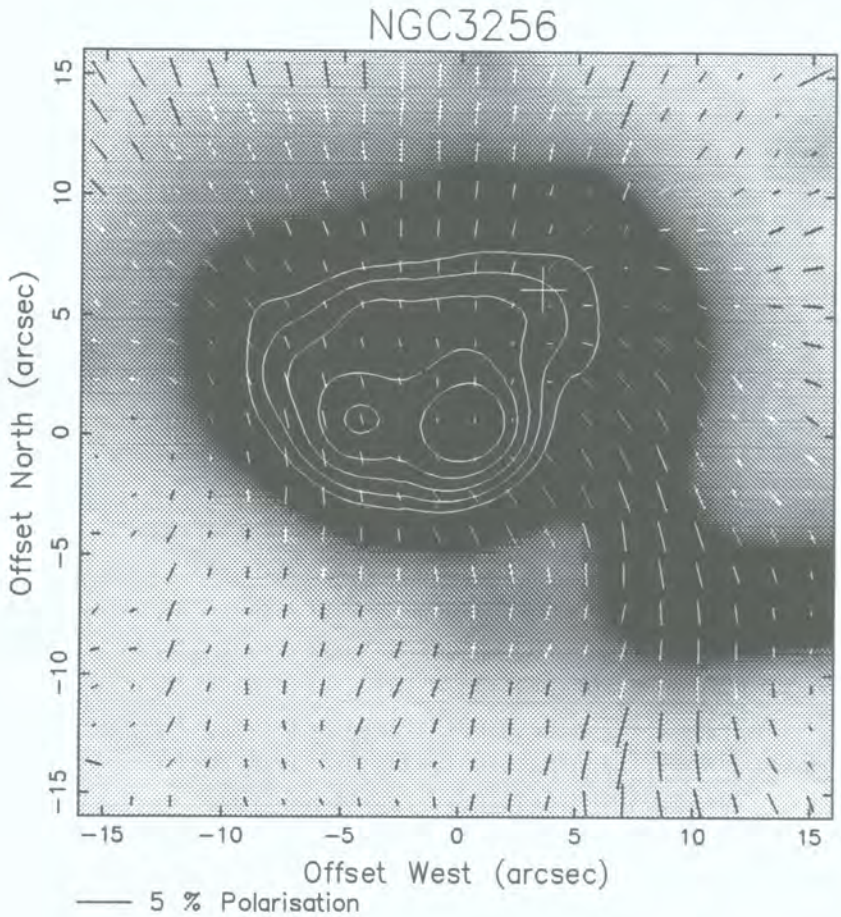


Figure 6.7 - NGC 3256 (Inner region)

The central region of NGC 3256 similar to Figure 6.6 with the polarization vectors drawn perpendicular to the *E*-vector of the radiation.

The superwind however depends on the production of massive OB type stars and will travel out at speeds of approximately 1000 kms^{-1} and so the effects will not be seen until timescales of 10^6 - 10^7 years. In the area immediately to the south of the system centre and running in an east-west direction, the polarization vectors are aligned parallel to each other (see Figure 6-6) which I have assumed correspond to a bar like structure and the possible transport mechanism of gas into the starburst regions. This pattern has all the hallmarks of dichroic extinction due to paramagnetic grains aligned in a magnetic field. If this is the case, then we need to address the issue of a magnetic field within a galactic-merger system. To the south-east of the map, the centro-symmetric pattern that dominates the northern quadrants deviates in direction and there is an area of virtually no polarization (< 1%), this is the expected result if there are indeed two polarization mechanism at work, as the two competing mechanisms effectively cancel out.

6.3.4 Conclusions

The galactic merger NGC 3256 shows a strong polarization pattern in the northern quadrants due probably to scattering from the dusty halo and it is quite possible that we view the galaxy with this area tilted toward us and therefore the polarized light is less diluted. It would seem as if there are multiple illuminators at work, one of which is totally obscured at the visual wavelengths, but it is impossible to conclude if one of them is the second nucleus from the merger.

The region due south of the nuclear region shows a parallel polarization pattern that I have attributed to dichroic extinction due to a magnetic field in a bar. The subject of magnetic fields in merging galaxies is extremely unclear and therefore all I wish to say is that there appears to be a field in this galaxy. Whether it is a product of the merger or was an original field in one of the galaxies and has survived the merger, I offer no opinion.

6.4 NGC 2146

6.4.1 Introduction

NGC 2146 is a well known and studied edge-on starburst galaxy that shows conspicuous dust lanes running through the centre of its structure (see Figure 6-8). Sandage and Tammann (1987) classified the galaxy as SbIIpec, and assuming H_o = 75 kms^{-1} Mpc^{-1} puts it at a distance of 14.5 Mpc which gives an image scale of approximately 70 pc $arcsec^{-1}$. The galaxy is in many ways a typical far-infrared (FIRG) / starburst galaxy, it has a luminosity of L_{FIR} = 6.3 x 10^{10} *Lo* and is thought to contain large amounts of molecular hydrogen gas, approximately 6 x 10^9 *Mo*, an ideal breeding ground for young massive stars that will drive a possible superwind. There is much evidence for the idea that a superwind is operating in NGC 2146, in both the optical and the X-ray parts of the spectrum. Armus, Heckman and Miley (1990) made observations of many FIRGs including NGC 2146 and found that it emitted radiation in the Hα-waveband. The emission has a filament-like appearance and is one of the expected signs of a superwind as it pushes out the shell of gas. Data from ROSAT presented by Armus, Heckman, Weaver and Lehnert (1995) shows a large and conspicuous X-ray halo that is cospatial with the emission-line flux along the minor galactic axis, suggesting that a superwind does indeed occur in this galaxy.

There is within the galaxy a strong radio source (4C78.06) which Kronberg and Biermann (1981) found did not correspond to the bright optical knot, but was visibly obscured somewhere within the dust lanes. Infrared data (Hutchings *et al.* 1990) in the K band shows an intensity peak that corresponds to the radio data and therefore we can assume that this is the nucleus buried deep within the dust.

Using imaging polarimetry, we should hopefully be able to see the effect of the superwind (as in NGG 1808 & NGC 3256) as it carries dusty material out towards the halo, by observing scattered light originating from a source within the nuclear region.

Figure 6-8 - NGC 2146

This image was obtained from the Digitized Sky Survey.

The Digitized Sky Surveys were produced at the Space Telescope Science Institute under U.S. Government grant NAG W-2166. The images of these surveys are based on photographic data obtained using the Oschin Schmidt Telescope on Palomar Mountain and the UK Schmidt Telescope. The plates were processed into the present compressed digital form with the permission of these institutions.

The fact that NGC 2146 also has a bisecting dust lane means that there may be some form of magnetic field involve and hopefully this will also be confirmed using the polarization data.

6.4.2 Observational Details

NGC 2146 was observed in January 1995 with the JKT, using the Mark IV Durham Imaging Polarimeter (see also Scarrott *et* al. 1983) and the EEV P86000 CCD AstroChrome™ coated detector. The seeing conditions were approximately 1.5 arcsec and a sequence of 16 exposures of a duration 800 seconds were taken with a V-waveband filter in the beam. The data were reduced in the standard manner (Draper 1988).

6.4.3 Discussion

Figure 6-9 shows a large-scale polarization map superimposed onto a greyscale image in the V-waveband and from it can be seen that the polarization pattern shows that the galaxy maybe somewhat chaotic. The two main features of this polarization map are firstly the vectors to the north-east and south-west that appear to form a centro-symetric pattern that is centred somewhere in the dust lane, and secondly, the dust lane itself. Polarization levels reach an approximate maximum of just less than 5% and it would appear that the centro-symmetric pattern is similar to that found in reflection nebulae, but on a galactic scale.

The galactic-scale reflection nebula is seen in the regions away from the prominent dust lane running through the galaxy and can be attributed to light being scattered by the dust in the galactic halo. It is thought that the dust in the halo is carried from the nuclear starforming regions entrained in the superwind. The pattern extends throughout the inner ±50 arcseconds and can be seen to be elliptical in manner, a point that shall be returned to latter in the text.

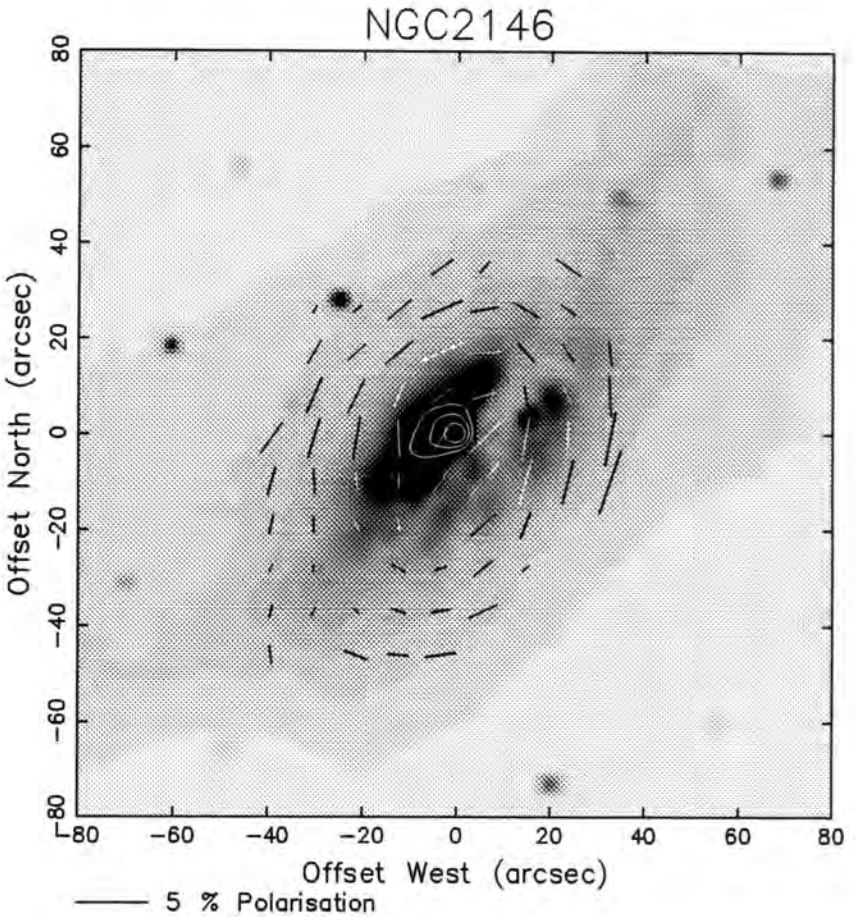


Figure 6.9 - NGC 2146

Large-scale grey-scale image overlaid with a polarization map. Inner region is shown with intensity contours.

If we assume that we have measured the extent of the reflection nebula feature, then it would appear to extend out approximately 2.5 Kpc along the minor axis and would therefore be cospatial with both the emission-line data (Armus *et al.* 1990) and X-ray data (Armus *et al.* 1995) both of which can be closely associated with superwinds as the gas is blown from the central region and shocks the cooler ISM.

The pattern in the north-eastern quadrant of the polarization map (Figure 6-10) shows a more symmetrical pattern than the corresponding one in the south-westerly quadrant. This could be due to the fact that the northern regions of the galaxy maybe tilted toward us and that we are seeing the nebula without the effects of the central dust lane. Using the V-waveband data along with the other colours, it can be determined that the illuminating source extends no more than 5 arcseconds and is located in the highly obscured part of the galaxy. The position of the source is shown in Figure 6-10 and corresponds within errors, to both the radio and infrared data (Kronberg & Biermann 1981).

The second major feature that we can see in this galaxy is the central dust lane that bisects the galaxy, and the polarization pattern that runs through it. In Figure 6-10 the inner regions of the galaxy are mapped and laid on top of an uncalibrated V - I waveband image with I-waveband intensity contours. From the map it is easy to see the parallel polarization vectors that run along the dust lane. The V - I image allows us to see the most reddened, and therefore the most dusty parts of the lane and from it we can see structure extending out to the south of the galaxy. This polarization pattern can be explained by the idea of a magnetic field running through the dust lane and aligning paramagnetic grains that act in a dichroic manner.

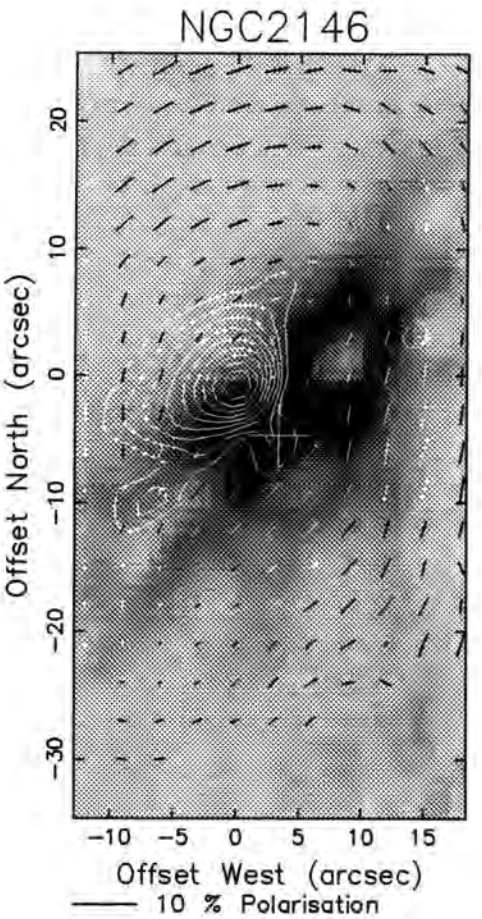


Figure 6.10 - NGC 2146

A V - I high spatial resolution image overlaid with a polarization map and intensity contours from the I waveband.

It can be seen that the levels of polarization decrease to almost zero at points ±15 arcseconds along the major axis of the galaxy as defined by the dust lane. It is at these points that the two competing polarization mechanisms have the same quantitative effect and null points are observed. The elliptical polarization pattern, as mentioned earlier, can also be attributed to the effect of the competing scattering and dichroism mechanisms.

6.4.4 Conclusion

The starburst driven superwind of NGC 2146 drives material into the galactic halo which is then illuminated by the newly formed massive stars. The light scattered from the dusty halo is by virtue of the scattering, polarized and leads to a polarization pattern that is reminiscent of a reflection nebula, though on a galactic scale.

The conspicuous dust lane that bisects the galaxy is similar in appearance to the non-starburst galaxies M104 and NGC 5128 that are viewed in the same orientation as NGC 2146, and posses magnetic fields that are detectable due to magnetically aligned grains. It would appear that NGC 2146 also has a magnetic field that has survived the event that triggered the starburst phase in the galaxy.

7. Conclusions

The Mark IV Durham polarimeter has followed in the path of three earlier and successful instruments and has advanced astronomical polarimetry to a new level of sophistication. The current instrument is used on a regular basis and since the commissioning run on the Jacobus Kapteyn Telescope (JKT) in November 1992 it has been used on twelve separate observing trips to date.

There are various aspects of the system that could be further developed and hopefully these improvements will be implemented in the future. These changes involve the optics of the instrument, the hardware that runs both the polarimeter and the CCD camera, and the software that controls both the instrument and camera. This final chapter will briefly summarise the current system and then talk about possible future developments.

7.1 Hardware

The polarimeter is controlled via the RS232C ports of a personal computer (PC) that runs a copy of the UNIX operating system. The PC also controls the CCD camera via a generic interface card supplied with the system and records the data onto various backup devices such as magneto-optical disks. The stepper motors that drive the various functional units (half wave plate etc.) within the polarimeter are powered by commercial stepper motor drivers and controlled via intelligent controller cards that accept simple command strings to make the motors move. The electronics and computing hardware are extremely reliable with only one problem during all the observing runs to-date. The problem occurred on the commissioning run on the JKT in

November 1992, when the night time temperature fell to within a few degrees of freezing causing the sensor assembly on the half wave plate to fail. The assembly was not actually faulty but its response time had decreased and therefore would not operate at a speed matching that of the half wave plate's rotation. The solution was simply to edit the configure file to reduce the speed of rotation of the half wave plate so that there was ample time for the sensors to respond.

There are a number of improvements that could be made to the system though none of them that would alter its functionality. The main areas for improvement are size and weight, and moving to more sophisticated computing hardware that would give better facilities at the observing site. The electronic control system for the instrument could be reduced in size and weight which would reduce shipping costs, make it easier to handle by the observers, and possibly improve balancing on the smaller telescopes such as the JKT. The main problem with this idea is the time and resources needed to improve on something that already works well. The second area for improvement is on the computer hardware front which cannot be left unmodified for fear of it becoming too outdated. There is also the need to provide the observer with a data reduction and analysis facility so that as little time as possible is wasted on objects that show no useful results. The plan of development for this is to use a Sun workstation to control both the camera and polarimeter, as well as provide the analysis system for the observer.

7.2 Software

The software, like the hardware, is dependable and works well. The programs are all written in the 'C' programming language or 'C' shell scripts and are therefore reasonably portable as discussed in Chapter 3 and have indeed been ported to other platforms and operating systems already. The major improvement within this area of the project would be the introduction of a graphical user interface (GUI) that would

allow the polarimeter to be controlled in a 'point-and-click' manner. This will eventually allow for both the camera and instrument control to be carried out using a dedicated front-end that is specifically designed for the needs of the application and will therefore improve efficiency during observing when time is critical.

The second improvement that will be made to the software is the automatic inclusion of the instrument parameters into the data files. When the camera and instruments control software is fully integrated, it will be possible to include the half wave plate position, the selected filter and the exposure with each data frame as it is recorded. This will avoid the possibility of an error occurring at the analysis stage due to the person who is reducing the data entering incorrect information.

7.3 Optics

The principles of the polarimeter go back to the last century and as far as can be seen could not be improved upon. The only improvements that could be made to the optics of the Durham polarimeter are either make them bigger or measure something slightly different. The main idea behind making the optics larger is to increase the area covered within each field which on the larger objects will save observing time as there will not be a need to make multiple mosaics of exposures. To achieve this improvement, the half wave plate, both the field and relay lenses and the CCD detector would all need to be replaced although the existing Wollaston prism could be retained as it is large enough for an increased beam width. The second possible development in the optics of the polarimeter is the idea of adding a quarter wave plate so as be able to measure circularly polarised light as well as linearly polarized light. Circularly polarized light can be produced by multiple scattering and is measured by a quarter wave plate rotating and converting the light to a linearly polarized state that can be measured in the standard fashion. The main problems experienced when measuring circular polarized light are due to the low levels of polarization which would give low signal-to-noise levels etc. and make accurate results difficult to obtain.

References

Armus, L., Heckman, T.M. & Miley, G.K., 1989. *Ap. J.*, **347**, 727.

Armus, L., Heckman, T.M. & Miley, G.K., 1990. *Ap. J.*, **364**, 471.

Armus, L., Heckman, T.M., Weaver, K.A. & Lenhert, M.D., 1995. *Ap. J.*, **445**, 666.

Arp, H. & Bertola, F., 1970. *Ap. Lett.*, **6**, 65.

Axon, D.J., 1977. *PhD thesis*, University of Durham.

Bingham, R. G., McMullen, D., Pallister, W.S., White, C., Axon, D.J. & Scarrott, S.M., 1976. *Nature*, **259**, 463.

Broadhurst, T. & Lehar, J., 1995. *Ap. J.*, **450**, L41.

Brown, R.L. & Vanden Bout, P.A., 1991. *Ap. J.*, **102**, 1956.

Burbidge, E.M. & Burbidge, G.R., 1968. *Ap. J.*, **151**, 99.

Casoli, F., Dupraz, C. & Combes, F., 1992. *A. & A.*, **264**, 49.

Chesterman, J.F. & Pallister, W.S., 1980. *M.N.R.A.S.*, **191**, 349.

Chevalier, R.A. & Clegg, A.W., 1985. *Nature*, **317**, 44.

Condon, J.J., Condon, M.A., Gisler, G. & Puschell, J.J., 1982. *Ap. J.*, **252**, 102.

Danks, A.C., Perez, M.R. & Altner, B., 1990. *"Bulges of Galaxies"*, ed. B.J. Jarvis & D.M. Terndrup (München:ESO), p.243.

Dahlem, M., Aalto, S., Kelin, U., Booth, R., Meobold, U., Wielebinski, R. & Lesch, H., 1990. *A. & A.*, **240**, 237.

Davis, L. & Greenstein, J.L., 1951. *Ap. J.*, **114**, 206.

Draine, B.T. & Lee, H.M., 1984. *Ap. J.*, **285**, 89.

Draper, P.W., 1988. *PhD thesis*, University of Durham.

Draper, P.W., Done, C., Scarrott, S.M. & Stockdale, D.P., 1995. *M.N.R.A.S.*, **277**, 1430.

Doyon, R., Joseph, R.D. & Wright, G.S., 1994. *Ap. J.*, **421**, 101.

Elsasser, H. & Staude, H.J., 1978. *A. & A.*, **70**, L3.

Field, G.B., Goldsmith, D.W. & Habing, H.J., 1969. *Ap. J.*, **155**, L149.

Forbes, D.A., Boisson, C. & Ward, M.J., 1992. *M.N.R.A.S.*, **259**, 293.

Gething, M.R., Warren-Smith, R.F., Scarrott, S.M. & Bingham, R.G., 1982. *M.N.R.A.S.*, **189**, 881.

Hernquist, L., 1989. *Nature*, **340**, 687.

Hsu, J. & Breger, M., 1882. *Ap. J.*, **262**, 732.

Hutchings, J.B., Neff, S.G., Stanford, S.A., Lo, E. & Unger, S.W., 1990. *A.J.*, **100**, 60.

Junkes, N., Pietsch, W. & Hensler, G., 1992. *COSPAR Symposium, "Advances in Space Research (Pergamon Press)"*, p154.

Koribalski, B., 1993. *PhD thesis*, University of Bonn.

Kronberg, P.P. & Biermann, P., 1981. *Ap. J.*, **243**, 89.

Larson, R.B. & Tinsley, B.M., 1978. *Ap. J.*, **219**, 46.

Menard, F., Bastien, P. & Roberts, C., 1988. *Ap.J.*, **335**, 290.

Mihos, J.C. & Hernquist, L., 1994. *Ap. J.*, **431**, L9.

Morgan, W.W., 1958. *P.A.S.P.*, **70**, 374.

Noguchi, M., 1991. *M.N.R.A.S.*, **251**, 360.

Norris, R.P. & Forbes, D.A., 1995. *Ap. J.*, **446**, 594.

Norman, C.A. & Ikeuchi, S., 1989. *Ap. J.*, **345**, 372.

Notni, P., 1985 *Astr. Nachr.*, **306**, 265.

Ohman, Y., 1939. *M.N.R.A.S.*, **99**, 624.

Pallister, W.S., 1976. *PhD thesis*, University of Durham.

Pickering, 1873. *American Academy of Arts and Science.*, **IX**, 1.

Pildis, R.A., Bergman, J.N. & Schombert, J.M., 1994. *Ap.J.*, **423**, 190.

Rieke, G.H., Cutri, R.M., Black, J.H., Kailey, W.F., Mcalary, C.W., Lebofsky, M.J. & Elston, R., 1985. *Ap.J.*, **290**, 116.

Reif, K., Meobold, U., Goss, W.M., van Woerden, H. & Siegman, B., 1982. *A. & A.S.*, **256**, 10.

Rieke, G.H., Lebofsky, M.J., Thompson, R.I., Low, F.J. & Tokunaga, A.T., 1980. *Ap. J.*, **238**, 24.

Rolph, C.D. 1990. *PhD thesis*, University of Durham.

Rowan-Robinson, M., Broadhurst, T., Oliver, S. J., Taylor, A. N., Lawrence, A., Mcmahon, R. G., Lonsdale, C. J., Hacking, P. B. & Conrow, T., 1991. *Nature*, **351**, 719.

Saikia, D.J., Pedlar, A., Unger, S.W. & Axon, D.J., 1994. *M.N.R.A.S.*, **270**, 46.

Saikia, D.J., Unger, S.W., Pedlar, A., Yates, G.J., Axon, D.J., Wolstencroft, R.D., Taylor, K. & Glydenkerne, K., 1990. *M.N.R.A.S.*, **245**, 397.

Sandage, A.R. & Tammann, G.A., 1987. *"A Revised Shapley-Ames Catalog of Bright Galaxies"*, (2nd ed.; Carnegie Inst. Washington Pub., Vol 635), [RSA].

Sanders, D.B., Scoville, N.Z. & Soifer, B.T., 1991. *Ap. J.*, **370**, 158.

Sargent, A.I., Sanders, D.B. & Phillips, T.G., 1989. *Ap. J.*, **346**, L9.

Scarrott, S.M., Draper, P.W., Stockdale, D.P., 1996. *M.N.R.A.S.*, **279**, 1325.

Scarrott, S.M., Draper, P.W., Stockdale, D.P. & Wolstencroft, R.D., 1993. *M.N.R.A.S.*, **264**, L7.

Scarrott, S.M., Draper, P.W., Warren-Smith, R.F., 1989. *M.N.R.A.S.*, **237**, 621.

Scarrott, S.M., Rolph, C.D. & Semple, D.P., 1989. *I.A.U.*, **140**, 245.

Scarrott, S.M., Ward-Thompson, D. & Warren-Smith, R.F., 1987. *M.N.R.A.S.*, **224**, 299.

Scarrott, S.M., Warren-Smith, R.F., Pallister, W.S., Axon, D.J. & Bingham, R.G., 1983. *M.N.R.A.S.*, **204**, 1163.

Schweizer, F., 1986. *Sci.*, **231**, 227.

Sérsic, J.L. & Pastoriza, M., 1965. *Pubs. astr. Soc. Pacif.*, **77**, 287.

Shapiro, P.R. & Field, G.B., 1976. *Ap. J.*, **205**, 762.

Smith, E.P. & Kassim, N.E., 1993. *A.J.*, **105**, 46.

Soifer, B. T., Neugebauer, G., Rowan-Robinson, M., Clegg, P. E., Emerson, J. P., Houck, J. R., De Jong, T., Aumann, H. H., Beichman, C. A. & Boggess, N., 1984. *Ap. J.*, **278**, L71.

Soifer, B.T., Sanders, D.B., Madore, B.F., Neugebauer, G., Danielson, G.E., Elias, J.H., Lonsdale, C.J. & Rice, W.L., 1987. *Ap.J.*, **320**, 238.

Suchcov, A.A., Balsara, D.S., Heckman, T.M. & Leitherer, C., 1994. *Ap. J.*, **430**, 511.

Telesco, C.M., 1988. *A. R. A. & A.*, **26**, 343.

Telesco, C.M. & Harper, D.A., 1980. *Ap. J.*, **235**, 392.

Toomre, A. & Toomre, J., 1972. *Ap. J.*, **178**, 623.

Véron-Cetty, M.-P. & Véron, P., 1985. *A. & A.*, **145**, 425.

Warren-Smith, R.F., 1979. *PhD thesis*, University of Durham.

Young, Y.S., Kleinmann, S.G. & Allen, L.E., 1988. *Ap. J.*, **334**, 63.

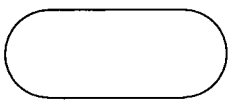
Zenner, S. & Lanzen, R., 1993. *A. & A. S.*, **101**, 363.

Appendix A - Microprocessor Command Set

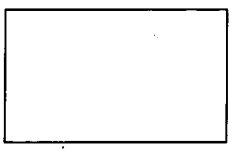
ASCII	Hex	Function
<LF>	0A	Line feed from controller
<CR>	0D	Initialise direct command
<SP>	20	Space character
#	23	Cancel
$	24	Index, run and sequence initialise
%	25	Wait for control
(	28	Baud rate detect
+	2B	Dir port high
,	2C	Delimiter
-	2D	Dir port low
/	2F	Linear interpolation
:	3A	Sequence divider
<	3C	Start / stop speed
=	3D	Repeat
>	3E	Select speed range
@	40	Speed data
A	41	Abort in sequence mode
B	42	Backlash
D	44	Inter character delay
E	45	Busy
F	46	Communications status
G	47	Run mode
H	48	Input high
I	49	Input status
K	4B	Motion status
L	4C	Input low
N	4E	Number of steps run or indexed

ASCII	Hex	Function
O	4F	Sequence status
P	50	Pulse output
Q	51	Quick status byte
R	52	Reset output low
S	53	Set output high
U	55	RS232C format set (termination on <CR>
V	56	RS232C format set (termination on <LF>
X	58	X clock
Y	59	Y clock
Z	5A	Z clock
^	5E	Acceleration rate data

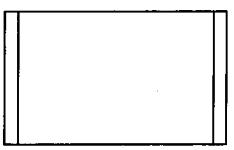
Appendix B - Flow Diagram Key



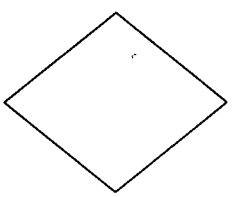
Process start



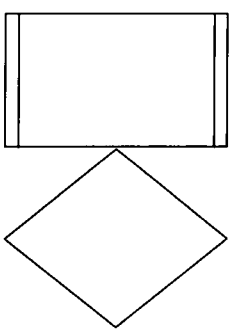
Process



Process defined elsewhere



Choice or Decision



Process and choice defined elsewhere

Appendix C - Polarimeter Control Software

HWP.C

```
/* This code communicates with the Digiplan stepper rack        */
/* to drive the half wave plate.                                */
/* the -Ml option causes errors - use "cc hwp.c"                */
/* execute me as "./a.out" or rename me as hwp                  */
/* check permissions on /dev/tty2 to see that you have read/write */
/* check permissions on /dev/tty3 to see that you have read/write */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"

/*********************************************************************/

int fd;              /* file descriptor for serial line */
FILE *fp1;           /* file pointer for config file    */
FILE *fp2;           /* file pointer for position file  */
FILE *fp3;           /* file pointer for log file       */
FILE *fp9;           /* file pointer for error file     */

char main_str[10];
char *main_str_pntr = &main_str[0];

char rx_buff[256];

struct config_struct con_str;
struct config_struct *config_data = &con_str;

int current_direction;     /* current direction of movement */
int error_check = OFF;     /* writes of all cmd sequences to a file */
int verbose = OFF;         /* displays all messages */
int error = FALSE;         /* flags if error occured when returning home */
int unit_type = HWP;       /* used to allow an offset */

long slot_distance = 250;   /* distance between slot centres */
long slot_home_dist = 72;   /* distance between home and home slot */
long slot_width = 12;       /* distance within slot */
int max_slots = 15;         /* max number of slots on device - should be 9 */
int backlash = 6;           /* backlash that motors need to take up */

int new_position;            /* position which to go to */
int old_position;            /* position which to go from */
int new_offset;              /* offset from slot which to go to */
```

```
int old_offset;              /* offset from which to go from */

int *new_pos_pntr = &new_position;   /* pointer to position which to go to */
int *old_pos_pntr = &old_position;   /* pointer to position which to go from */
int *new_off_pntr = &new_offset;     /* pointer to offset which to go to */
int *old_off_pntr = &old_offset;     /* pointer to offset which to go from */

/*****************************************************************/



void locate_centre()

/* locate the centre of the slot */
{
 int steps[4];
 int slot_width;
 int backlash;
 long back_steps;

 display_message("Locating centre of slot...\n",0);
 conditional_motor_command(FORWARD,SLOW,"L1");  /* go to end of slot */
 motion_status();
 conditional_motor_command(BACKWARD,SLOW,"H1");
 motion_status();
 steps[0] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(BACKWARD,SLOW,"L1");
 motion_status();
 steps[1] = steps_exec();          /* get number of steps across slot */
 conditional_motor_command(FORWARD,SLOW,"H1");
 motion_status();
 steps[2] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(FORWARD,SLOW,"L1");
 motion_status();
 steps[3] = steps_exec();          /* get number of steps across slot */
 backlash = (steps[0] + steps[2]) / 2;
 slot_width = (steps[1] + steps[3]) / 2;
 back_steps = backlash + (slot_width / 2);
 motor_command(BACKWARD,back_steps,SLOW);
 motion_status();
}


void locate_edge()

/* locate edge of slot */
{
 conditional_motor_command(FORWARD,SLOW,"L1");  /* go to end of slot */
 motion_status();
}


int check_at_home()

/* checks to see if at home slot */
{
 int pos_err;
```

```
  motor_command(BACKWARD,slot_home_dist,FAST);
  motion_status();
  pos_err = check_home();
  if (pos_err == PASS)
     {
     return(PASS);
     }
  else
     {
     return(FAIL);
     }
}


int find_home()

/* if all else fails - find home */
{
  printf("Locating home...");
  printf("%c%c%c%c\n",7,7,7,7);
  conditional_motor_command(BACKWARD,FAST,"L1H2");
  motion_status();
  conditional_motor_command(FORWARD,SLOW,"L1H2");
  motion_status();
  printf("moved %d steps\n",steps_exec());
  motor_command(FORWARD,slot_home_dist,SLOW);
  motion_status();
  locate_centre();
  motor_command(BACKWARD,slot_home_dist,SLOW);
  motion_status();
  return(PASS);
}


int go_to_position(position,offset)

int position;
int offset;

{
  long steps;
  int error_status = PASS;
  int attempts = 0;

  do
    {
    if (error_status == FAIL)
       {
       if (attempts == 2)
          {
          printf("Failed to find required position");
          printf("%c%c%c%c\n",7,7,7,7);
          return(FAIL);
          }
       find_home();
       attempts++;
       }
    else
```

```
        {
        error_status = FAIL;
        }
    if (position != 0)
        {
        steps = ((position * slot_distance) + (slot_home_dist));
        printf("Moving to slot : %d",position);
        if (offset < 10)
            {
            printf(" with offset : 0.0%d\n",offset);
            }
        else
            {
            printf(" with offset : 0.%d\n",offset);
            }
        motor_command(FORWARD,steps,FAST);
        motion_status();
        }
    else
        {
        steps = slot_home_dist;
        motor_command(FORWARD,steps,FAST);
        motion_status();
        }
   }
 while (check_slot() == FALSE || check_steps(steps) == FALSE);
 locate_edge();
 if (offset != 0)
    {
    steps = ((offset * slot_distance) / 100);
    motor_command(FORWARD,steps,FAST);
    motion_status();
    }
 return(PASS);
}


int go_to_home(pos_err,position,offset)

int pos_err;
int position;
int offset;

{
 long steps;
 int err_no = PASS;

 if (pos_err == PASS)              /* valid if managed to read old_pos */
    {
    steps = ((position * slot_distance) + ((offset * slot_distance) / 100) + (slot_width / 2));
    printf("Moving to home from slot : %d",position);
    if (offset < 10)
        printf(" with offset : 0.0%d\n",offset);
    else
        printf(" with offset : 0.%d\n",offset);
    motor_command(BACKWARD,steps,FAST);
    motion_status();
    pos_err = (check_slot() & check_steps(steps));
```

```
      if (pos_err == PASS)
         {
         pos_err = check_at_home();
         if (pos_err == FAIL)
            {
            error = TRUE;
            err_no = find_home();
            }
         }
      else
         {
         error = TRUE;
         err_no = find_home();
         }
      }
   else
      {
      err_no = find_home();
      }
   if (err_no == PASS)
      {
      return(PASS);
      }
   else
      {
      return(FAIL);
      }
}


int open_config()

/* open the configure file */
{
 fp1 = fopen("config.dat","r");
 if (fp1 == NULL)
    {
    printf("Error : cant open configuration file...");
    printf("%d\n",fp1);
    return(FAIL);
    }
 else
    {
    return(PASS);
    }
}


int get_config()

/* get the config info for rack,channel etc */
{
 int err_no;
 int ret;

 err_no = open_config();
 if (err_no == PASS)
    {
```

```
    /* read structures until HWP one is found */
    display_message("reading configure file...\n",0);
    do
      {
      ret = fread(config_data,16,1,fp1);
      (*config_data).name[3] = '\0';
      }
    while ((strcmp(config_data,"HWP") != 0) && (ret != 0));
    fclose(fp1);
    if (ret == 0)
       {
       printf("Error : could not find HWP configure record...\n");
       return(FAIL);
       }
    else
       {
       (*config_data).channel[1] = '\0';
       (*config_data).h_speed[4] = '\0';
       (*config_data).s_speed[4] = '\0';
       return(PASS);
       }
    }
}


int open_position(read_write)

int read_write;

/* open the configure file */
{
 if (read_write == READ)
    {
    fp3 = fopen("hwp.pos","r");
    }
 else
    {
    fp3 = fopen("hwp.pos","w");
    }
 if (fp3 == NULL)
    {
    printf("Error : cant open position file for HWP...");
    printf("%d\n",fp3);
    return(FAIL);
    }
 else
    {
    return(PASS);
    }
}


int write_position(new_position,new_offset)

int new_position;
int new_offset;

/* write the position */
```

```
{
 int err_no;

 err_no = open_position(WRITE);
 if (err_no == PASS)
    {
    itoa(new_position);
    fputs(main_str,fp3);
    fputs(".",fp3);
    itoa(new_offset);
    if (new_offset < 10)
       {
       fputs("0",fp3);
       }
    fputs(main_str,fp3);
    fclose(fp3);
    return(PASS);
    }
 else
    {
    return(FAIL);
    }
}


int read_old_position()

/* read the current position */
{
 int err_no;
 char position_str[6];
 char *position_str_ptr = position_str;                /* current position of HWP   */

 err_no = open_position(READ);
 if (err_no == PASS)
    {
    fgets(position_str_ptr,5,fp3);
    fclose(fp3);
    format_pos_str(position_str_ptr,old_pos_pntr,old_off_pntr);
    if (old_position > max_slots)
       {
       return(FAIL);
       }
    else
       {
       return(PASS);
       }
    }
 else
    {
    return(FAIL);
    }
}


int open_log()

/* open the log file */
```

```
 {
  fp2 = fopen("polar.log","a");
  if (fp2 == NULL)
    {
     printf("Error : cant open log file...");
     printf("%d\n",fp2);
     return(FAIL);
    }
  else
    return(PASS);
 }


 void write_log(position,offset)

 int position;
 int offset;

 /* write the log info */
 {
  int err_no;
  int ret;
  time_t t_time;
  time_t *tloc = &t_time;

  err_no = open_log();
  if (err_no == PASS)
     {
     t_time = time(tloc);
     if (error == TRUE)
        fputs("******** Error in returning home ********\n",fp2);
     fputs("HWP  ",fp2);
     fputs((*config_data).channel,fp2);
     fputs("  ",fp2);
     itoa(position);
     fputs(main_str,fp2);
     fputs(".",fp2);
     itoa(offset);
     fputs(main_str,fp2);
     fputs("  ",fp2);
     fputs(ctime(tloc),fp2);
     fclose(fp2);
     }
 }



 int main(argc,argv)


 int argc;
 char *argv[];

 /* main routine to move Half Wave Plate (HWP) to specified location */
 {
  int err_no;
  int pos_err;
  int return_error_code = 5;
```

```
  err_no = get_config();
  if (err_no == PASS)
     {
     return_error_code--;
     pos_err = read_old_position();
     if (argc == 1)
        get_new_position();
     else
        if (read_new_position(argc,argv) == FAIL)
           get_new_position();
     if (init_comm() == PASS && check_rack() == PASS)
        {
        return_error_code--;
        sensor_switch(ON);
        sleep(1);
        err_no = check_sensor();
        if (err_no == PASS)
           {
           return_error_code--;
           motor_switch(ON);
           sleep(1);
           err_no = go_to_home(pos_err,old_position,old_offset);
           if (err_no == PASS)
              {
              return_error_code--;
              err_no = go_to_position(new_position,new_offset);
              if (err_no == PASS)
                 return_error_code--;
              }
           motor_switch(OFF);
           write_position(new_position,new_offset);
           write_log(new_position,new_offset);
           }
        sensor_switch(OFF);
        }
     close_comm();
     }
  return(return_error_code);
}
```

filter.c

```
/* This code communicates with the Digiplan stepper rack        */
/* to drive the filter wheel.                                   */
/* the -Ml option causes errors - use "cc filter.c"             */
/* execute me as "./a.out" or rename me as filter               */
/* check permissions on /dev/tty2 to see that you have read/write */
/* check permissions on /dev/tty3 to see that you have read/write */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"


/*****************************************************************/

int fd;              /* file descriptor for serial line */
FILE *fp1;           /* file pointer for config file    */
FILE *fp2;           /* file pointer for position file  */
FILE *fp3;           /* file pointer for log file       */
FILE *fp9;           /* file pointer for error file     */

char main_str[10];
char *main_str_pntr = &main_str[0];

char rx_buff[256];

struct config_struct con_str;
struct config_struct *config_data = &con_str;

int current_direction;     /* current direction of movement */
int error_check = OFF;     /* writes of all cmd sequences to a file */
int verbose = OFF;         /* displays all messages */
int error = FALSE;         /* flags if error occured when returning home */
int unit_type = FILTER;    /* used to allow an offset */

long slot_distance = 750;  /* distance between slot centres */
long slot_home_dist = 52;  /* distance between home and home slot */
long slot_width = 18;      /* distance within slot */
int max_slots = 7;         /* max number of slots on device - should be 9 */
int backlash = 16;         /* backlash that motors need to take up */

int std_offset = 96;       /* offset to align filter with optical path */

int new_position;            /* position which to go to */
int old_position;            /* position which to go from */
int new_offset;              /* offset from slot which to go to */
int old_offset;              /* offset from which to go from */

int *new_pos_pntr = &new_position;   /* pointer to position which to go to */
int *old_pos_pntr = &old_position;   /* pointer to position which to go from */
```

```
int *new_off_pntr = &new_offset;     /* pointer to offset which to go to */
int *old_off_pntr = &old_offset;     /* pointer to offset which to go from */

/*****************************************************************/


void locate_centre()

/* locate the centre of the slot */
{
 int steps[4];
 int slot_width;
 int backlash;
 long back_steps;

 display_message("Locating centre of slot...\n",0);
 conditional_motor_command(FORWARD,SLOW,"L1");  /* go to end of slot */
 motion_status();
 conditional_motor_command(BACKWARD,SLOW,"H1");
 motion_status();
 steps[0] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(BACKWARD,SLOW,"L1");
 motion_status();
 steps[1] = steps_exec();          /* get number of steps across slot */
 conditional_motor_command(FORWARD,SLOW,"H1");
 motion_status();
 steps[2] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(FORWARD,SLOW,"L1");
 motion_status();
 steps[3] = steps_exec();          /* get number of steps across slot */
 backlash = (steps[0] + steps[2]) / 2;
 slot_width = (steps[1] + steps[3]) / 2;
 back_steps = backlash + (slot_width / 2);
 motor_command(BACKWARD,back_steps,SLOW);
 motion_status();
}


void locate_edge()

/* locate edge of slot */
{
 conditional_motor_command(FORWARD,SLOW,"L1");  /* go to end of slot */
 motion_status();
}


int check_at_home()

/* checks to see if at home slot */
}
 int pos_err;

 motor_command(BACKWARD,slot_home_dist,FAST);
 motion_status();
 pos_err = check_home();
 if (pos_err == PASS)
```

```
    {
    return(PASS);
    }
  else
    {
    return(FAIL);
    }
}


int find_home()

/* if all else fails - find home */
{
 printf("Locating home...");
 printf("%c%c%c%c\n",7,7,7,7);
 conditional_motor_command(BACKWARD,FAST,"L1H2");
 motion_status();
 conditional_motor_command(FORWARD,SLOW,"L1H2");
 motion_status();
 printf("moved %d steps\n",steps_exec());
 motor_command(FORWARD,slot_home_dist,SLOW);
 motion_status();
 locate_centre();
 motor_command(BACKWARD,slot_home_dist,SLOW);
 motion_status();
 return(PASS);
}


int go_to_position(position,offset)

int position;
int offset;

{
 long steps;
 int error_status = PASS;
 int attempts = 0;

 do
  {
  if (error_status == FAIL)
     {
     if (attempts == 2)
        {
        printf("Failed to find required position");
        printf("%c%c%c%c\n",7,7,7,7);
        return(FAIL);
        }
     find_home();
     attempts++;
     }
  else
     {
     error_status = FAIL;
     }
  if (position != 0)
```

```
        {
        steps = ((position * slot_distance) + (slot_home_dist));
        printf("Moving to slot : %d",position);
        if (offset < 10)
            {
            printf(" with offset : 0.0%d\n",offset);
            }
        else
            {
            printf(" with offset : 0.%d\n",offset);
            }
        motor_command(FORWARD,steps,FAST);
        motion_status();
        }
    else
        {
        steps = slot_home_dist;
        motor_command(FORWARD,steps,FAST);
        motion_status();
        }
    }
  while (check_slot() == FALSE || check_steps(steps) == FALSE);
  locate_edge();
  steps = (((offset * slot_distance) / 100) + ((std_offset * slot_distance) / 100));
  motor_command(FORWARD,steps,FAST);
  motion_status();
  return(PASS);
}


int go_to_home(pos_err,position,offset)

int pos_err;
int position;
int offset;

{
  long steps;
  int err_no = PASS;

  if (pos_err == PASS)              /* valid if managed to read old_pos */
    {
        steps = ((position * slot_distance) + ((offset * slot_distance) / 100) + (slot_width / 2) +
((std_offset * slot_distance) / 100));
    printf("Moving to home from slot : %d",position);
    if (offset < 10)
        printf(" with offset : 0.0%d\n",offset);
    else
        printf(" with offset : 0.%d\n",offset);
    motor_command(BACKWARD,steps,FAST);
    motion_status();
    pos_err = (check_slot() & check_steps(steps));
    if (pos_err == PASS)
        {
        pos_err = check_at_home();
        if (pos_err == FAIL)
            {
            error = TRUE;
```

```
            err_no = find_home();
            }
        }
     else
        {
        error = TRUE;
        err_no = find_home();
        }
    }
 else
    {
    err_no = find_home();
    }
 if (err_no == PASS)
    {
    return(PASS);
    }
 else
    {
    return(FAIL);
    }
}


int open_config()

/* open the configure file */
{
 fp1 = fopen("config.dat","r");
 if (fp1 == NULL)
    {
    printf("Error : cant open configuration file...");
    printf("%d\n",fp1);
    return(FAIL);
    }
 else
    {
    return(PASS);
    }
}


int get_config()

/* get the config info for rack,channel etc */

{
 int err_no;
 int ret;

 err_no = open_config();
 if (err_no == PASS)
    {
    /* read structures until FLT one is found */
    display_message("reading configure file...\n",0);
    do
      {
      ret = fread(config_data,16,1,fp1);
```

```
        (*config_data).name[3] = '\0';
        }
    while ((strcmp(config_data,"FLT") != 0) && (ret != 0));
    fclose(fp1);
    if (ret == 0)
        {
        printf("Error : could not find FLT configure record...\n");
        return(FAIL);
        }
    else
        {
        (*config_data).channel[1] = '\0';
        (*config_data).h_speed[4] = '\0';
        (*config_data).s_speed[4] = '\0';
        return(PASS);
        }
    }
}


int open_position(read_write)

int read_write;

/* open the configure file */
{
 if (read_write == READ)
    {
    fp3 = fopen("flt.pos","r");
    }
 else
    {
    fp3 = fopen("flt.pos","w");
    }
 if (fp3 == NULL)
    {
    printf("Error : cant open position file for FLT...");
    printf("%d\n",fp3);
    return(FAIL);
    }
 else
    {
    return(PASS);
    }
}


int write_position(new_position,new_offset)

int new_position;
int new_offset;

/* write the position */
{
 int err_no;

 err_no = open_position(WRITE);
 if (err_no == PASS)
```

```
    {
    itoa(new_position);
    fputs(main_str,fp3);
    fputs(".",fp3);
    itoa(new_offset);
    if (new_offset < 10)
       {
       fputs("0",fp3);
       }
    fputs(main_str,fp3);
    fclose(fp3);
    return(PASS);
    }
 else
    {
    return(FAIL);
    }
}


int read_old_position()

/* read the current position */
{
 int err_no;
 char position_str[6];
 char *position_str_ptr = position_str;                /* current position of HWP   */

 err_no = open_position(READ);
 if (err_no == PASS)
    {
    fgets(position_str_ptr,5,fp3);
    fclose(fp3);
    format_pos_str(position_str_ptr,old_pos_pntr,old_off_pntr);
    if (old_position > max_slots)
       {
       return(FAIL);
       }
    else
       {
       return(PASS);
       }
    }
 else
    {
    return(FAIL);
    }
}


int open_log()

/* open the log file */
{
 fp2 = fopen("polar.log","a");
 if (fp2 == NULL)
   {
   printf("Error : cant open log file...");
```

```
    printf("%d\n",fp2);
    return(FAIL);
    }
  else
    return(PASS);
}


void write_log(position,offset)

int position;
int offset;

/* write the log info */
{
  int err_no;
  int ret;
  time_t t_time;
  time_t *tloc = &t_time;

  err_no = open_log();
  if (err_no == PASS)
     {
     t_time = time(tloc);
if (error == TRUE)
        fputs("******** Error in returning home ********\n",fp2);
     fputs("HWP  ",fp2);
     fputs((*config_data).channel,fp2);
     fputs("  ",fp2);
     itoa(position);
     fputs(main_str,fp2);
     fputs(".",fp2);
     itoa(offset);
     fputs(main_str,fp2);
     fputs("  ",fp2);
     fputs(ctime(tloc),fp2);
     fclose(fp2);
     }
}



int main(argc,argv)

int argc;
char *argv[];

/* main routine to move Filter Wheel (FLT) to specified location */
{
  int err_no;
  int pos_err;
  int return_error_code = 5;


  err_no = get_config();
  if (err_no == PASS)
     {
     return_error_code--;
```

```
    pos_err = read_old_position();
    if (argc == 1)
       get_new_position();
    else
       if (read_new_position(argc,argv) == FAIL)
          get_new_position();
    if (init_comm() == PASS && check_rack() == PASS)
       {
       return_error_code--;
       sensor_switch(ON);
       sleep(1);
       err_no = check_sensor();
       if (err_no == PASS)
          {
          return_error_code--;
          motor_switch(ON);
          sleep(1);
          err_no = go_to_home(pos_err,old_position,old_offset);
          if (err_no == PASS)
             {
             return_error_code--;
             err_no = go_to_position(new_position,new_offset);
             if (err_no == PASS)
                return_error_code--;
             }
          motor_switch(OFF);
          write_position(new_position,new_offset);
          write_log(new_position,new_offset);
          }
       sensor_switch(OFF);
       }
    close_comm();
    }
 return(return_error_code);
}
```

focus.c

```
/* This code communicates with the Digiplan stepper rack        */
/* to drive the focus.                                          */
/* the -Ml option causes errors - use "cc focus.c"              */
/* execute me as "./a.out" or rename me as focus                */
/* check permissions on /dev/tty2 to see that you have read/write */
/* check permissions on /dev/tty3 to see that you have read/write */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"


/*****************************************************************/

int fd;              /* file descriptor for serial line */
FILE *fp1;           /* file pointer for config file    */
FILE *fp2;           /* file pointer for position file  */
FILE *fp3;           /* file pointer for log file       */
FILE *fp9;           /* file pointer for error file     */

char main_str[10];
char *main_str_pntr = &main_str[0];

char rx_buff[256];

struct config_struct con_str;
struct config_struct *config_data = &con_str;

int current_direction;     /* current direction of movement */
int error_check = OFF;     /* writes of all cmd sequences to a file */
int verbose = OFF;         /* displays all messages */
int error = FALSE;         /* flags if error occured when returning home */
int unit_type = FOCUS;     /* used to allow an offset */

long slot_distance = 150;  /* distance between slot centres */
long slot_home_dist = 23;  /* distance between home and home slot */
long slot_width = 22;      /* distance within slot */
int max_slots = 9;         /* max number of slots on device - should be 9 */
int backlash = 6;          /* backlash that motors need to take up */

int new_position;          /* position which to go to */
int old_position;          /* position which to go from */
int new_offset;            /* offset from slot which to go to */
int old_offset;            /* offset from which to go from */

int *new_pos_pntr = &new_position;   /* pointer to position which to go to */
int *old_pos_pntr = &old_position;   /* pointer to position which to go from */
int *new_off_pntr = &new_offset;     /* pointer to offset which to go to */
int *old_off_pntr = &old_offset;     /* pointer to offset which to go from */
```

```
/*****************************************************************/


void locate_centre()

/* locate the centre of the slot */
{
 int steps[4];
 int slot_width;
 int backlash;
 long back_steps;

 display_message("Locating centre of slot...\n",0);
 conditional_motor_command(FORWARD,SLOW,"L1"); /* go to end of slot */
 motion_status();
 conditional_motor_command(BACKWARD,SLOW,"H1");
 motion_status();
 steps[0] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(BACKWARD,SLOW,"L1");
 motion_status();
 steps[1] = steps_exec();          /* get number of steps across slot */
 conditional_motor_command(FORWARD,SLOW,"H1");
 motion_status();
 steps[2] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(FORWARD,SLOW,"L1");
 motion_status();
 steps[3] = steps_exec();          /* get number of steps across slot */
 backlash = (steps[0] + steps[2]) / 2;
 slot_width = (steps[1] + steps[3]) / 2;
 back_steps = backlash + (slot_width / 2);
 motor_command(BACKWARD,back_steps,SLOW);
 motion_status();
}


void locate_edge()

/* locate edge of slot */
{
 conditional_motor_command(BACKWARD,SLOW,"L1"); /* go to end of slot */
 motion_status();
}


int check_at_home()

/* checks to see if at home slot */
{
 int pos_err;

 motor_command(FORWARD,slot_home_dist,FAST);
 motion_status();
 pos_err = check_home();
 if (pos_err == PASS)
    {
    return(PASS);
```

```
    }
  else
    {
    return(FAIL);
    }
}


int find_home()

/* if all else fails - find home */
{
 printf("Locating home...");
 printf("%c%c%c%c\n",7,7,7,7);
 conditional_motor_command(FORWARD,SLOW,"L1H2");
 if (motion_status() == LIMIT)
    {
    do
      {
      if (current_direction == FORWARD)
          {
          printf("test point going back\n");
          conditional_motor_command(BACKWARD,SLOW,"L1H2");
          }
       else
          {
          printf("test point going forward\n");
          conditional_motor_command(FORWARD,SLOW,"L1H2");
          }
       }
    while (motion_status() == LIMIT);
    }
 printf("moved %d steps\n",steps_exec());
 motor_command(BACKWARD,slot_home_dist,SLOW);
 motion_status();
 locate_centre();
 motor_command(FORWARD,slot_home_dist,SLOW);
 motion_status();
 return(PASS);
}


int go_to_position(position,offset)

int position;
int offset;

{
 long steps;
 int error_status = PASS;
 int attempts = 0;

 do
   {
   if (error_status == FAIL)
       {
       if (attempts == 2)
          {
```

```
            printf("Failed to find required position");
            printf("%c%c%c%c\n",7,7,7,7);
            return(FAIL);
            }
         find_home();
         attempts++;
         }
      else
         {
         error_status = FAIL;
         }
      if (position != 0)
         {
         steps = ((position * slot_distance) + (slot_home_dist));
         printf("Moving to slot : %d",position);
         if (offset < 10)
            {
            printf(" with offset : 0.0%d\n",offset);
            }
         else
            {
            printf(" with offset : 0.%d\n",offset);
            }
         motor_command(BACKWARD,steps,FAST);
         motion_status();
         }
      else
         {
         steps = slot_home_dist;
         motor_command(BACKWARD,steps,FAST);
         motion_status();
         }
    }
  while (check_slot() == FALSE || check_steps(steps) == FALSE);
  locate_edge();
  if (offset != 0)
     {
     steps = ((offset * slot_distance) / 100);
     motor_command(BACKWARD,steps,FAST);
     motion_status();
     }
  return(PASS);
}


int go_to_home(pos_err,position,offset)

int pos_err;
int position;
int offset;

{
  long steps;
  int err_no = PASS;

  if (pos_err == PASS)              /* valid if managed to read old_pos */
     {
     steps = ((position * slot_distance) + ((offset * slot_distance) / 100) + (slot_width / 2));
```

```
        printf("Moving to home from slot : %d",position);
        if (offset < 10)
            printf(" with offset : 0.0%d\n",offset);
        else
            printf(" with offset : 0.%d\n",offset);
        motor_command(FORWARD,steps,FAST);
        motion_status();
        pos_err = (check_slot() & check_steps(steps));
        if (pos_err == PASS)
            {
            pos_err = check_at_home();
            if (pos_err == FAIL)
                {
                error = TRUE;
                err_no = find_home();
                }
            }
        else
            {
            error = TRUE;
            err_no = find_home();
            }
        }
    else
        {
        err_no = find_home();
        }
    if (err_no == PASS)
        {
        return(PASS);
        }
    else
        {
        return(FAIL);
        }
}


int open_config()

/* open the configure file */
{
    fp1 = fopen("config.dat","r");
    if (fp1 == NULL)
        {
        printf("Error : cant open configuration file...");
        printf("%d\n",fp1);
        return(FAIL);
        }
    else
        {
        return(PASS);
        }
}
```

```
int get_config()

/* get the config info for rack,channel etc */
{
 int err_no;
 int ret;

 err_no = open_config();
 if (err_no == PASS)
    {
    /* read structures until FOC one is found */
    display_message("reading configure file...\n",0);
    do
      {
      ret = fread(config_data,16,1,fp1);
      (*config_data).name[3] = '\0';
      }
    while ((strcmp(config_data,"FOC") != 0) && (ret != 0));
    fclose(fp1);
    if (ret == 0)
       {
       printf("Error : could not find FOC configure record...\n");
       return(FAIL);
       }
    else
       {
       (*config_data).channel[1] = '\0';
       (*config_data).h_speed[4] = '\0';
       (*config_data).s_speed[4] = '\0';
       return(PASS);
       }
    }
}


int open_position(read_write)

int read_write;

/* open the configure file */
{
 if (read_write == READ)
    {
    fp3 = fopen("foc.pos","r");
    }
 else
    {
    fp3 = fopen("foc.pos","w");
    }
 if (fp3 == NULL)
    {
    printf("Error : cant open position file for FOC...");
    printf("%d\n",fp3);
    return(FAIL);
    }
 else
    {
    return(PASS);
```

```
    }
}


int write_position(new_position,new_offset)

int new_position;
int new_offset;

/* write the position */
{
 int err_no;

 err_no = open_position(WRITE);
 if (err_no == PASS)
    {
    itoa(new_position);
    fputs(main_str,fp3);
    fputs(".",fp3);
    itoa(new_offset);
    if (new_offset < 10)
       {
       fputs("0",fp3);
       }
    fputs(main_str,fp3);
    fclose(fp3);
    return(PASS);
    }
 else
    {
    return(FAIL);
    }
}


int read_old_position()

/* read the current position */
{
 int err_no;
 char position_str[6];
 char *position_str_ptr = position_str;          /* current position of HWP  */

 err_no = open_position(READ);
 if (err_no == PASS)
    {
    fgets(position_str_ptr,5,fp3);
    fclose(fp3);
    format_pos_str(position_str_ptr,old_pos_pntr,old_off_pntr);
    if (old_position > max_slots)
       {
       return(FAIL);
       }
    else
       {
       return(PASS);
       }
    }
```

```
 else
    {
     return(FAIL);
    }
}


int open_log()

/* open the log file */
{
 fp2 = fopen("polar.log","a");
 if (fp2 == NULL)
   {
    printf("Error : cant open log file...");
    printf("%d\n",fp2);
    return(FAIL);
   }
 else
   return(PASS);
}


void write_log(position,offset)

int position;
int offset;

/* write the log info */
{
 int err_no;
 int ret;
 time_t t_time;
 time_t *tloc = &t_time;

 err_no = open_log();
 if (err_no == PASS)
    {
     t_time = time(tloc);
     if (error == TRUE)
        fputs("******** Error in returning home ********\n",fp2);
     fputs("HWP  ",fp2);
     fputs((*config_data).channel,fp2);
     fputs("  ",fp2);
     itoa(position);
     fputs(main_str,fp2);
     fputs(".",fp2);
     itoa(offset);
     fputs(main_str,fp2);
     fputs("  ",fp2);
     fputs(ctime(tloc),fp2);
     fclose(fp2);
    }
}
```

```
int main(argc,argv)


int argc;
char *argv[];

/* main routine to move FOCUS (FOC) to specified location */
{
 int err_no;
 int pos_err;
 int return_error_code = 5;


 err_no = get_config();
 if (err_no == PASS)
    {
    return_error_code--;
    pos_err = read_old_position();
    if (argc == 1)
       get_new_position();
    else
       if (read_new_position(argc,argv) == FAIL)
          get_new_position();
    if (init_comm() == PASS && check_rack() == PASS)
       {
       return_error_code--;
       sensor_switch(ON);
       sleep(1);
       err_no = check_sensor();
       if (err_no == PASS)
          {
          return_error_code--;
          motor_switch(ON);
          sleep(1);
          err_no = go_to_home(pos_err,old_position,old_offset);
          if (err_no == PASS)
             {
             return_error_code--;
             err_no = go_to_position(new_position,new_offset);
             if (err_no == PASS)
                return_error_code--;
             }
          motor_switch(OFF);
          write_position(new_position,new_offset);
          write_log(new_position,new_offset);
          }
       sensor_switch(OFF);
       }
    close_comm();
    }
 return(return_error_code);
}
```

grid.c

```
/* This code communicates with the Digiplan stepper rack        */
/* to drive the grids.                                          */
/* the -Ml option causes errors - use "cc grid.c"               */
/* execute me as "./a.out" or rename me as grid                 */
/* check permissions on /dev/tty2 to see that you have read/write */
/* check permissions on /dev/tty3 to see that you have read/write */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"

/*****************************************************************/

int fd;              /* file descriptor for serial line */
FILE *fp1;           /* file pointer for config file    */
FILE *fp2;           /* file pointer for position file  */
FILE *fp3;           /* file pointer for log file       */
FILE *fp9;           /* file pointer for error file     */

char main_str[10];
char *main_str_pntr = &main_str[0];

char rx_buff[256];

struct config_struct con_str;
struct config_struct *config_data = &con_str;

int current_direction;    /* current direction of movement */
int error_check = OFF;    /* writes of all cmd sequences to a file */
int verbose = OFF;        /* displays all messages */
int error = FALSE;        /* flags if error occured when returning home */
int unit_type = GRID;     /* used to allow an offset */

long slot_distance = 3150;  /* distance between slot centres */
long slot_home_dist = 1650; /* distance between home and home slot */
long slot_width = 738;      /* distance within slot */
int max_slots = 15;         /* max number of slots on device - should be 15 */
int backlash = 80;          /* backlash that motors need to take up */

int new_position;           /* position which to go to */
int old_position;           /* position which to go from */
int new_offset;             /* offset from slot which to go to */
int old_offset;             /* offset from which to go from */

int *new_pos_pntr = &new_position;   /* pointer to position which to go to */
int *old_pos_pntr = &old_position;   /* pointer to position which to go from */
int *new_off_pntr = &new_offset;     /* pointer to offset which to go to */
int *old_off_pntr = &old_offset;     /* pointer to offset which to go from */
```

```
/*****************************************************************/


void locate_centre()

/* locate the centre of the slot */
{
 int steps[4];
 int slot_width;
 int backlash;
 long back_steps;

 display_message("Locating centre of slot...\n",0);
 conditional_motor_command(BACKWARD,SLOW,"L1");  /* go to end of slot */
 motion_status();
 conditional_motor_command(FORWARD,SLOW,"H1");
 motion_status();
 steps[0] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(FORWARD,SLOW,"L1");
 motion_status();
 steps[1] = steps_exec();          /* get number of steps across slot */
 conditional_motor_command(BACKWARD,SLOW,"H1");
 motion_status();
 steps[2] = steps_exec();          /* get number of steps in backlash */
 conditional_motor_command(BACKWARD,SLOW,"L1");
 motion_status();
 steps[3] = steps_exec();          /* get number of steps across slot */
 backlash = (steps[0] + steps[2]) / 2;
 slot_width = (steps[1] + steps[3]) / 2;
 back_steps = backlash + (slot_width / 2);
 motor_command(FORWARD,back_steps,SLOW);
 motion_status();
}


void locate_edge()

/* locate edge of slot */
{
 conditional_motor_command(FORWARD,SLOW,"L1");  /* go to end of slot */
 motion_status();
}


int check_at_home()

/* checks to see if at home slot */
{
 int pos_err;

 motor_command(BACKWARD,slot_home_dist,FAST);
 motion_status();
 pos_err = check_home();
 if (pos_err == PASS)
    {
    return(PASS);
    }
```

```
 else
    {
    return(FAIL);
    }
}


int find_home()

/* if all else fails - find home */
{
 printf("Locating home...");
 printf("%c%c%c%c\n",7,7,7,7);
 conditional_motor_command(BACKWARD,SLOW,"L1H2");
 if (motion_status() == LIMIT)
    {
    do
      {
      if (current_direction == FORWARD)
          {
          printf("test point going back\n");
          conditional_motor_command(BACKWARD,SLOW,"L1H2");
          }
      else
          {
          printf("test point going forward\n");
          conditional_motor_command(FORWARD,SLOW,"L1H2");
          }
      }
    while (motion_status() == LIMIT);
    }
 printf("moved %d steps\n",steps_exec());
 motor_command(FORWARD,slot_home_dist,SLOW);
 motion_status();
 locate_centre();
 motor_command(BACKWARD,slot_home_dist,SLOW);
 motion_status();
 return(PASS);
}


int go_to_position(position,offset)

int position;
int offset;

{
 long steps;
 int error_status = PASS;
 int attempts = 0;

 do
  {
  if (error_status == FAIL)
      {
      if (attempts == 2)
          {
          printf("Failed to find required position");
```

```
            printf("%c%c%c%c\n",7,7,7,7);
            return(FAIL);
            }
          find_home();
          attempts++;
          }
       else
          {
          error_status = FAIL;
          }
       if (position != 0)
          {
          steps = ((position * slot_distance) + (slot_home_dist));
          printf("Moving to slot : %d",position);
          if (offset < 10)
             {
             printf(" with offset : 0.0%d\n",offset);
             }
          else
             {
             printf(" with offset : 0.%d\n",offset);
             }
          motor_command(FORWARD,steps,FAST);
          motion_status();
          }
       else
          {
          steps = slot_home_dist;
          motor_command(FORWARD,steps,FAST);
          motion_status();
          }
     }
   while (check_slot() == FALSE || check_steps(steps) == FALSE);
   locate_edge();
   if (offset != 0)
      {
      steps = ((offset * slot_distance) / 100);
      motor_command(FORWARD,steps,FAST);
      motion_status();
      }
   return(PASS);
}


int go_to_home(pos_err,position,offset)

int pos_err;
int position;
int offset;

{
 long steps;
 int err_no = PASS;

 if (pos_err == PASS)              /* valid if managed to read old_pos */
    {
    steps = ((position * slot_distance) + ((offset * slot_distance) / 100) + (slot_width / 2));
    printf("Moving to home from slot : %d",position);
```

```
        if (offset < 10)
            printf(" with offset : 0.0%d\n",offset);
        else
            printf(" with offset : 0.%d\n",offset);
        motor_command(BACKWARD,steps,FAST);
        motion_status();
        pos_err = (check_slot() & check_steps(steps));
        if (pos_err == PASS)
            {
            pos_err = check_at_home();
            if (pos_err == FAIL)
                {
                error = TRUE;
                err_no = find_home();
                }
            }
        else
            {
            error = TRUE;
            err_no = find_home();
            }
        }
    else
        {
        err_no = find_home();
        }
    if (err_no == PASS)
        {
        return(PASS);
        }
    else
        {
        return(FAIL);
        }
}


int open_config()

/* open the configure file */
{
    fp1 = fopen("config.dat","r");
    if (fp1 == NULL)
        {
        printf("Error : cant open configuration file...");
        printf("%d\n",fp1);
        return(FAIL);
        }
    else
        {
        return(PASS);
        }
}


int get_config()

/* get the config info for rack,channel etc */
```

```
{
 int err_no;
 int ret;

 err_no = open_config();
 if (err_no == PASS)
    {
    /* read structures until GRD one is found */
    display_message("reading configure file...\n",0);
    do
      {
      ret = fread(config_data,16,1,fp1);
      (*config_data).name[3] = '\0';
      }
    while ((strcmp(config_data,"GRD") != 0) && (ret != 0));
    fclose(fp1);
    if (ret == 0)
       {
       printf("Error : could not find GRD configure record...\n");
       return(FAIL);
       }
    else
       {
       (*config_data).channel[1] = '\0';
       (*config_data).h_speed[4] = '\0';
       (*config_data).s_speed[4] = '\0';
       return(PASS);
       }
    }
}


int open_position(read_write)

int read_write;

/* open the configure file */
{
 if (read_write == READ)
    {
    fp3 = fopen("grd.pos","r");
    }
 else
    {
    fp3 = fopen("grd.pos","w");
    }
 if (fp3 == NULL)
    {
    printf("Error : cant open position file for GRD...");
    printf("%d\n",fp3);
    return(FAIL);
    }
 else
    {
    return(PASS);
    }
}
```

```
int write_position(new_position,new_offset)

int new_position;
int new_offset;

/* write the position */
{
 int err_no;

 err_no = open_position(WRITE);
 if (err_no == PASS)
    {
    itoa(new_position);
    fputs(main_str,fp3);
    fputs(".",fp3);
    itoa(new_offset);
    if (new_offset < 10)
       {
       fputs("0",fp3);
       }
    fputs(main_str,fp3);
    fclose(fp3);
    return(PASS);
    }
 else
    {
    return(FAIL);
    }
}


int read_old_position()

/* read the current position */
{
 int err_no;
 char position_str[6];
 char *position_str_ptr = position_str;          /* current position of HWP  */

 err_no = open_position(READ);
 if (err_no == PASS)
    {
    fgets(position_str_ptr,5,fp3);
    fclose(fp3);
    format_pos_str(position_str_ptr,old_pos_pntr,old_off_pntr);
    if (old_position > max_slots)
       {
       return(FAIL);
       }
    else
       {
       return(PASS);
       }
    }
 else
```

```
    {
     return(FAIL);
    }
}


int open_log()

/* open the log file */
{
 fp2 = fopen("polar.log","a");
 if (fp2 == NULL)
   {
    printf("Error : cant open log file...");
    printf("%d\n",fp2);
    return(FAIL);
   }
 else
   return(PASS);
}


void write_log(position,offset)

int position;
int offset;

/* write the log info */
{
 int err_no;
 int ret;
 time_t t_time;
 time_t *tloc = &t_time;

 err_no = open_log();
 if (err_no == PASS)
    {
     t_time = time(tloc);
     if (error == TRUE)
        fputs("******** Error in returning home ********\n",fp2);
     fputs("HWP  ",fp2);
     fputs((*config_data).channel,fp2);
     fputs("  ",fp2);
     itoa(position);
     fputs(main_str,fp2);
     fputs(".",fp2);
     itoa(offset);
     fputs(main_str,fp2);
     fputs("  ",fp2);
     fputs(ctime(tloc),fp2);
     fclose(fp2);
    }
}
```

```
int main(argc,argv)


int argc;
char *argv[];

/* main routine to move Grid (GRD) to specified location */
{
 int err_no;
 int pos_err;
 int return_error_code = 5;


 err_no = get_config();
 if (err_no == PASS)
    {
    return_error_code--;
    pos_err = read_old_position();
    if (argc == 1)
       get_new_position();
    else
       if (read_new_position(argc,argv) == FAIL)
          get_new_position();
    if (init_comm() == PASS && check_rack() == PASS)
       {
       return_error_code--;
       sensor_switch(ON);
       sleep(1);
       err_no = check_sensor();
       if (err_no == PASS)
          {
          err_no = check_limit();
          if (err_no == PASS)
             {
             return_error_code--;
             motor_switch(ON);
             sleep(1);
             err_no = go_to_home(pos_err,old_position,old_offset);
             if (err_no == PASS)
                {
                return_error_code--;
                err_no = go_to_position(new_position,new_offset);
                if (err_no == PASS)
                   return_error_code--;
                }
             motor_switch(OFF);
             write_position(new_position,new_offset);
             write_log(new_position,new_offset);
             }
          }
       sensor_switch(OFF);
       }
    close_comm();
    }
 return(return_error_code);
}
```

polar.c

```
/*  Polar.c - all routines that are used in all four programs      */
/*  This program has been converted to run with the new single rack */
/*  and four card setup - A & B in first rack, C & D in second rack */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <stropts.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"



void ltoa(integer)

long integer;

/* converts a long integer to a string */
/* done by finding number of digits, then for each */
/* digit of the number, add ascii 0 to it */
{
int i;
int digits;
long t_integer;
char str[10];

t_integer = integer;              /* find how many digits in integer */
for (digits = 1; (t_integer = t_integer / 10) != 0; digits++);
t_integer = integer;                        /* use temporary variable */
str[digits] = '\0';       /* place string delimiter in correct place */
for (i = (digits - 1); i > -1; i--)
    {
    str[i] = (t_integer - ((t_integer / 10) * 10) + '0');
    t_integer = t_integer / 10;
    }
strcpy(main_str,str);
}


void itoa(integer)

int integer;

/* convert integer to string */
/* done by finding number of digits, then for each digit*/
/* of the number, add ascii 0 to it  */
{
int i;
int digits;
int t_integer;
```

```
char str[10];

t_integer = integer;              /* find how many digits in integer */
for (digits = 1; (t_integer = t_integer / 10) != 0; digits++);
t_integer = integer;                    /* use temporary variable */
str[digits] = '\0';       /* place string delimiter in correct place */
for (i = (digits - 1); i > -1; i--)
    {
    str[i] = (t_integer - ((t_integer / 10) * 10) + '0');
    t_integer = t_integer / 10;
    }
strcpy(main_str,str);
}



int strlim(str_pntr,n)

char *str_pntr;
int n;

/*  limits length of string to n chars */

{
 int i;

 if (strlen(str_pntr) > n)
    {
    *(str_pntr + n) = '\0';
    return(1);
    }
 else
    {
    return(0);
    }
}


void tx_string(cmd_sequence)

char cmd_sequence[];

/*  transmit a string to file fd */
/*  finds the string length and then transmits  */
/*  the string using write. The number of bytes */
/*  transmitted is compared to the string length*/
/*  and if different, an error message is sent  */
{
 int i;
 int tx_length , tx_ed;

 tx_length = strlen(cmd_sequence);
 tx_ed = write(fd,cmd_sequence,tx_length);
 ioctl(fd,I_FLUSH,"FLUSHW");
 if (error_check == ON)
    {
    fputs("{TX}",fp9);
    fputs(cmd_sequence,fp9);
```

```
    fflush(fp9);
    }
 if (tx_ed != tx_length)
    {
    printf("Error in transmission : ");
    printf("%s\n",cmd_sequence);
    }
}


void rx_string()

/*  read the rx channel                                          */
/*  a string, of max 256, is read from fd into rx_buff and */
/*  ended with a null                                            */
{
 int status_no;
 int rx_ed;

 sleep(1);
 rx_ed = read(fd,rx_buff,256);
 rx_buff[rx_ed] = '\0';  /*  set end of read string to null */
 if (error_check == ON)
    {
    fputs("{RX}",fp9);
    fputs(rx_buff,fp9);
    fputs("\n",fp9);
    fflush(fp9);
    }
}


void display_message(message,level)

char *message;
int level;

/* displays a message depending if in verbose mode or not */
{
 if (level > 0)
    printf("%s",message);
 else
    {
    if (verbose == ON)
       printf("%s",message);
    }
}


int steps_exec()

/*  find the number of steps executed during last  */
/*  movement, by sending N                         */
{
 char tx_buff[2];

 tx_string("N\n");
 rx_string();
```

```
 return(atoi(rx_buff));
}


void comms_fault()

/* check motion status or clear comms fault */
{
 int status_no;

 tx_string("F\n");
 rx_string();
 status_no = atoi(rx_buff);
 if ((status_no & 1) == 1)
    {
     printf("Parity error\n");
    }
 if ((status_no & 2) == 2)
    {
     printf("Overrun error\n");
    }
 if ((status_no & 4) == 4)
    {
     printf("Framing error\n");
    }
 if ((status_no & 8) == 8)
    {
     printf("Data fault or Buffer overflow\n");
    }
 if ((status_no & 16) == 16)
    {
     printf("Out-of-range error\n");
    }
}


int motion_status()

/* check motion status or clear comms fault */
{
 char tx_buff[2];
 int status_no;

 do
  {
   tx_string("K\n");
   rx_string();
   status_no = atoi(rx_buff);
  }
 while (status_no > 0 && status_no < 10);
 if (status_no == 0)
    {
     return(0);
    }
 if ((status_no & 16) == 16)
    {
     printf("Drive fault on rack !!\n");
     printf("Aborting with a core dump\n");
```

```
    abort();    /* causes all files to be closed and the core dumped */
    }
 if ((status_no & 32) == 32)
    {
    printf("Emergency stop pressed !!\n");
    printf("Aborting with a core dump\n");
    abort();    /* causes all files to be closed and the core dumped */
    }
 if ((status_no & 64) == 64)
    {
    printf("Limit reached!!\n");
    printf("Aborting with a core dump\n");
    abort();    /* causes all files to be closed and the core dumped */
    }
 if ((status_no & 128) == 128)
    {
    printf("Communication fault on rack !!\n");
    comms_fault();
    printf("Aborting with a core dump\n");
    abort();    /* causes all files to be closed and the core dumped */
    }
}


int check_rack()

/*  checks to see if rack will respond and whether in fault condition */
{
 tx_string("E\n");
 rx_string();
 if (rx_buff[0] == 'E' || rx_buff[0] == 'C')
    {
    return(PASS);
    }
 else if (rx_buff[0] == 'F')
    {
    display_message("Rack is in a fault condition, run fault",1);
    printf("%c%c%c%c\n",7,7,7,7);
    return(FAIL);
    }
 else
    {
    printf("Rack is not initialised...");
    printf("%c%c%c%c\n",7,7,7,7);
    return(FAIL);
    }
}


int check_sensor()

/* check to see if sensor is working - ie if current */
/*  is flowing thro' the LED.  SENSOR NEEDS TO BE ON */
{
 char tx_buff[2];
 int status_no;

 tx_string("I\n");
```

```
 rx_string();
 status_no = atoi(rx_buff);
 if ((status_no & 128) != 0)
     {
      printf("Error : sensor failure...\n");
      return(FAIL);
     }
 else
     {
      return(PASS);
     }
}


int check_steps(steps)

int steps;

/* check to see if actual steps executed is same */
/* as desired number of steps executed          */
{
 if (steps_exec() != steps)
     {
      printf("Error in motor steps - check !!!\n");
      printf("Steps executed : %d\n",steps_exec());
      printf("Steps required : %d\n",steps);
      return(FAIL);
      /* home automatically and then terminate */
     }
 else
     {
      return(PASS);
     }
}


int check_slot()

/* check to see if sensor 1 is over slot */
/* and that sensor 2 (home) is off       */
{
 char tx_buff[2];
 int status_no;

 tx_string("I\n");
 rx_string();
 status_no = atoi(rx_buff);
 display_message("doing check_slot...\n",0);
 if ((status_no & 8) == 0 && (status_no & 64) == 64)
     {
      return(PASS);
     }
 else
     {
      printf("Error : sensor not over slot\n");
      return(FAIL);
     }
}
```

```
int check_home()

/* check to see if sensor 2 is over hole (home position) */
{
 char tx_buff[2];
 int status_no;

 tx_string("I\n");
 rx_string();
 status_no = atoi(rx_buff);
 display_message("doing check_home...\n",0);
 if ((status_no & 8) == 8 && (status_no & 64) == 0)
     {
     return(PASS);
     }
 else
     {
     display_message("Error : sensor not over home\n",1);
     return(FAIL);
     }
}


int check_limit()

/* check to see if sensors 1 & 2 are on (limit position) */
{
 char tx_buff[2];
 int status_no;

 tx_string("I\n");
 rx_string();
 status_no = atoi(rx_buff);
 display_message("doing check_limit...\n",0);
 if ((status_no & 8) == 0 && (status_no & 64) == 0)
     {
     display_message("Error : Limit condition at switch on - CHECK\n",1);
     return(FAIL);
     }
 else
     {
     return(PASS);
     }
}


void motor_switch(off_on)

int off_on;

/* switch off/on channel 4 - motor energise  */
{
 char tx_buff[4];

 if (off_on == ON)
   tx_string("S4$\n");
```

```
  else
    tx_string("R4$\n");
}


void sensor_switch(off_on)

int off_on;

/*  switch off/on sensor output - number */
/*  dependant on the config_data of the */
/* configure file */
{

  char c;

  if (off_on == ON)
     {
     switch((*config_data).channel[0])
      {
      case 'A' : tx_string("S1$\n");
                 display_message("switching on sensor A...\n",0);
                 break;
      case 'B' : tx_string("S2$\n");
                 display_message("switching on sensor B...\n",0);
                 break;
      case 'C' : tx_string("S3$\n");
                 display_message("switching on sensor C...\n",0);
                 break;
      case 'D' : tx_string("S1$\n");
                 display_message("switching on sensor D...\n",0);
                 break;
      }
     tx_string("S5$\n");
     }
  else
     {
     switch((*config_data).channel[0])
      {
      case 'A' : tx_string("R1$\n");
                 display_message("switching off sensor A...\n",0);
                 break;
      case 'B' : tx_string("R2$\n");
                 display_message("switching off sensor B...\n",0);
                 break;
      case 'C' : tx_string("R3$\n");
                 display_message("switching off sensor C...\n",0);
                 break;
      case 'D' : tx_string("R1$\n");
                 display_message("switching off sensor D...\n",0);
                 break;
      }
     tx_string("R5$\n");
     }
}
```

```
void conditional_motor_command(direction,speed,condition)

int direction;
int speed;
char condition[];

{
 current_direction = direction;
 tx_string(":");
 switch((*config_data).channel[0])
   {
   case 'A' : tx_string("Z");
              break;
   case 'B' : tx_string("Y");
              break;
   case 'C' : tx_string("X");
              break;
   case 'D' : tx_string("X");
   }
 if (direction == BACKWARD)
    {
    tx_string("-@");
    }
 else
    {
    tx_string("@");
    }
 if (speed == FAST)
    {
    tx_string((*config_data).h_speed);
    }
 else
    {
    tx_string((*config_data).s_speed);
    }
 tx_string("G:");
 tx_string(condition);
 tx_string("A:$\n");
}


void motor_command(direction,steps,speed)

int direction;
long steps;
int speed;

/* move the motor a fixed number */
/* of steps at one of two speeds */
{
 current_direction = direction;
 ltoa(steps);     /* convert steps(int) to steps(string) , put in main_str */
 switch((*config_data).channel[0])
   {
   case 'A' : tx_string("Z");
              break;
   case 'B' : tx_string("Y");
              break;
```

```
    case 'C' : tx_string("X");
            break;
    case 'D' : tx_string("X");
    }
 if (direction == BACKWARD)
    {
     tx_string("-");
    }
 tx_string(main_str);        /* send the steps(string) */
 tx_string("@");             /* motor command */
 if (speed == FAST)
    {
     tx_string((*config_data).h_speed);
    }
 else
    {
     tx_string((*config_data).s_speed);
    }
 tx_string("$\n");
}


format_pos_str(position_str,position,offset)

char *position_str;
int *position;
int *offset;

/* formats position string into */
{
 int i , j;
 char *str_pnt1;
 char *str_pnt2;


 str_pnt1 = position_str;
 for (i = 0; (ispunct(j = *(position_str + i))) == FALSE && *(position_str + i) != '\0'; i++);
 if (j == '.' || j == '\0')
    {
     *(position_str + i) = '\0';          /* change the '.' to '\0' */
     i++;                                 /* skip over the new null */
     str_pnt2 = (position_str + i);       /* second string for frac */
     if (strlen(str_pnt2) == 1)           /* if offset .x then make .x0 */
        {
         strcat(str_pnt2,"0");
        }
     strlim(str_pnt2,2);
     *offset = atoi(str_pnt2);
    }
 strlim(str_pnt1,2);
 *position = atoi(str_pnt1);
 /* printf("%d and %d\n",*position,*offset); */
}


int check_position_value(position_str)

char *position_str;
```

```
/* checks if input value within range */
{
 int position;
 int offset;
 int *position_pntr = &position;
 int *offset_pntr = &offset;
 int c;

 format_pos_str(position_str,position_pntr,offset_pntr);
 if (*position_pntr <= max_slots)
    {
    if (unit_type == HWP || unit_type == FILTER)
       {
       if (*offset_pntr != 0)
          {
          display_message("Are you sure you require this offset [y,n] : ",1);
          c = getchar();
          putchar(c);
          if (c != 'y')
             return(FALSE);
          }
       }
    *new_pos_pntr = position;
    *new_off_pntr = offset;
    return(TRUE);
    }
 else
    {
    return(FALSE);
    }
}


void get_new_position()

/* gets a new position until valid */
{
 char *position_str;
 int  c , i;

 do
  {
   printf("Enter position : ");
   for (i = 0; (c = getchar()) != '\n'; i++)
       {
       putchar(c);
       *(position_str + i) = c;
       }
   *(position_str + i) = '\0';
   putchar('\n');
  }
 while (check_position_value(position_str) == FALSE);
}


int read_new_position(argc,argv)
```

```
int argc;
char *argv[];

/* reads the arguments on the command line and converts them to new position */
/* and switch operators                                                      */
{
 char pos_str[6];
 char *position_str = &pos_str[0];
 int digit_arg = FALSE;
 int i,j;

 for (i = 1;i != argc;i++)
     {
     strcpy(position_str,argv[i]);
     if (isdigit(pos_str[0]) != FALSE)
         {
         digit_arg = TRUE;
         if (check_position_value(position_str) == FALSE)
             {
             printf("Input value out of range");
             printf("%c%c%c%c%c\n",7,7,7,7,7);
             get_new_position();
             }
         }
     else if (pos_str[0] == '-')
         {
         for (j = 1;j != strlen(position_str);j++)
             {
             if (pos_str[j] == 'v')
                 verbose = ON;
             else if (pos_str[j] == 'e')
                 error_check = ON;
             else
                 printf("not a valid command line argument...\n");
             }
         }
     }
 if (digit_arg == FALSE)
    return(FAIL);
}
```

comm.c

```
/*  comms routine - to open and close the comms port          */
/*  This is the pc workstation running Microport System V/AT  */
/*  Modified to work with single rack housing system, with four */
/*  units.                                                    */

#include <fcntl.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"



int init_comm()

/* open the serial device and configure */
{
 struct termios io_settings;
 struct termios *termios_ptr = &io_settings;  /* see termios(3V) */
 int errflag;

 if (error_check == ON)
    fp9 = fopen("output.lst","w");  /* log file of all tx'ed and rx'ed */

 /* non blocking reads and synchronous writes */
 if ((*config_data).channel[0] == 'A' || (*config_data).channel[0] == 'B')
    {
    fd = open("/dev/ttya",O_NDELAY|O_RDWR|O_NOCTTY|O_SYNC);
    }
 else if ((*config_data).channel[0] == 'C' || (*config_data).channel[0] == 'D')
    {
    fd = open("/dev/ttya",O_NDELAY|O_RDWR|O_NOCTTY|O_SYNC);
    }
 else
    {
    printf("Error : cant open rack\n");
    return(FAIL);
    }
 if (fd < 0)
  {
  printf("Error : cant open rack on /dev/tty(a,b)\n");
  printf("%d\n",fd);
  return(FAIL);
  }

/* set line for no echo and correct speed */
errflag = tcgetattr(fd,termios_ptr);
if (errflag)
  printf("Get attributes failed\n");
```

```
 io_settings.c_cflag &= ~(CBAUD|PARENB);
 io_settings.c_cflag |= CS8 | CLOCAL | HUPCL | B2400;
 io_settings.c_lflag &= ~(ECHO|ICANON|ISIG);
 io_settings.c_iflag = 0;
 io_settings.c_cc[VEOF] = '\1';
 io_settings.c_cc[VEOL] = '\0';
 errflag = tcsetattr(fd,TCSANOW,termios_ptr);
 if (errflag)
   printf("ioctl failed\n");
 return(PASS);
}


int close_comm()

{
 close(fd);
 if (error_check == ON)
    fclose(fp9);
 return(PASS);
}
```

polar.h

```
/* polar.h   include file with various structure in to be */
/*          included in all polarimetry control files */

#define FALSE 0
#define TRUE 1
#define FAIL 0
#define PASS 1
#define OFF 0
#define ON 1
#define BACKWARD 0
#define FORWARD 1
#define READ 0
#define WRITE 1
#define SLOW 0
#define FAST 1
#define LOW 0
#define HIGH 1
#define LIMIT 1
#define UNDEF 0
#define HWP 1
#define FILTER 2
#define FOCUS 3
#define GRID 4
#define TOP 1
#define BOTTOM 2

struct config_struct
     {
     char name[4];
     char channel[2];
     char h_speed[5];
     char s_speed[5];
     };

extern int fd;              /* file descriptor for serial line */
extern FILE *fp1;           /* file pointer for config file    */
extern FILE *fp2;           /* file pointer for position file  */
extern FILE *fp3;           /* file pointer for log file       */
extern FILE *fp9;           /* file pointer for error file     */

extern char main_str[];
extern char *main_str_pntr;

extern char rx_buff[];

extern struct config_struct con_str;
extern struct config_struct *config_data;

extern int current_direction;  /* current direction of movement */
extern int error_check;        /* writes of all cmd sequences to a file */
extern int verbose;            /* displays all messages */
extern int error;              /* flags if error occured when returning home */
extern int unit_type;          /* type of unit ie HWP, FILTER etc. */
```

```
extern long slot_distance;     /* distance between slot centres */
extern long slot_home_dist;    /* distance between home and home slot */
extern long slot_width;        /* distance within slot */
extern int max_slots;          /* max number of slots on device - should be 9 */
extern int backlash;           /* backlash that motors need to take up */

extern int new_position;       /* position which to go to */
extern int old_position;       /* position which to go from */
extern int new_offset;         /* offset from slot which to go to */
extern int old_offset;         /* offset from which to go from */

extern int *new_pos_pntr;      /* pointer to position which to go to */
extern int *old_pos_pntr;      /* pointer to position which to go from */
extern int *new_off_pntr;      /* pointer to offset which to go to */
extern int *old_off_pntr;      /* pointer to offset which to go from */
```

initial.c

```
/* This code communicates with the Digiplan stepper rack        */
/* to initialise racks.                                         */
/* check permissions on /dev/ttya to see that you have read/write */
/* check permissions on /dev/ttyb to see that you have read/write */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"

/*****************************************************************/

int fd;                 /* file descriptor for serial line */
FILE *fp9;              /* file pointer for error file     */

char main_str[10];
char *main_str_pntr = &main_str[0];

char rx_buff[256];

struct config_struct con_str;
struct config_struct *config_data = &con_str;

/******** The following are not used but are referenced in polar.c ********/
int current_direction;
int unit_type;
int max_slots;
int *new_pos_pntr;
int *new_off_pntr;

int error_check = OFF;      /* writes of all cmd sequences to a file */
int verbose = OFF;          /* displays all messages */


/*****************************************************************/



int init_rack()

/* initialise rack */

{
 tx_string("(");
 sleep(1);
 rx_string();
 if (rx_buff[0] == 'U')
    {
     tx_string("U31\n");
     printf("Rack initialised\n");
```

```
      return(PASS);
      }
   else
      {
      printf("Rack failed to initialise or is already initialise\n");
      return(FAIL);
      }
}



main()

/* main routine to initialise racks */
{
 int err_no;

 (*config_data).channel[0] = 'A';
 err_no = init_comm();
 if (err_no == PASS)
      {
      init_rack();
      close_comm();
      }
}
```

fault.c

```
/* This code communicates with the Digiplan stepper rack        */
/* to determine a fault.                                        */
/* check permissions on /dev/tty2 to see that you have read/write */
/* check permissions on /dev/tty3 to see that you have read/write */


#include <fcntl.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include "polar.h"

/*****************************************************************/

static int fd;              /* file descriptor for serial line */
static FILE *fp1;           /* file pointer for config file    */
static FILE *fp2;           /* file pointer for position file  */
static FILE *fp3;           /* file pointer for log file       */
static FILE *fp9;           /* file pointer for error file     */

char main_str[10];
char *main_str_pntr = &main_str[0];

static char rx_buff[256];

struct config_struct con_str;
struct config_struct *config_data = &con_str;



/*************************************************************************/

void tx_string(cmd_sequence)

char cmd_sequence[];

/*  transmit a string to file fd */
/*  finds the string length and then transmits  */
/*  the string using write. The number of bytes */
/*  transmitted is compared to the string length*/
/*  and if different, an error message is sent  */
{
 int i;
 int tx_length , tx_ed;

 tx_length = strlen(cmd_sequence);
 tx_ed = write(fd,cmd_sequence,tx_length);
 fflush(fd);
 if (tx_ed != tx_length)
    {
     printf("Error in transmission : ");
     printf("%s\n",cmd_sequence);
```

```
    }
}


void rx_string()

/*  read the rx channel                                 */
/*  a string, of max 256, is read from fd into rx_buff and */
/*  ended with a null                                   */
{
 int status_no;
 int rx_ed;

 sleep(1);
 rx_ed = read(fd,rx_buff,256);
 rx_buff[rx_ed] = '\0';  /*  set end of read string to null */
}


void comms_fault()

/* check motion status or clear comms fault */
{
 int status_no;

 tx_string("F\n");
 rx_string();
 status_no = atoi(rx_buff);
 printf("Status code : %d - ",status_no);
 if ((status_no & 1) == 1)
    {
    printf("Parity error\n");
    }
 if ((status_no & 2) == 2)
    {
    printf("Overrun error\n");
    }
 if ((status_no & 4) == 4)
    {
    printf("Framing error\n");
    }
 if ((status_no & 8) == 8)
    {
    printf("Data fault or Buffer overflow\n");
    }
 if ((status_no & 16) == 16)
    {
    printf("Out-of-range error\n");
    }
}


int motion_status()

/* check motion status or clear comms fault */
{
 int status_no;
```

```
  tx_string("K\n");
  rx_string();
  status_no = atoi(rx_buff);
  printf("Rack 1(top)\n");
  printf("Status code : %d - ",status_no);
  if (status_no == 0)
     {
     printf("No detectable fault !!\n");
     }
  else
     {
     if ((status_no & 16) == 16)
        {
        printf("Drive fault\n");
        }
     if ((status_no & 32) == 32)
        {
        printf("Emergency stop pressed\n");
        }
     if ((status_no & 64) == 64)
        {
        printf("Limit reached\n");
        }
     if ((status_no & 128) == 128)
        {
        printf("Communications fault on rack");
        printf("%c%c%c%c%c%c%c\n",7,7,7,7,7,7,7);
        comms_fault();
        }
     }
}


void motor_switch(off_on)

int off_on;

/* switch off/on channel 4 - motor energise  */
{
 char tx_buff[4];

 if (off_on == ON)
    tx_string("S4$\n");
 else
    tx_string("R4$\n");
}


void sensor_switch(off_on)

int off_on;

/*  switch off/on sensor output - number */
/*  dependant on the config_data of the */
/* configure file */
{
  if (off_on == ON)
     {
```

```
        tx_string("S123$\n");
        }
    else
        {
        tx_string("R123$\n");
        }
}


int init_comm()

/* open the serial device and configure */
{
 struct termio termstruc;   /* see termio(7) in Runtime */
 int errflag;

 /* non blocking reads and synchronous writes */
 fd = open("/dev/tty2",O_NDELAY|O_RDWR|O_SYNC);
 if (fd < 0)
   {
    printf("Error : cant open rack on /dev/tty2\n");
    printf("%d\n",fd);
    return(FAIL);
   }
 /* set line for no echo and correct speed */
 errflag = ioctl(fd, TCGETA, &termstruc);
 if (errflag)
   printf("ioctl failed\n");
 termstruc.c_cflag &= ~(CBAUD|LOBLK);
 termstruc.c_cflag |= CS8 | CLOCAL | HUPCL | B2400;
 termstruc.c_lflag &= ~(ECHO|ICANON|ISIG);
 termstruc.c_iflag = 0;
 termstruc.c_cc[VMIN] = '\1';
 termstruc.c_cc[VTIME] = '\0';
 errflag = ioctl(fd, TCSETA, &termstruc);
 if (errflag)
   printf("ioctl failed\n");
 return(PASS);
}


int close_comm()

{
 fclose(fd);
 return(PASS);
}



main()

/* main routine to report fault */
{
 int err_no;

 err_no = init_comm();
```

```
 if (err_no == PASS)
    {
    motion_status();
    motor_switch(OFF);
    sensor_switch(OFF);
    close_comm();
    }
}
```

References

Armus, L., Heckman, T.M. & Miley, G.K., 1989. *Ap. J.*, **347**, 727.

Armus, L., Heckman, T.M. & Miley, G.K., 1990. *Ap. J.*, **364**, 471.

Armus, L., Heckman, T.M., Weaver, K.A. & Lenhert, M.D., 1995. *Ap. J.*, **445**, 666.

Arp, H. & Bertola, F., 1970. *Ap. Lett.*, **6**, 65.

Axon, D.J., 1977. *PhD thesis*, University of Durham.

Bingham, R. G., McMullen, D., Pallister, W.S., White, C., Axon, D.J. & Scarrott, S.M., 1976. *Nature*, **259**, 463.

Broadhurst, T. & Lehar, J., 1995. *Ap. J.*, **450**, L41.

Brown, R.L. & Vanden Bout, P.A., 1991. *Ap. J.*, **102**, 1956.

Burbidge, E.M. & Burbidge, G.R., 1968. *Ap. J.*, **151**, 99.

Casoli, F., Dupraz, C. & Combes, F., 1992. *A. & A.*, **264**, 49.

Chesterman, J.F. & Pallister, W.S., 1980. *M.N.R.A.S.*, **191**, 349.

Chevalier, R.A. & Clegg, A.W., 1985. *Nature*, **317**, 44.

Condon, J.J., Condon, M.A., Gisler, G. & Puschell, J.J., 1982. *Ap. J.*, **252**, 102.

Danks, A.C., Perez, M.R. & Altner, B., 1990. *"Bulges of Galaxies"*, ed. B.J. Jarvis & D.M. Terndrup (München:ESO), p.243.

Dahlem, M., Aalto, S., Kelin, U., Booth, R., Meobold, U., Wielebinski, R. & Lesch, H., 1990. *A. & A.*, **240**, 237.

Davis, L. & Greenstein, J.L., 1951. *Ap. J.*, **114**, 206.

Draine, B.T. & Lee, H.M., 1984. *Ap. J.*, **285**, 89.

Draper, P.W., 1988. *PhD thesis*, University of Durham.

Draper, P.W., Done, C., Scarrott, S.M. & Stockdale, D.P., 1995. *M.N.R.A.S.*, **277**, 1430.

Doyon, R., Joseph, R.D. & Wright, G.S., 1994. *Ap. J.*, **421**, 101.

Elsasser, H. & Staude, H.J., 1978. *A. & A.*, **70**, L3.

Field, G.B., Goldsmith, D.W. & Habing, H.J., 1969. *Ap. J.*, **155**, L149.

Forbes, D.A., Boisson, C. & Ward, M.J., 1992. *M.N.R.A.S.*, **259**, 293.

Gething, M.R., Warren-Smith, R.F., Scarrott, S.M. & Bingham, R.G., 1982. *M.N.R.A.S.*, **189**, 881.

Hernquist, L., 1989. *Nature*, **340**, 687.

Hsu, J. & Breger, M., 1882. *Ap. J.*, **262**, 732.

Hutchings, J.B., Neff, S.G., Stanford, S.A., Lo, E. & Unger, S.W., 1990. *A.J.*, **100**, 60.

Junkes, N., Pietsch, W. & Hensler, G., 1992. *COSPAR Symposium, "Advances in Space Research (Pergamon Press)"*, p154.

Koribalski, B., 1993. *PhD thesis*, University of Bonn.

Kronberg, P.P. & Biermann, P., 1981. *Ap. J.*, **243**, 89.

Larson, R.B. & Tinsley, B.M., 1978. *Ap. J.*, **219**, 46.

Menard, F., Bastien, P. & Roberts, C., 1988. *Ap.J.*, **335**, 290.

Mihos, J.C. & Hernquist, L., 1994. *Ap. J.*, **431**, L9.

Morgan, W.W., 1958. *P.A.S.P.*, **70**, 374.

Noguchi, M., 1991. *M.N.R.A.S.*, **251**, 360.

Norris, R.P. & Forbes, D.A., 1995. *Ap. J.*, **446**, 594.

Norman, C.A. & Ikeuchi, S., 1989. *Ap. J.*, **345**, 372.

Notni, P., 1985 *Astr. Nachr.*, **306**, 265.

Ohman, Y., 1939. *M.N.R.A.S.*, **99**, 624.

Pallister, W.S., 1976. *PhD thesis*, University of Durham.

Pickering, 1873. *American Academy of Arts and Science.*, **IX**, 1.

Pildis, R.A., Bergman, J.N. & Schombert, J.M., 1994. *Ap.J.*, **423**, 190.

Rieke, G.H., Cutri, R.M., Black, J.H., Kailey, W.F., Mcalary, C.W., Lebofsky, M.J. & Elston, R., 1985. *Ap.J.*, **290**, 116.

Reif, K., Meobold, U., Goss, W.M., van Woerden, H. & Siegman, B., 1982. *A. & A.S.*, **256**, 10.

Rieke, G.H., Lebofsky, M.J., Thompson, R.I., Low, F.J. & Tokunaga, A.T., 1980. *Ap. J.*, **238**, 24.

Rolph, C.D. 1990. *PhD thesis*, University of Durham.

Rowan-Robinson, M., Broadhurst, T., Oliver, S. J., Taylor, A. N., Lawrence, A., Mcmahon, R. G., Lonsdale, C. J., Hacking, P. B. & Conrow, T., 1991. *Nature*, **351**, 719.

Saikia, D.J., Pedlar, A., Unger, S.W. & Axon, D.J., 1994. *M.N.R.A.S.*, **270**, 46.

Saikia, D.J., Unger, S.W., Pedlar, A., Yates, G.J., Axon, D.J., Wolstencroft, R.D., Taylor, K. & Glydenkerne, K., 1990. *M.N.R.A.S.*, **245**, 397.

Sandage, A.R. & Tammann, G.A., 1987. *"A Revised Shapley-Ames Catalog of Bright Galaxies"*, (2nd ed.; Carnegie Inst. Washington Pub., Vol 635), [RSA].

Sanders, D.B., Scoville, N.Z. & Soifer, B.T., 1991. *Ap. J.*, **370**, 158.

Sargent, A.I., Sanders, D.B. & Phillips, T.G., 1989. *Ap. J.*, **346**, L9.

Scarrott, S.M., Draper, P.W., Stockdale, D.P., 1996. *M.N.R.A.S.*, **279**, 1325.

Scarrott, S.M., Draper, P.W., Stockdale, D.P. & Wolstencroft, R.D., 1993. *M.N.R.A.S.*, **264**, L7.

Scarrott, S.M., Draper, P.W., Warren-Smith, R.F., 1989. *M.N.R.A.S.*, **237**, 621.

Scarrott, S.M., Rolph, C.D. & Semple, D.P., 1989. *I.A.U.*, **140**, 245.

Scarrott, S.M., Ward-Thompson, D. & Warren-Smith, R.F., 1987. *M.N.R.A.S.*, **224**, 299.

Scarrott, S.M., Warren-Smith, R.F., Pallister, W.S., Axon, D.J. & Bingham, R.G., 1983. *M.N.R.A.S.*, **204**, 1163.

Schweizer, F., 1986. *Sci.*, **231**, 227.

Sérsic, J.L. & Pastoriza, M., 1965. *Pubs. astr. Soc. Pacif.*, **77**, 287.

Shapiro, P.R. & Field, G.B., 1976. *Ap. J.*, **205**, 762.

Smith, E.P. & Kassim, N.E., 1993. *A.J.*, **105**, 46.

Soifer, B. T., Neugebauer, G., Rowan-Robinson, M., Clegg, P. E., Emerson, J. P., Houck, J. R., De Jong, T., Aumann, H. H., Beichman, C. A. & Boggess, N., 1984. *Ap. J.*, **278**, L71.

Soifer, B.T., Sanders, D.B., Madore, B.F., Neugebauer, G., Danielson, G.E., Elias, J.H., Lonsdale, C.J. & Rice, W.L., 1987. *Ap.J.*, **320**, 238.

Suchcov, A.A., Balsara, D.S., Heckman, T.M. & Leitherer, C., 1994. *Ap. J.*, **430**, 511.

Telesco, C.M., 1988. *A. R. A. & A.*, **26**, 343.

Telesco, C.M. & Harper, D.A., 1980. *Ap. J.*, **235**, 392.

Toomre, A. & Toomre, J., 1972. *Ap. J.*, **178**, 623.

Véron-Cetty, M.-P. & Véron, P., 1985. *A. & A.*, **145**, 425.

Warren-Smith, R.F., 1979. *PhD thesis*, University of Durham.

Young, Y.S., Kleinmann, S.G. & Allen, L.E., 1988. *Ap. J.*, **334**, 63.

Zenner, S. & Lanzen, R., 1993. *A. & A. S.*, **101**, 363.

Acknowledgements

It has taken many years to complete this thesis on the part-time basis and during that time I have seen many students and PDRAs come and go, but I have only had one supervisor, Dr S. M. Scarrott. My sincere thanks go to Mike Scarrott who has instructed, overseen, advised, criticised and generally pushed me along to end; without him it would not have happened.

I would also like to thank Professor Sir Arnold Wolfendale, who as chairman of the board of studies when I started this degree, gave his full support to the idea of a member of the technical staff pursuing a higher degree. Professors Martin, Bloor and Tanner are also acknowledged for all the facilities provided by the Physics Department. My thanks go to all the Durham Polarimeterists over the years who have helped further my knowledge, provided fun and laughter and been good companions during long nights whilst observing. My special thanks go to Dr Peter Draper for answering what must have seemed like an endless barrage of questioning.

The support I received from my colleges in the Electronics Workshop, and especially Tom Jackson, was invaluable and for that I thank each of them, particularly my good friends, Clive Doloughan and Chris Mullaney. The polarimeter would not exist if it was not for the precise work of Phil Armstrong in the Teaching Services Workshop, again, for his help I am indebted. Thank-you to Alan Lotts, not only for the support of the Starlink facility, but also for the advise and guidance on all things computing.

Finally I must thank my immediate family who have supported me without question. Thank-you to Mum for both financial and moral support, without which I would have been forced to give up long ago. To my wife Claire I must say a special thank-you, she gave up a lot so that I could continue and although I sometimes appear to forget this, deep down I know what you have done for me.