

A Charge Coupled Device  
Star Sensor System  
for a Low Earth Orbit Microsatellite



by

BC Greyling

Thesis presented in partial fulfillment of the requirements for the  
degree of Master of Engineering at the University of Stellenbosch.  
1995

Study Leader : WH Steyn

I declare that, unless otherwise stated, this is my own work.

BC Greyling

27/9/95

To my parents and my wife, Annemie.

## Abstract

Star sensor systems provide extra accuracy to the attitude determination and control systems of spacecraft. These systems are generally complex and costly. This text describes the development of a low cost CCD star sensor system to meet the needs of a low earth orbit micro satellite. The attitude accuracy of this satellite is low (3 arc minutes) compared to other systems. This makes the use of cheaper and less sophisticated equipment possible.

An investigation was done into the system as a whole with more detailed study of the camera electronics, processing of the stellar image and pattern recognition techniques for finding the attitude. In all sub-sections of the system an attempt has been made to optimise the space of electronics and processing time of algorithms in keeping with the requirements of a micro satellite.

Algorithms for image segmentation, centroid determination, pattern recognition and attitude calculation were implemented and developed. A prototype star camera was developed using the TC211 CCD and a Cosmocar lens in order to gain experience in CCD camera design and show that such a low cost system is feasible.

It was found that such a project for the development and implementation of a star sensor system for a micro satellite is within reach of a small project group with limited resources and that this paper could serve as basis for such a venture.

## Samevatting

Stersensorstelsels verskaf ekstra akkuraatheid aan die standbepaling- en beheerstelsels van ruimtetuie. Sulke stelsels is gewoonlik ingewikkeld en duur. Die ontwikkeling van 'n lae koste CCD-stersensorstelsel om die behoeftes van 'n lae-aardwentelbaan mikrosatelliet te bevredig, word hier beskryf. Die standakkuraatheid van hierdie satelliet is laag (3 minute) in vergelyking met ander soortgelyke stelsels. Dit maak die gebruik van goedkoper en minder gesofistikeerde apparaat moontlik.

'n Ondersoek is gedoen na die algehele stelsel met meer noukeurige studie van die kamera-elektronika, verwerking van die sterrebeelde en patroonherkenningstegnieke vir standbepaling. Daar is in die analise van alle substelsels gepoog om die ruimte wat die elektronika beslaan en die uitvoertyd van algoritmes te optimaliseer in tred met die vereistes van 'n mikrosatelliet.

Algoritmes vir beeldsegmentering, beeldswaartepuntbepaling, patroonherkenning en standberekening is ontwikkel en geïmplimenteer. 'n Prototipe sterkamera is ontwikkel met die gebruik van die TC211 CCD en 'n Cosmical lens om ondervinding in CCD-kamera-ontwerp in te win en te wys dat 'n lae koste stelsel lewensvatbaar is.

Daar is bevind dat 'n projek vir die ontwikkeling en implimentering van 'n stersensorstelsel vir 'n mikrosatelliet binne die bereik is van 'n klein projekspan met beperkte hulpbronne en dat hierdie tesis as basis kan dien vir so 'n onderneming.

# Table of Contents

1. Introduction	1
2. Autonomous Satellite Navigation	3
2.1 Star Sensor Configurations	4
2.2 Sensor Hardware	6
2.2.1 CCD Technology	6
2.2.2 Electronics	8
2.3 Sensor Software	9
3. A Low Earth Orbit Microsatellite	11
3.1 Specifications and Requirements	11
3.2 The Satellite Orbit	12
3.3 Stellar Data Required for Attitude Determination	16
3.3.1 Star Catalogues	16
3.3.2 Stellar Distribution Density	17
3.3.3 FOV Size	22
3.4 Star Camera Calculations	23
3.4.1 CCD Responsivity	24
3.4.2 Sources of Noise	27
3.4.3 The Camera Lens	29
3.4.4 Stellar Magnitude Calculation	32
3.5 System Design	35
3.5.1 Hardware	35
3.5.2 Software	36
4. Finding Stars in the CCD Image	37
4.1 The Stellar Image Model	37
4.2 Image Segmentation	42
4.2.1 Region Growing Algorithm	42
4.2.2 Sources of Error	46
4.2.3 Simulation	47
4.3 Centroid Determination	48
4.3.1 Centroiding Techniques	48
4.3.2 Sources of Error	48
4.3.3 Simulation	49
5. Obtaining the Spacecraft Attitude	56
5.1 Star Pattern Recognition	56
5.1.1 Problem Definition	56
5.1.2 Recognition Algorithms	57
5.2 The Van Bezooijen Algorithm	58
5.2.1 The Guide Star Database	58
5.2.2 Algorithm Description	64
5.2.3 Simulation Program Description	67
5.2.4 Test Results	68
5.3 Calculation of the Attitude	69
5.3.1 Coordinate Translation and Rotation	69
5.3.2 Attitude Errors & Accuracy	71

6. A CCD Camera Prototype	74
6.1 Description	74
6.2 Electronic Circuit Design	76
6.2.1 PC Interface and Test Environment	76
6.2.2 Clock Generation Circuitry	77
6.2.3 Output Signal Conditioning	82
6.3 Prototype Measurements	86
Chapter 7 Summary and Conclusions	87
7.1 Summary	87
7.2 Conclusions	89
Appendix A	90
Appendix B	96
Appendix C	181
Table of Authorities	182

## List of Figures

Fig. 2.1 Structure of a CCD	6
Fig. 2.2 Spectral responses of a CCD and the human eye	7
Fig. 2.3 Software components	9
Fig. 3.1 The satellite orbit	37
Fig. 3.2 The local level coordinates	14
Fig. 3.3 The path of the camera boresight on the celestial sphere	15
Fig. 3.4 Stellar Distribution density	18
Fig. 3.5 Stellar distribution density along the boresight path ( $m_v=5.0$ )	19
Fig. 3.6 Stellar distribution density along the boresight path ( $m_v=6.0$ )	20
Fig. 3.7 Stellar distribution density along the boresight path ( $m_v=7.0$ )	21
Fig. 3.8 Minimum distribution density	22
Fig. 3.9 Components of the CCD Camera	24
Fig. 3.10 Quantum efficiency of a CCD	25
Fig. 3.11 Frequency response of the TC211	25
Fig. 3.12 Calculated irradiances of some stars	27
Fig. 3.13 Output voltage vs. Lens Diameter vs. Integration time ( $m_v=6.5$ )	32
Fig. 3.14 Output voltage vs. Lens Diameter vs. Integration time ( $m_v=-1.47$ )	32
Fig. 3.15 Output voltage vs. Visual magnitude	35
Fig. 3.16 System hardware	36
Fig. 3.17 System software functional flowchart	37
Fig. 4.1 The Airy function	38
Fig. 4.2 The Airy and Gaussian functions	39
Fig. 4.3 Relative error between Airy and Gaussian ( $\sigma=1.3497$ ) functions	40
Fig. 4.4 Gaussian pixel intensity spread	41
Fig. 4.5 The region growing algorithm	42
Fig. 4.6 Fast search pattern	43
Fig. 4.7 Image Segmentation Flow Diagram	45
Fig. 4.8 Intensity distribution for offset from 0 to 0.5	49
Fig. 4.9 Algorithm bias error	50
Fig. 4.10 Noise error	51
Fig. 4.11 Maximum noise error	52
Fig. 4.12 Maximum error vs. smear direction	53
Fig. 4.13 Maximum error vs. distortion	54
Fig. 5.1 The attitude determination problem	56
Fig. 5.2 Number of lines versus points	59
Fig. 5.3 Guide star selection	62
Fig. 5.4 Catalogue zones	63
Fig. 5.5 Number of stars in zones	63
Fig. 5.6 Algorithm flow diagram	65
Fig. 5.7 The mirror test	66
Fig. 5.8 Star map and FOV coordinates	70
Fig. 5.9 Star map and FOV coordinates	71
Fig. 5.10 Pitch angle accuracy	72
Fig. 5.11 Pitch angle accuracy	73
Fig. 6.1 Functional block diagram of the TC211	74
Fig. 6.2 CCD and lens configuration	75

Fig. 6.3 Functional block diagram of the camera electronics	76
Fig. 6.4 Modes of operation of TC211	77
Fig. 6.5 Clock signals IAG and SRG	78
Fig. 6.6 Camera State machine	80
Fig. 6.7 Circuit diagram of ASM implementation	82
Fig. 6.8 Output and associated signals	83
Fig. 6.9 Image with incorrect focus	84
Fig. 6.10 Image with slow clock smear	85
Fig. 6.11 Image with 2MHz clock	86

## List of Tables

Table 2.1. Typical AD&CS Sensors.	3
Table 2.2. Examples of star sensors.	5
Table 3.1. Specifications for the star sensor system	11
Table 3.2. Orbital parameters	13
Table 3.3. Stars and Magnitudes	17
Table 3.4. Stars and Magnitudes	26
Table 4.1. Algorithm bias error	50
Table 4.2. Accuracies for different CCDs	56
Table 5.1. Number of stars and lines for different magnitudes	61
Table 5.2. Van Bezooijen algorithm test results	68
Table 6.1. Control addresses	76
Table 6.2. State table of the ASM chart	81

## Glossary

AD&CS	Attitude Determination and Control System
ASM	Algorithmic State Machine
CCD	Charge Coupled Device
CMOS	Metal Oxide Silicon
dec	Declination
DSNU	Dark Signal Non-Uniformity
DSP	Digital Signal Processing
FOV	Field Of View
LEO	Low Earth Orbit
LMT	Local Mean Time
PAL	Programmable Array Logic
RA	Right Ascension
RAM	Random Access Memory
SNR	Signal to Noise Ratio

# 1. Introduction

Navigation systems for satellites vary in complexity and accuracy depending on the needs of the project. A significant number of applications, particularly those related to remote sensing and communications, are geocentric pointing. This requirement dominates attitude determination and control system (AD&CS) design. Orbits can be circular, elliptical with low perigee passages, sun-synchronous, etc. Moreover, once it has been achieved, an orbit may or may not need to be controlled. The selection of sensors/actuators for control and attitude determination will differ depending on whether a three-axis or a spin stabilised configuration is required.

In this application, the spacecraft is assumed to be a sun-synchronous, low earth orbit (LEO) microsatellite. Apart from the sun and horizon sensors, one or more star sensors are also needed to ensure accurate pointing for earth imaging.

Since the advent of integrated circuits the size, weight and power consumption of satellite systems have decreased considerably. One of the key technological advances in electro-optical devices is the charge coupled device or CCD. These devices are physically small but robust, have a low power consumption and are relatively immune to burn-in. All the above mentioned characteristics are ideal for the space environment. A further advantage is that CCDs offer high sensitivity at a lower price than for instance photo multiplier tubes.

This project has attempted to develop a star sensor system for a LEO microsatellite. The most important task of the system is to add the extra accuracy that is needed for stability during push broom imaging of the earth to the AD&CS.

Chapter 2 deals with autonomous navigation systems in general and with CCD star sensor systems in particular.

In Chapter 3 the specific application of a star sensor system for a microsatellite is studied. The requirements for the mission are stated and possible solutions for the implementation of the different parts of the system are investigated.

Chapter 4 presents image processing algorithms and programs that are used to obtain the positions of stellar images in the CCD array. The performance of particular algorithms are tested with computer simulations.

Star identification algorithms and attitude determination techniques are investigated in Chapter 5. A simulation program implementing the Van Bezooijen [1989] star pattern recognition technique is described and tested.

Chapter 6 describes the development and test results of a prototype CCD camera. The TC211 CCD from Texas Instruments was used for this purpose.

Finally, work done is summarised in Chapter 7. Conclusions and recommendations for future developments are made.

## 2. Autonomous Satellite Navigation

Satellite navigation on an autonomous basis is the ability of the AD&CS of the satellite to determine its own position and velocity in real time. A greater or lesser degree of autonomy requires more or less maintenance and input of information from earth stations.

The attitude determination systems used on spacecraft vary in size, accuracy, cost and complexity. Some, such as gyroscopes, have highly intricate mechanical constructions while others, such as sun sensors, are fairly simple in design and have no moving parts. Table 2.1 [4th AIAA/USU Conference, 1990] gives examples of a number of widely used sensors.

**Table 2.1. Typical AD&CS Sensors.**

Sensor	Performance	Weight (kg)	Power (W)
Gyroscope	0.003°/hr - 1°/hr	3 - 25	10 - 200
Sun sensor	1' - 3°	0.5 - 2	0 - 3
Star sensor	1" - 1'	3 - 7	5 - 20
Horizon sensor	0.1° - 1°	2 - 5	5 - 10
Magnetometer	0.5° - 3°	0.6 - 1.2	< 1

The sensor configuration for a satellite depends on the size of the spacecraft, the type of mission and of course available funds.

Inertial measurement units, such as gyroscope systems, use the fixed orientation of a spinning gyroscope to provide a reference for attitude measurements. Sun sensors use solid state visible light (or infrared radiation) sensitive devices to obtain an estimate of the sun's position. Horizon sensors are also light sensitive and use the illuminated limb of the earth for attitude measurements.

To achieve a greater or lesser degree of autonomy the AD&CS uses combinations of these sensors [4th AIAA/USU Conference,1990] to provide the required accuracy and redundancy for the particular mission.

A typical configuration could be :

1. a sun sensor to initially acquire the spacecraft attitude from an unknown orientation;
2. a gyro to provide a stable, accurate reference when doing maneuvers; and
3. a pair of horizon sensors to update gyro data and provide the extra accuracy to  $0.1^\circ$ .

To achieve this accuracy, the horizon sensors are the best choice because they cost less than star sensors.

The above-mentioned sensors are intrinsically not very accurate with regard to visible light. One of the reasons is that neither the sun nor the earth's albedo is an object with well-defined edges in the visible wavelengths. This can be attributed to the sun's corona and the atmosphere of the earth respectively. The solution is then to use sensors that are sensitive to other parts of the electromagnetic spectrum. These sensors are, because of their special characteristics, very costly, and thus eliminates the possibility of using them as high accuracy devices in this case. If a greater accuracy is required, say  $< 0.1^\circ$ , while the cost is to be kept low, star sensors should be considered.

## 2.1 Star Sensor Configurations

Star sensors provide a satellite with a high level of autonomy. Apart from fixing the attitude of the satellite to a chosen stellar direction, stars can be identified from which the absolute attitude and position of the satellite can be calculated.

There are three classes of star sensor configurations. They are scanners, trackers and mappers [4th AIAA/USU Conference,1990]. These three are differentiated by the hardware they employ and by the way in which they operate. Scanners are usually found on rotating satellites. The vehicle's attitude is derived by measurements done on multiple images of stars passing through slits in the scanner's field of view (FOV). A star tracker, on the other hand, uses a wide FOV and the scanning of the image is done electronically. Once a star of predetermined brightness has been found, its position in the FOV is monitored. Any motion of the spacecraft will subsequently show up as an apparent shift in the stellar position in the FOV and the error signal derived from this technique can be used to hold a vehicle fixed in inertial space. Mappers are similar to trackers, but use a number of stars in their FOVs. An image is captured and then stored data is used to determine the positions of the stars in the FOV from which the inertial position of the satellite in space can be calculated. All star sensors use some kind of camera to record stellar images. The type and size of the camera hardware depends on the application for which it is intended.

Table 2.2. Examples of star sensors.

Type of star sensor	FOV	Accuracy (arcsec)	Magnitude ( $m_v$ )	Acquisition time (sec)	Max. star rate	Weight (kg)	Input power (W)	Memory capacity (k RAM)
Galileo star scanner	10°	60	200 brightest stars	N/A	20°/sec	20	3	2
SED 12 CCD Star tracker	7.5 x 10°	3	-1 to +8	< 8	0.1°/sec	6.5	2	16-32
Honeywell star sensor (mapper)	7 x 9°	1	< 6	10	N/A	6	20	214

Scanner cameras generally point in a direction perpendicular to the spin axis of the satellite. They need record only the latitude at which a stellar image passes the slit.

Tracker cameras have a wide FOV. They scan electronically until a star of predetermined brightness is reached, whereafter the movement of that particular star in the FOV is used to determine any motion of the satellite. This error signal can then be used to keep the spacecraft fixed in inertial space.

Mappers differ from scanners in that they use the positions of several stars in the FOV. A snapshot of the stars is made and stored in memory. Stored stellar data is then used to determine the position of the satellite in inertial space.

The normal mode for operation of a star sensor is to use the sensor for sporadic updates of the spacecraft attitude. The microsatellite under consideration here is of the spinning kind (rotation around its yaw axis), but the spin of the satellite will be stopped at certain times. It is during these non-spinning, three axis stabilized periods that the star sensor will be most useful. High attitude accuracy for stability is only required during certain parts of the orbit, during those periods when areas of the surface of the earth will be photographed.

## 2.2 Sensor Hardware

The lens of the star camera will be an off-the-shelf item. For this reason different types of lenses and optical configurations will therefore not be discussed (Chapter 3 presents calculations for specifying a lens for the system). Overviews of CCD technology and the electronics for the camera are given here.

### 2.2.1 CCD Technology

Charge coupled device (CCD) technology is already more than 20 years old. The basic structure of the CCD is illustrated in Figure 2.1.

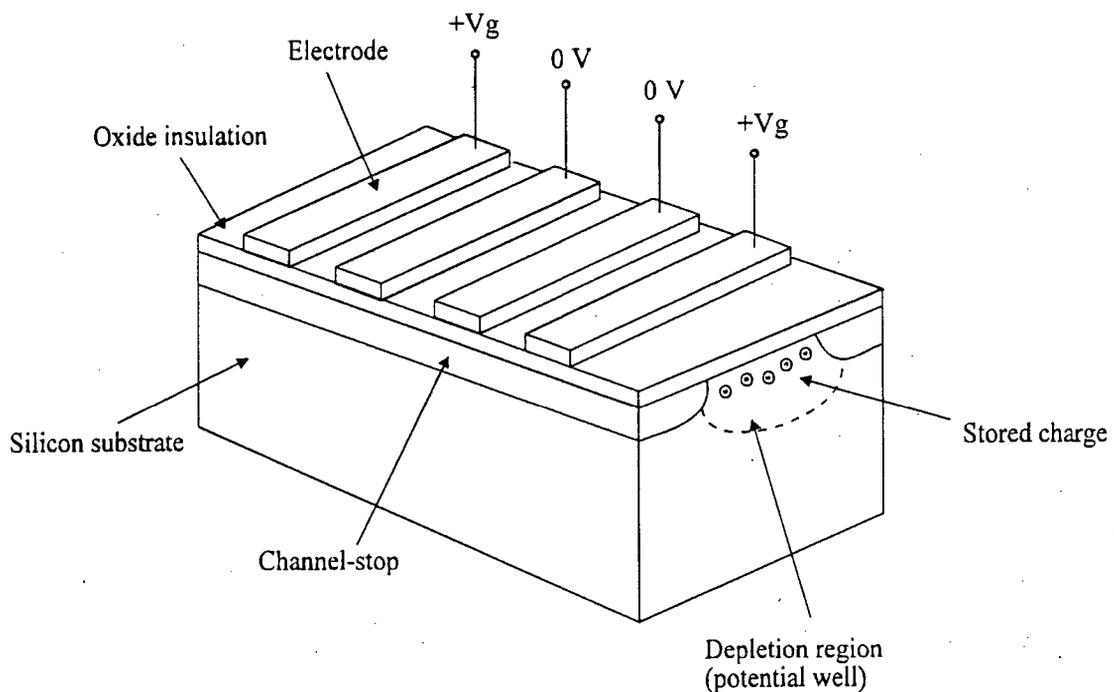
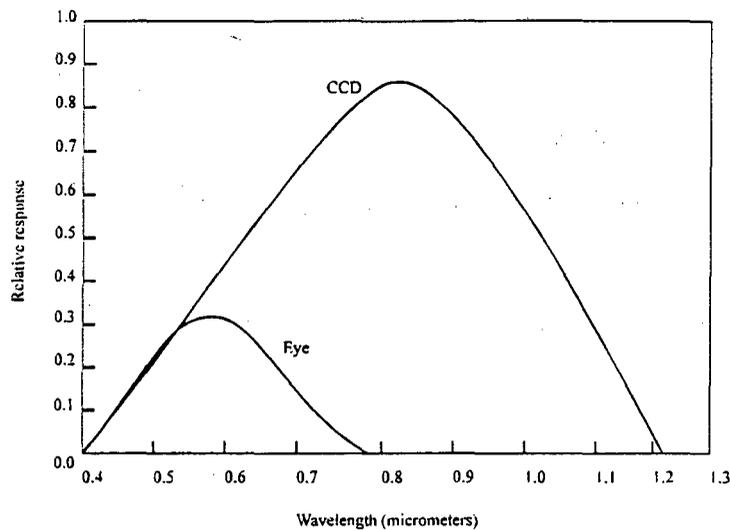


Fig. 2.1 Structure of a CCD

A silicon substrate is covered by an insulating oxide layer on which there are closely spaced electrodes. The channel under the electrodes is bounded by electrically inactive p-type channel stop regions. Light photons penetrate the silicon and create electron/hole pairs in the substrate by means of the Einstein photoelectric effect. CCDs generally have different spectral responses to that of the human eye. As Figure 2.2 shows, the relative response is at its maximum in the near infrared region of the electromagnetic spectrum. This sensitivity to visible light makes the CCD ideal for optical applications.



**Fig. 2.2 Spectral responses of a CCD and the human eye**

If a positive voltage is applied to an electrode, the area underneath it is in depletion and will be positively charged. The electrons therefore localize below the electrodes with the highest positive voltages while the hole is effectively lost into the substrate. Charge is thereby stored in "packets" under these electrodes. Charge coupling is the way in which the voltages of subsequent electrodes are varied in order to move the charge packets in a desired direction from underneath one electrode to underneath the next.

In a linear,  $1 \times N$  pixel device, signal charge is transferred along the electrodes to an output amplifier. After the integration period the values of the charge for each photo site, or pixel, are clocked out and stored. An area,  $N \times N$ , CCD works in much the same way. After integration, rows of pixels are clocked into the output register, from where they are clocked out serially and stored.

Clock signals typically have amplitudes of 10 V and are applied in phases. In a three-phase device each photosensitive element is an area bounded by the channel-stop and the triplet of electrodes centered on the biased electrode.

The advantages of CCDs over beam-scanned image tubes are myriad. In respect of space applications, the most important attributes of CCDs are their small size, light weight and robustness to environmental stress. The solid state nature of these devices enables them to withstand the mechanical shock and temperature fluctuations that satellite components have to endure. CCDs also have other qualities, such as : long life-time, low power consumption, good sensitivity, low image distortion, low burn-in, no image lag (all the charge in the device is removed after every integration period). Many CCDs also provide anti-blooming facilities to stop the spread of charge around an area on the CCD with high light intensity.

Matrix-type ( $N \times N$ ) CCDs, as are used for the star camera in this application, are available from a number of manufacturers in many different sizes. A typical format is  $512 \times 512$  with a sensitivity of about 2 lux. The upper end in terms of the amount of pixels is at about  $1024 \times 1024$ , but is expected to rise as manufacturing techniques improve. Prices range from R100 to about R50 000 depending on size and quality.

### 2.2.2 Electronics

The electronics of a CCD star sensor consists of the camera electronics, memory storage electronics and some form of processing power to control the camera and perform real-time calculations on the data.

Digital clock signals for CCDs have frequencies in the order of 1 MHz [Thomson, 1974]. Solid state components which operate at this frequency level are relatively cheap and readily available. TTL logic devices can be used to design the state machine for the CCD camera clock signals. The LS, HCT and ACT range of CMOS devices provides a wide variety of digital electronic components. A prototype can be developed using the cheapest range and once the design has been tested the final product can be built using military specification (or radiation-hardened) components.

Control of the camera should come from the on-board AD&CS computer. Generally the AD&CS processor is chosen without taking the requirements of a star sensor into consideration, and this should not pose a problem as the AD&CS computer has many processing tasks to fulfill and can easily cope with any demands made by a star sensor. Keeping all of the above in mind it is clear that the design of the star sensor should be such that the control and memory interfaces to the processor are kept as standard as possible in order for them to be adaptable to any kind of mission. Such a star sensor system could then be an off-the-shelf item which could be purchased and integrated into any satellite AD&CS system with a minimum of effort.

In the case of this design, the camera (Chapter 6) has its own memory space for storing an image. Address lines of the AD&CS are decoded by the camera electronics to ensure that the processor has direct access to the stored image. The only memory space that the AD&CS computer has to cater for, is for the storage of stellar information. A further enhancement for improved performance is the implementation of pixel correction. The particular CCD device that will be used in the camera should be examined and the positions and sensitivity offsets of any blemishes on the CCD surface recorded. After an image has been clocked into memory, the pixel correction information is used to correct any errors by adjusting the digital values of the blemished pixels. The implementation of this correction can be done by electronics on the camera or by the AD&CS computer.

## 2.3 Sensor Software

Star sensor software is generally more complex than that of other sensors. Processing involves identifying observed stars with those found in a star catalogue, which can require extensive data and memory resources. These star ephemerides can be maintained using crude lookup tables or very complex algorithms.

Star sensor software has to be integrated with the rest of the AD&CS software. To keep the software of the complete system homogenous and easy to maintain and enhance, certain design aspects have to be taken into account. Modular design is of the greatest importance to the management of complex software. Without well-defined tasks and coding standards, software management could easily become a nightmare. Today many high-level languages provide the basis for well structure programming. Languages such as Modula-2, Pascal and C all lend themselves to writing structured programs that compile to highly efficient executable code. Of the three programming languages, C is the most widely used in the engineering fraternity.

Attitude determination software for star sensors normally consists of five components [Wertz, 1986], as shown in Figure 2.3. The exact implementation of these components depends on the type of sensor, accuracy requirements, the accuracy to which the other sensors can determine the satellite attitude, the size of the FOV, the orientation of the sensor, its sensitivity and the complexity of the attitude model, to mention but a few.

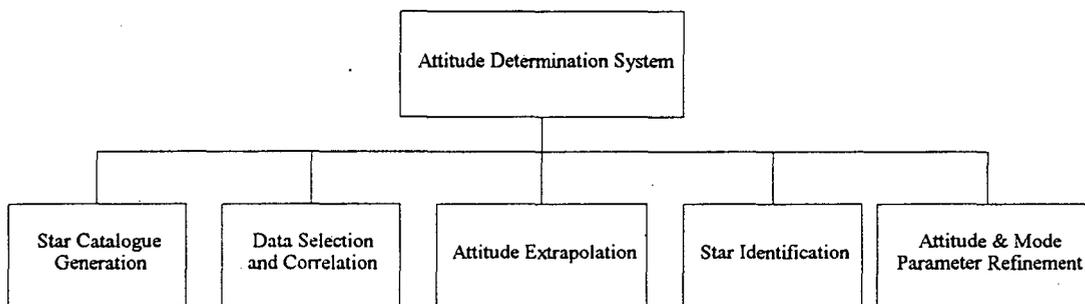


Fig. 2.3 Software components

Only details of stars necessary for the operation of the system are included in the onboard star catalogue [Anderson *et al.*, 1990]. This information can be updated periodically to compensate for the inertial shift in the orbit of the satellite around the earth. Such a sub-catalogue generally contains the Right Ascension (RA or  $\alpha$ )\*, the declination (dec or  $\delta$ )† and the visual magnitude ( $m_v$ ) of each guide star. A guide star, in this instance, is a star that conforms to certain criteria.

\* The abbreviation, RA, and symbol,  $\alpha$ , are used for the Right Ascension [Illingworth, 1994].

† The abbreviation, dec, and symbol,  $\delta$ , are used for the declination [Illingworth, 1994].

The star must

- be in the area of the celestial sphere where the FOV is expected to be;
- have a numeric visual magnitude below a certain threshold; and
- be located at an angular distance greater than a set threshold from its closest neighbours [Van Bezooijen, 1989, 1990].

To keep the access time of information in this catalogue to a minimum, it should be sorted according to one of the three parameter fields, for instance the RA. A sorted sub-catalogue is again divided into zones that, in cases where the orbit and general attitude of the satellite is well known, could be uploaded as certain sections of the celestial sphere are needed.

The most hardware-dependent of the five components is data selection and correction. The accuracy and quality of the sensor optics and electronics will determine the amount of discriminatory processing to be done on a raw FOV image. This software has to obtain the centroid and magnitude data of all the stellar images that are to be found in the FOV. In the case of a CCD sensor, a certain amount of digital processing has to be done on the image in order to obtain the positions of the stars. Magnitude and centroid data of each star in the FOV are passed to the attitude determination software as unit vectors in the FOV or spacecraft frame.

An estimate of the attitude at the time of the stellar sighting has to be available to the identification software. This is done by extrapolating an initial attitude using a model of the spacecraft motion [Wertz, 1986]. The higher the accuracy of the attitude estimate, the less processing is needed by the stellar identification software, and the less processing there has to be done, the less time is taken for the attitude determination. It is therefore important to have an accurate enough attitude model at hand in order to keep the processing time of a system with limited resources, such as the one described in this paper, to a minimum.

### 3. A Low Earth Orbit Microsatellite

A microsatellite presents a number of challenges when it comes to the development of an attitude determination system. Apart from meeting the required accuracy and environmental requirements, the components, such as the star camera, must conform to a certain weight and size and must operate from the limited power supply of the satellite. *A priori* knowledge of the satellite orbit and of system performance specifications is used to calculate the required performance of the star camera, the numeric processing power for the pattern recognition and the memory requirements for storing the camera image.

The first sections of this chapter present data and calculations for specifying the components of a microsatellite for a low earth orbit mission. In the last section a complete system design with possible implementations is given.

#### 3.1 Specifications and Requirements

The sensor system must comply with certain specifications and requirements. The most important of these [Milne, 1990] are listed below:

**Table 3.1 Specifications for the star sensor system**

Attribute	Specification
Size	100 x 50 x 50 mm (CCD element/lens) 300 x 100 x 20 mm (electronics)
Weight	200g (CCD element/lens) 50g (electronics)
Power	2 Watt continuous
Resolution	0.025° (1.5 arc minutes) - Roll and Yaw 0.286° (17.16 arc minutes) - Pitch
Update time	1 s

The sensor has to be a small, light-weight device with a relatively low accuracy by the standards listed in Table 3.1. As mentioned in the introduction, the purpose of this paper was to find ways of implementing the different components of a star sensor system using readily available technology and techniques in order to keep the cost as low as possible.

The attitude information provided by the star sensor is essential to the success of the mission. A certain level of hardware and software redundancy should therefore be provided to ensure correct operation of the system. Fault tolerance should also be implemented in the electronic and software design. The areas where fault tolerance is particularly important are the identification of stellar images (Chapter 4) and the recognition of star patterns (Chapter 5).

### 3.2 The Satellite Orbit

The particular microsatellite for which this study was done, is of the sun-synchronous, spinning, low earth orbit type [Rosengren, 1990]. It is intended for experiments in the fields of communication, science and remote sensing [Milne, 1990].

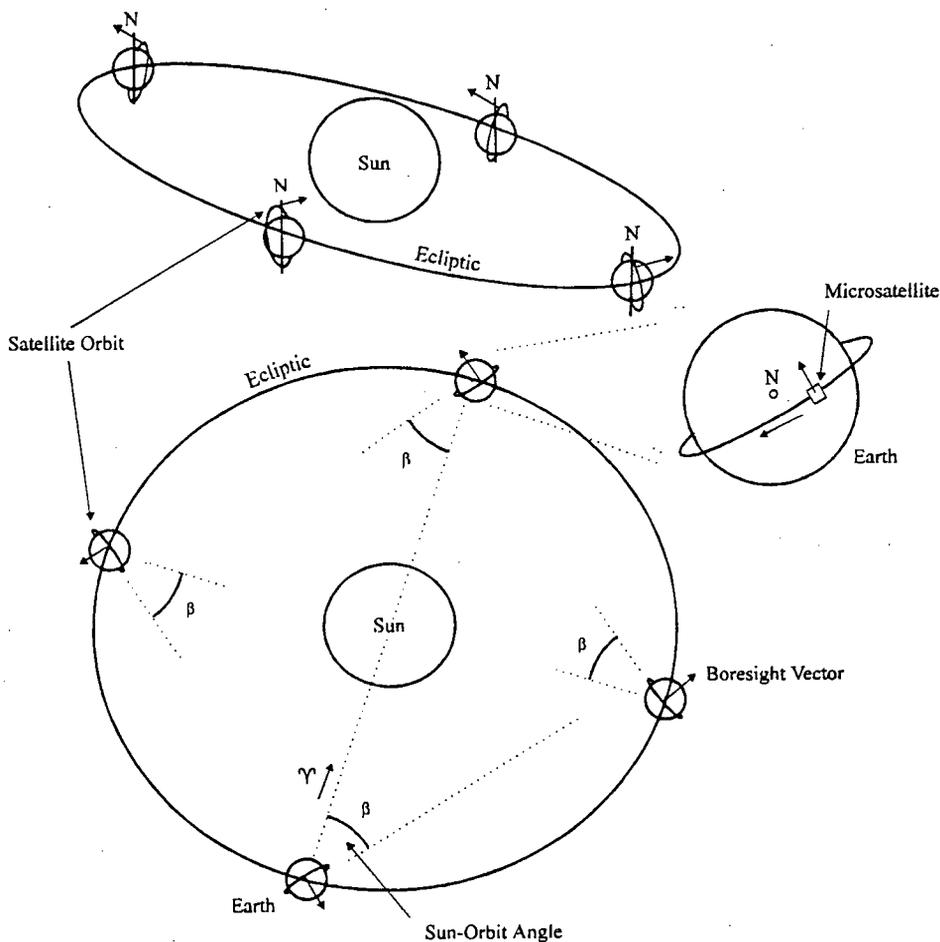


Fig. 3.1 The satellite orbit as viewed from the north celestial pole

An ideal orbit, shown in Figure 3.1, is chosen so that the lighting conditions when the satellite passes over the sunlit side of the earth are optimal for remote sensing. The orbit is circular and has an inclination angle,  $i$ , of  $98.6^\circ$  at an altitude of 800 km [Davidoff, 1987].

**Table 3.2 Orbital parameters**

Parameter	Description	Value
$a$	semi-major axis	$6378 + 800 = 7178$ km
$h$	altitude	800 km
$e$	eccentricity	0 for circular orbit
$u^*$	true angular nodal elongation	variable
$i$	inclination of equatorial plane	$98.6^\circ$
$\Omega$	longitude of ascending node	variable ( $155^\circ$ at $\mathcal{V}$ )
$T$	period	100 minutes

Due to the design and purpose of the microsatellite, the attitude information obtained from the sun and horizon sensors will be adequate most of the time. It is only when more accurate attitude information for better stability and spacecraft orientation is required that the star sensor will be used. One of these instances occurs during remote sensing. Before each remote sensing exercise, the spin of the satellite is stopped. The pushbroom CCD imager then takes a number of pictures of the surface of the earth. During this operation the star sensor system will be used to add the extra attitude accuracy needed for the stabilisation of the satellite for the high resolution imagery.

The local level coordinates of the satellite are as depicted in Figure 3.2. The sensor will be mounted on the top facet of the satellite, pointing in the orbit normal ( $+Y_{LL}$ ) direction when the satellite is not spinning. When so orientated, the camera will point to a certain area of the celestial sphere. The path is calculated by transforming the unit vector of the boresight of the camera through the coordinate systems as shown in Appendix A. A star's position in the focal plane of the star sensor can be found by a non-linear mapping from inertial celestial coordinates to image focal plane coordinates.

The boresight vector in camera coordinates is:

$$r_{POV} = \begin{bmatrix} x_{POV} \\ y_{POV} \\ z_{POV} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad (3.1)$$

\* The true angular elongation is preferred for a circular orbit [Wertz, 1986].

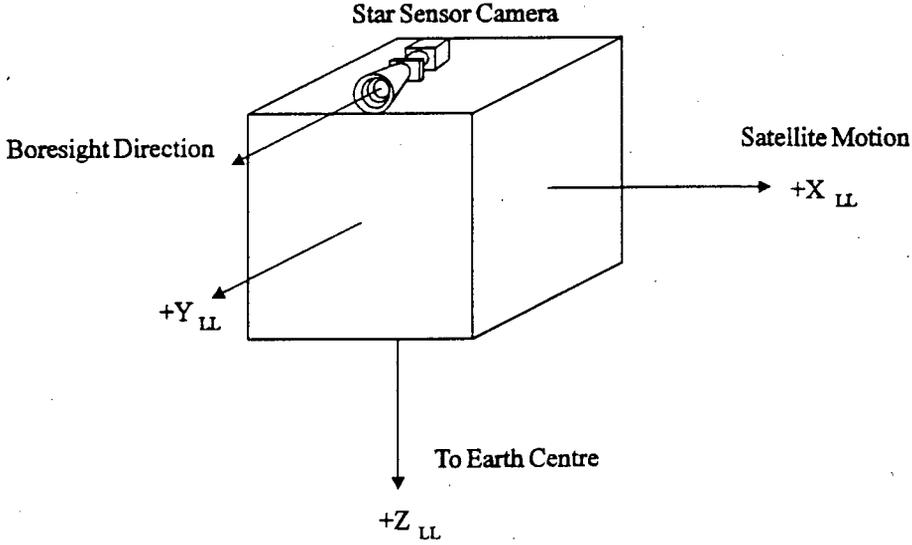


Fig. 3.2 The local level coordinates

Transformation from *FOV* to *Local Level* to *Earth* coordinates is done by :

$$F_E = M_{Total} \cdot F_{FOV} \quad (3.2)$$

where the transformation matrix (Appendix A),  $M_{Total}$ , is given by :

$$M_{Total} = \begin{bmatrix} -\cos\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) + \sin\left(u+\frac{\pi}{2}\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(\frac{\pi}{2}-i\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) & -\sin\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) - \cos\left(u+\frac{\pi}{2}\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) \\ \cos\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) + \sin\left(u+\frac{\pi}{2}\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) & -\sin\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) - \cos\left(u+\frac{\pi}{2}\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) \\ -\cos\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) & -\cos\left(\frac{\pi}{2}-i\right) & -\sin\left(u+\frac{\pi}{2}\right) \sin\left(\frac{\pi}{2}-i\right) \end{bmatrix} \quad (3.3)$$

The complete transformation to RA and declination is given by (Appendix B) :

$$RA = \text{atan} \left( \frac{xFOV \cdot \left( -\cos\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) + \sin\left(u+\frac{\pi}{2}\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) \right) + yFOV \cdot \left( \sin\left(\frac{\pi}{2}-i\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) \right) + zFOV \cdot \left( -\sin\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) - \cos\left(u+\frac{\pi}{2}\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) \right)}{xFOV \cdot \left( \cos\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) + \sin\left(u+\frac{\pi}{2}\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) \right) + yFOV \cdot \left( -\sin\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) \right) + zFOV \cdot \left( \sin\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) - \cos\left(u+\frac{\pi}{2}\right) \cos\left(\beta+\frac{\pi}{2}-\Omega\right) \right)} \right) \quad (3.4)$$

$$dec = \left( xFOV \cdot \left( -\cos\left(u+\frac{\pi}{2}\right) \cos\left(\frac{\pi}{2}-i\right) \sin\left(\beta+\frac{\pi}{2}-\Omega\right) \right) + yFOV \cdot \left( -\cos\left(\frac{\pi}{2}-i\right) \right) + zFOV \cdot \left( -\sin\left(u+\frac{\pi}{2}\right) \sin\left(\frac{\pi}{2}-i\right) \right) \right) \quad (3.5)$$

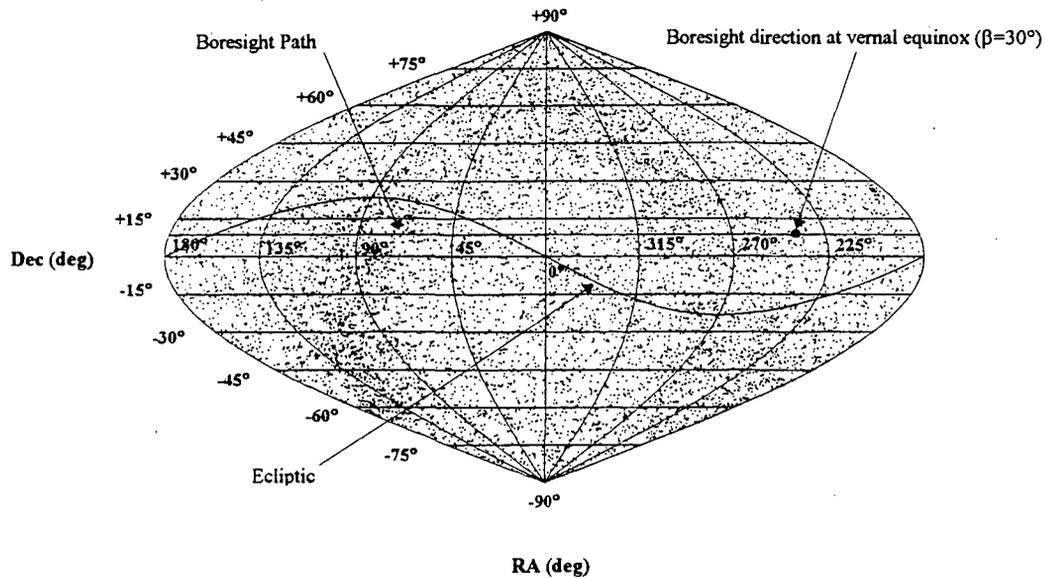


Fig. 3.3 The path of the camera boresight on the celestial sphere\*

During one orbit, the boresight vector of the star sensor camera does a complete revolution as the satellite Local Level axis system turns once around its  $+Y_{LL}$  axis. This motion causes the FOV to rotate through  $360^\circ$  with the boresight centred at a particular RA and declination. Movement of the FOV on the celestial sphere is caused by the orbit of the Earth around the sun. The plane of this path is parallel to the equator of the earth (and therefore parallel to the celestial equator) and the boresight direction is  $(90+\beta)^\circ$  ( $\beta$  is the sun angle of the orbit and the boresight points in the  $+Y_{LL}$  direction) behind that of the sun (which is shown at  $0^\circ$  at vernal equinox) as shown in Figure 3.3. The boresight vector has a fixed declination of  $+8.6^\circ$  as the earth rotates around the sun. From the figure can be seen that the boresight direction crosses the densely populated galactic plane at  $90^\circ$  and  $270^\circ$ . Any offset from the  $+Y_{LL}$  axis will cause the boresight path to form circles around a declination of  $8.6^\circ$  as the satellite rotates around its earth orbit.

\* Sanson-Flamsteed projection [McDonnel, 1979]

### 3.3 Stellar Data Required for Attitude Determination

Accurate information on all stars that will be visible to the sensor camera is needed for the star identification process. These parameters are the RA, dec and magnitude for each star. A major constraint on the system design is that a certain number of stars must be visible in the camera FOV at any instant [Van Bezooijen, 1989]. In this section the stellar brightness detection limit of the star sensor camera and the dimensions of the FOV are calculated.

#### 3.3.1 Star Catalogues

Many sources of stellar data exist on magnetic media. The use of one above the other depends on the mission requirements and resources. In this case the main requirement is that attitude information should be available during any part of the orbit at an update time of 1 second. Memory storage capacity and processor power are limited which suggests the use of the minimum amount of stellar data. Four examples of star catalogues are [Van Bezooijen, 1989; Wertz, 1986]:

- Catalogue of Bright Stars [Hoffleit, 1964]
- Smithsonian Astrophysical Observatory Catalogue (SAO)
- SKYMAP version 3.3 star catalogue [McLaughlin, 1989]
- Yale Star Catalogue

These catalogues vary in completeness and magnitude range. The SAO catalogue, for instance, provides positional accuracy to 1 arcsec at epoch 2000. Adjustments can be made to the stellar positions in the catalogue by taking the exact time of observation into account but should only be taken into account for very accurate measurements. Proper motion which is generally less than 10 arc seconds per century for 95 % of the stars brighter than ninth magnitude and aberration caused by the motion of the earth around the sun which attains a maximum of 20 arcsec. are both ignored for this application. For the purpose of this star sensor system far less accurate stellar positions are required. (The highest accuracy is 1.5 arcmin. for the roll and yaw axes.)

The fourth catalogue mentioned above, viz. the Yale Bright Star Catalogue, was chosen for this application. It gives the RA and dec of each star to an accuracy of 0.1 and 1 second respectively. Each star's visual magnitude is also given to an accuracy of 0.01 magnitude.

For this application, a core catalogue consisting of a number of sub-catalogues or zones was generated. The core catalogue consists of "guide stars", i.e. only those stars in the Yale catalogue that conform to certain criteria. The creation of such a core catalogue is discussed further in Chapter 5.

### 3.3.2 Stellar Distribution Density

In order for the pattern recognition algorithms, considered in Chapter 5, to be successful, at least 3 stars have to be visible at all times. The camera sensitivity defined in terms of the stellar brightness limit and FOV size must therefore be such that 3 stars are always visible. Star brightness as measured in visual magnitude is defined by [Shu, 1982] :

$$m_v = -2.5 \log(F) + m_0 \quad (3.6)$$

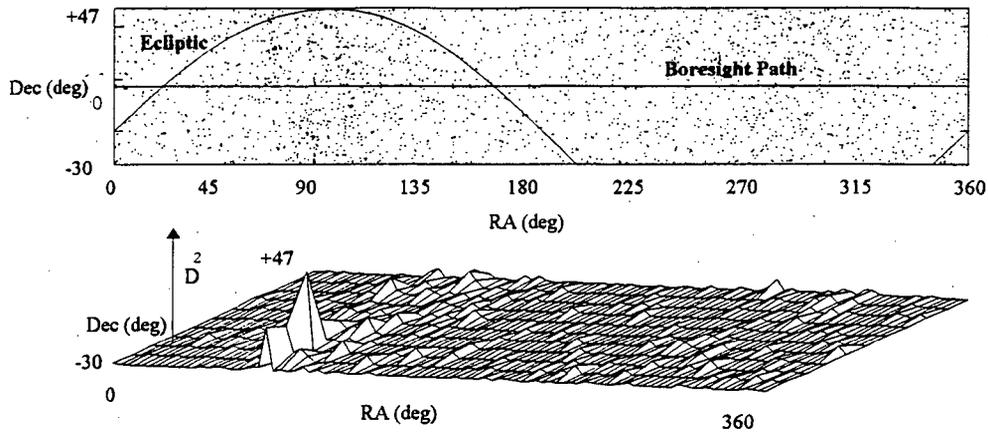
where  $m_0 = -13.94188$

and  $F$  (measured in terms of the scotopic response curve of the human eye) is the luminous flux density of the star, measured in lux.

Table 3.3 Stars and Magnitudes [Wertz, 1986]

Limiting Visual Magnitude ( $m_v$ )	Number of Stars
3.0	187
4.0	556
5.0	1660
6.0	5146
7.0	15095
8.0	44700

The relationship between visual magnitude and number of stars visible in the celestial sphere is non-linear as shown by Table 3.3. The higher the magnitude specification for the system, the more stored data is required which would slow down the recognition algorithms. In general, the processing power of the system would place an upper limit on the highest detectable magnitude star. In a microsatellite system, however, weight and size limitations on the optics are such that an upper limit for  $m_v$  is determined by the optics rather than by the processor.



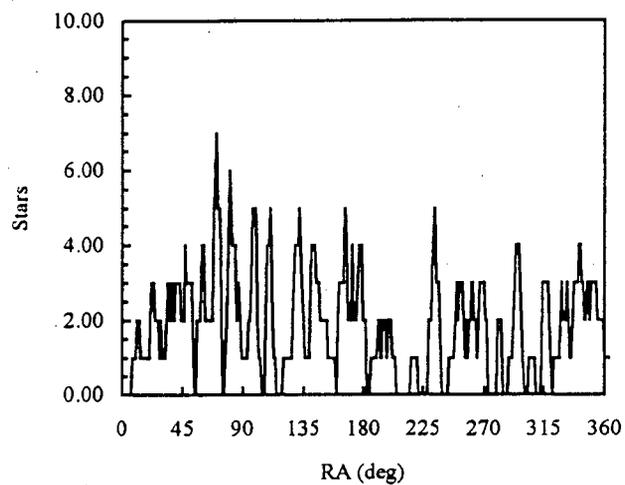
**Fig. 3.4 Stars and stellar distribution density along the boresight path**

Stellar distribution density is the key parameter for determining the magnitude limit for a particular FOV size. In the areas of the celestial sphere where the Milky Way is crossed by the boresight path, the galaxy is viewed edge-on [Shu, 1982] and consequently the distribution of stars is high. The remaining areas, on the other hand, are sparsely populated.

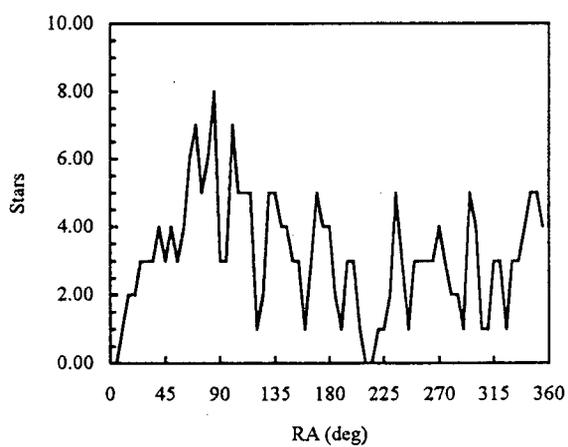
Calculation of the stellar distribution density was done for each rectangular  $5^\circ$  by  $5^\circ$  area of the celestial sphere for all the stars in the Yale Bright Star Catalogue. Figure 3.4 shows the stars and a 3-D contour plot of the stellar distribution for the area of the celestial sphere in the vicinity of the boresight path (The square of the stellar distribution density,  $D^2$ , was used to obtain a better 3-D graph). It can be seen that in the area of probable pointing direction, the sections between  $320^\circ$  and  $75^\circ$  and between  $150^\circ$  and  $210^\circ$  are the most sparsely populated. These areas are opposite the densely populated areas around  $90^\circ$  and  $270^\circ$  on the celestial sphere which corresponds to the disk shape of the galaxy and the fact that it is viewed from the inside.

More calculations were done to obtain a more accurate measure of the distribution density. The number of stars in a  $10^\circ$  by  $10^\circ$  (and  $12^\circ$  by  $12^\circ$ ) area were calculated as the window was moved along the boresight path at  $1^\circ$  and  $5^\circ$  intervals. Figures 3.5 to 3.7 show the number of stars per square degree for this area around the boresight direction for visual magnitudes 5.0, 6.0 and 7.0.

The low density regions can again be seen close to  $0^\circ$  and  $180^\circ$  where the boresight path moves away from the direction of the galactic plane. Figure 3.5 shows that the minimum stellar distribution density for  $m_v = 5.0$  is 0 at many points along the boresight path. A magnitude limit value between 6.0 and 7.0 as shown by figures 3.6 and 3.7 will ensure that there are at least 3 stars in a  $10^\circ \times 10^\circ$  FOV.



**Fig. 3.5 (a) Stars along the boresight path ( $m_v=5.0$ , resolution =  $1^\circ$ )**



**Fig. 3.5 (b) Stars along the boresight path ( $m_v=5.0$ , resolution =  $5^\circ$ )**

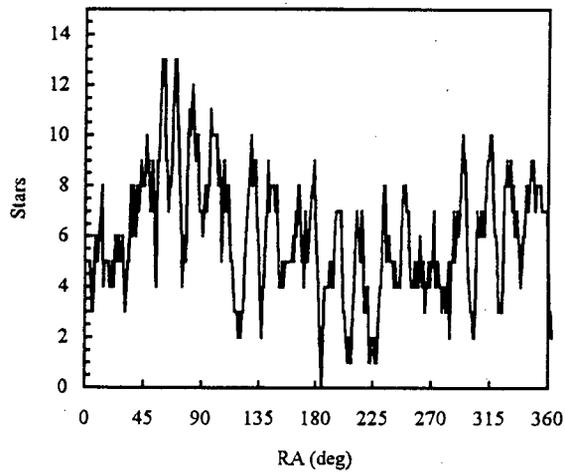


Fig. 3.6 (a) Stars along the boresight path ( $m_v = 6.0$ , resolution =  $1^\circ$ )

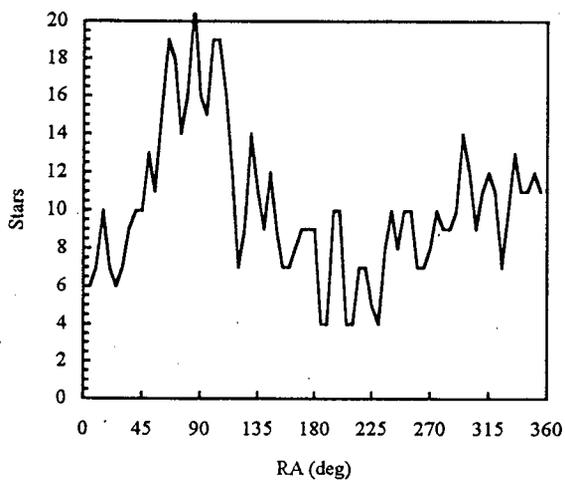
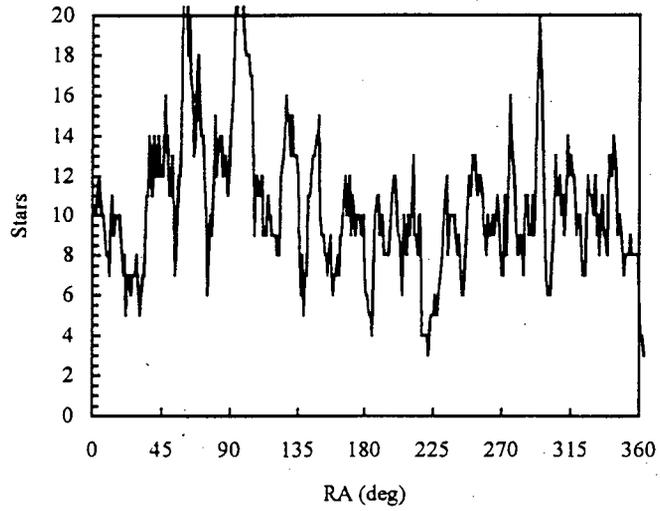
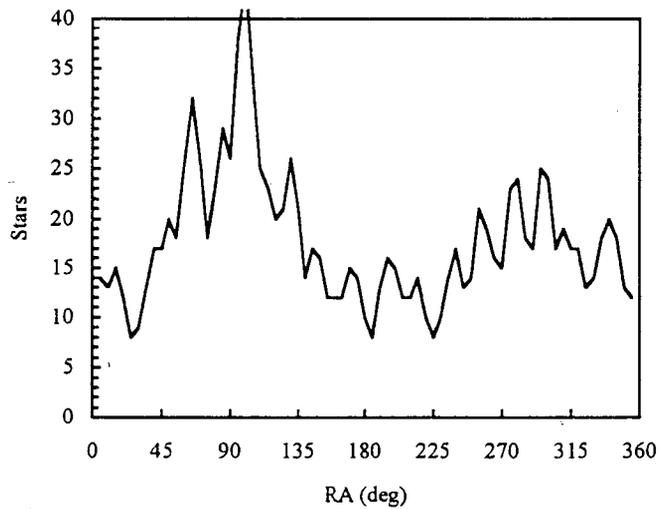


Fig. 3.6 (b) Stars along the boresight path ( $m_v = 6.0$ , resolution =  $5^\circ$ )



**Fig. 3.7 (a) Stars along the boresight path ( $m_v = 7.0$ , resolution =  $1^\circ$ )**



**Fig. 3.7 (b) Stars along the boresight path ( $m_v = 7.0$ , resolution =  $5^\circ$ )**

### 3.3.3 FOV Size

The shape and size of the FOV depends on a number of parameters. Commercially available area CCDs have rectangular dimensions. A rectangular FOV will therefore be adopted.

Parameters that determine the FOV size are the pixel dimensions (size and XY-count) of the CCD, the  $f/\#$  of the lens to be used, the magnitude limit imposed by the complete system and the required accuracy that is to be attained.

Because there is always the chance that a stellar image can span two or more pixels, star sensor systems use defocused stellar images (an image spread over several pixels) and interpolation techniques to calculate the stellar centroids to the required accuracy. Defocusing causes loss in individual pixel signal power which then requires a larger lens. A star's image is therefore defocused onto a grid of no more than 25 pixels (5 by 5; Chapter 4) to limit the loss in signal power.

In the previous paragraph it was shown that stellar distribution density varies greatly over the probable camera boresight directions in the orbit. The criterion of a minimum number of stars in the FOV has to be met in the areas of lowest stellar distribution density. Figure 3.8 is a plot of the lowest distribution densities of Figures 3.5 to 3.7 for visual magnitudes 5 to 7.5 for FOVs of  $10^\circ \times 10^\circ$  and  $12^\circ \times 12^\circ$  sizes. The departure from the logarithmic curve for stars above a visual magnitude of 6.5 is due to the incompleteness of the catalogue for these fainter stars.

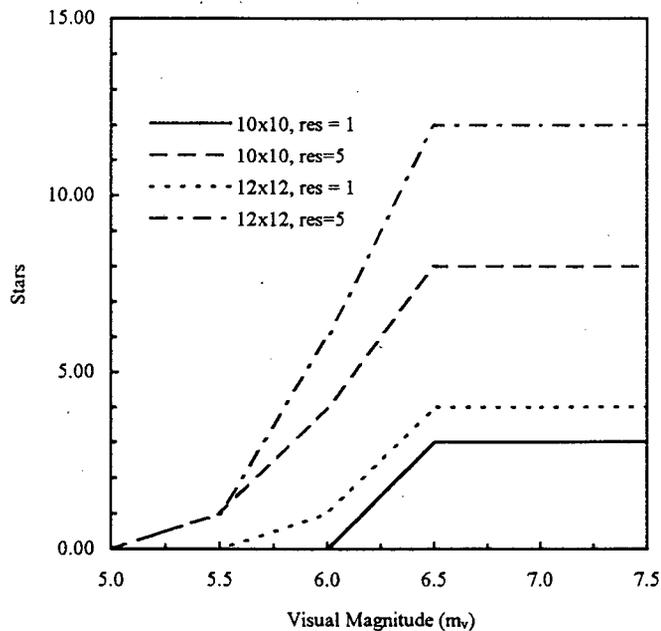


Fig. 3.8 Minimum distribution density

The upper limit to the FOV size is imposed mainly by the processing power of the AD&CS and the characteristics of the camera lens while the lower limit is set by the minimum number of visible stars. From Table 3.1 the specified attitude update time is 1 second which includes integration and processing time. In order to keep the processing time to a minimum, the lower limit for the FOV size is preferred to the upper.

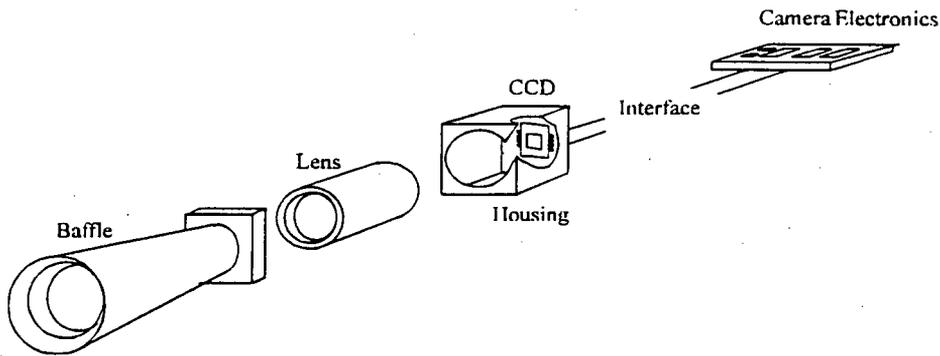
Using the lower limit as guide a value of  $10^\circ$  by  $10^\circ$  for the FOV dimensions was chosen. Whether this FOV size is practical will also depend on the lens characteristics namely the f-ratio and angle of view.

### 3.4 Star Camera Calculations

For this application a mapper type of star sensor camera has been chosen. The mapper camera has to have a large FOV (100 square degrees for the Honeywell mapper in Table 2.2), high sensitivity (a low f-ratio) and very low spatial distortion. These requirements place high demands on the optical system. CCD technology provides the required light sensitivity with the added benefits of light weight and small size.

Commercially available CCD cameras support the popular NTSC or PAL video standards [EEV, 1987, 1990]. These cameras scan the FOV continually at a fixed frame rate. A CCD star camera, on the other hand, has to operate like a photographic camera in the sense that a picture (frame) is recorded when necessary and that the camera must be in low power, standby mode at all other times. Another criterion for the sensor camera is that it must be possible to vary the integrating time according to the viewing conditions (the proximity of the sun and moon to the FOV) and the brightness of stars in the FOV. A CCD camera prototype was developed using the TC211 CCD from Texas Instruments (Chapter 6) in order to test the viability of such a low-cost CCD camera.

The main components of the camera are illustrated in Figure 3.8. These components are: a baffle, a lens, CCD in a housing and the camera electronics. As shown in Figure 3.5, the baffle, lens and CCD with housing will be located on the  $-Z_{LL}$  facet of the satellite, while the camera electronics will be positioned inside the satellite body. The components on the outside of the satellite will be subjected to the harsh physical conditions of outer space. Temperature variations of as much as  $150^\circ$  ( $-80^\circ\text{C}$  to  $70^\circ\text{C}$ ) [4th Conference on Small Satellites, 1990] may be encountered during a single orbit of approximately 100 minutes. The outside surfaces of the three external components of the camera therefore have to be coated with a highly reflective material to keep the CCD and lens at a low temperature. On the other hand, the inside surfaces of these components have to be non-reflective in order to keep the stray illumination on the CCD as low as possible.



**Fig. 3.9 Components of the CCD Camera**

### 3. 4. 1 CCD Responsivity

The output voltage of a pixel of a CCD is determined by the number of electrons that are formed in that pixel during the periods of integration and clockout. This relationship can be expressed as [EEV, 1987]:

$$V_o = \frac{qN_e G}{C_o} \quad [\text{V}] \quad (3.7)$$

where

$q$  = electron charge (1.6e-19 C)

$G$  = output amplifier gain (typically 0.7 for source follower MOS transistor)

$C_o$  = output capacitance (typically 0.1 pF)

$N_e$  = Number of electrons

Electrons are formed by two processes, electromagnetic radiation that penetrates the silicon of the pixel during integration, and noise :

$$N_e^- = N_{e^- \text{ Radiation}} + N_{e^- \text{ Noise}} \quad (3.8)$$

Incident electromagnetic radiation (photons) on a CCD causes electrons to form in each pixel according to the amount of radiation and the quantum efficiency of the silicon. The TC211 CCD that was used for the prototype converts electrons to an output voltage at a rate of 1.4  $\mu\text{V}/e^-$ . The quantum efficiency (number of photons of light needed to excite electrons) of this CCD is shown in Figure 3.10 to have a peak value at a wavelength of about 0.75 micrometers.

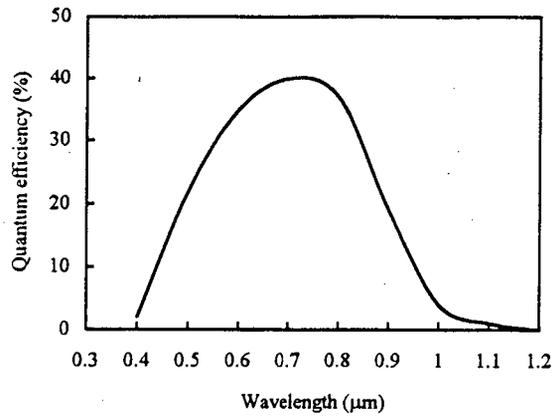


Fig. 3.10 Quantum efficiency of a CCD [EEV, 1987]

Generally CCDs have responses that are shifted more (in relation to the response of the human eye which is centred at 555 nm) to the infrared region [Thomson, 1974; Texas Instruments 1987] of the electromagnetic spectrum. A representative frequency response of a CCD sensor, the TC211 from Texas Instruments (Figure 3.11) is shown to have its peak at a wavelength of about 700 nm.

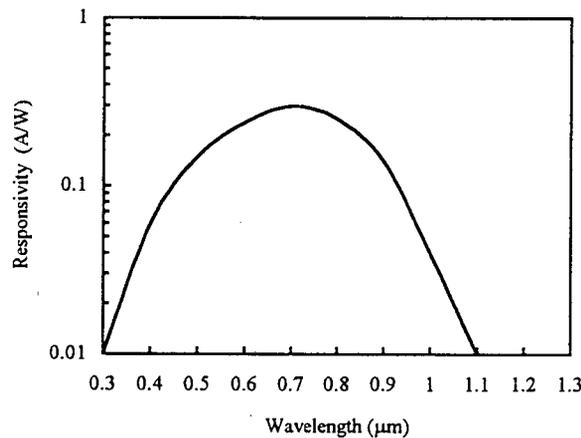


Fig. 3.11 Frequency response of the TC211 [Texas Instruments, 1987]

The number of electrons generated in a CCD [Glass, 1994] per square meter per second, by a star is :

$$N_{e^- \text{ radiation}} = \int_0^{\infty} \frac{E_{e\lambda}}{h\nu} \cdot \eta_{\lambda} \cdot d\lambda \quad (3.9)$$

where

$E_{e\lambda}$  = spectral irradiance of the star

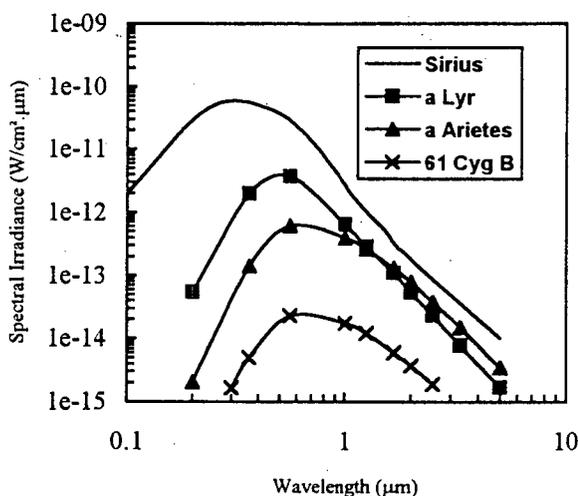
$h\nu$  = energy in a photon

$\eta_{\lambda}$  = quantum efficiency of the CCD

Spectral irradiance curves of individual stars vary greatly which makes the accurate determination of CCD response to starlight impossible unless the response to each spectral type of star is calculated individually. A compromise must be made on the accuracy of measurements by choosing a representative spectral star type for stellar magnitude calculation. Figure 3.12 shows the spectral irradiance curves for a number of stars with different stellar spectral types and magnitudes as tabulated below.

**Table 3.4 Stars and Magnitudes [Shu, 1990]**

Star	Spectral Type	Visual Magnitude
Sirius (a CMaj)	A1V	-1.47
a Lyr	A0V	0.03
a Arietes	K2III	1.99
61 Cyg B	K7V	6.02



**Fig. 3.12 Calculated irradiances of some stars [Hecht, 1987]**

However, as shown in Figures 3.10 to 3.12, the spectral responses of stars fall within the response wavelengths of the CCD. A CCD is therefore very well suited for the detection of stellar radiation in the visual to infrared area of the electromagnetic spectrum. Which stars a particular CCD sensor will actually detect also depend further on the integration time and the lens aperture of the camera.

### 3. 4. 2 Sources of Noise

In order to achieve greater accuracy in the choice of a minimum detection voltage, the noise sources of the CCD and their contributions to the output are taken into account.

The main sources of noise are [EEV, 1987; Thomson 1974] :

- Photonic noise
- Dark Current
- Dark Signal Non-Uniformity (DSNU)
- Reset noise

The first of these, photonic noise, is caused by the corpuscular nature of photons and is equal to,

$$N_{e^{-}ph} = \sqrt{N_{e^{-}Radiation}} \quad (3.10)$$

where  $N_e$  is the number of electrons formed in a photo site. This type of noise dominates at low radiation intensity levels.

Dark current is formed by the thermal creation of electrons in the CCD's silicon.

$$I_D = Ae^{\frac{-V_{BG}}{q}} \quad (3.11)$$

where

$k$  = Boltzman's constant

$T$  = temperature in Kelvin

$V_{BG}$  = voltage of a silicon diode

$q$  = electron charge

$A$  = constant for the particular CCD

This formula is valid for the temperature range  $-60$  °C to  $75$  °C and shows that the dark signal doubles for every  $10$  °C temperature rise above  $-15$  °C. It is highly dependent on temperature, as well as on time, and follows the diode law. Device cooling is therefore needed when longer integration periods are used. For the Texas Instruments range of TCXX CCDs the maximum dark signal at room temperature ( $25$  °C) is  $15$  mV.

Dark Signal Non-Uniformity (DSNU), the peak-to-peak difference between pixel output voltages, follows the same law as the Dark Current (the TC211 CCD has a maximum DSNU of  $15$  mV at  $25$  °C).

Charging a diode to its reference potential causes reset noise which has a value of :

$$N_{e^{-} \text{ reset}} = \frac{\sqrt{kTC_L}}{q} \quad (3.12)$$

where

$C_L$  = readout capacitance

Equation 3.12 can be written as  $400\sqrt{C_L}$  at room temperature (25 °C).

The total noise on the output signal is therefore :

$$N_{e^{-} \text{ Noise}} = N_{e^{-} \text{ ph}} + N_{e^{-} \text{ DS}} + N_{e^{-} \text{ DSNU}} + N_{e^{-} \text{ reset}} \quad (3.13)$$

Typical values [Thomson, 1974] for these noise signals at 25 °C and a saturation voltage are :

$$\begin{aligned} N_{e^{-} \text{ Noise}} &= 1000 + 100 + 100 + 110 \\ &= 1310 \text{ electrons} \end{aligned}$$

with the conditions that

$$\begin{aligned} N_{e^{-} \text{ Radiation}} &= 1\,000\,000 \\ C_L &= 0.08 \text{ pF} \end{aligned}$$

Using a value of 1310 for the number of electrons,  $N$ , in Equation (3.7) gives an output voltage of nearly 1.5 mV. This gives a signal-to-noise-ratio of:

(3.14)

$$\begin{aligned} SNR_{dB} &= 20 \log \left( \frac{V_{out \text{ max}}}{|V_{noise}|} \right) \\ &= 20 \log \left( \frac{450 \text{ mV}}{1.5 \text{ mV}} \right) \\ &= 49.54 \text{ dB} \end{aligned}$$

at a temperature of 25 °C. This value can of course be lowered by cooling the CCD by means of a solid state peltier cooler.

Another source of electron hole pairs is ionising radiation from cosmic rays. An incident muon can generate a signal of ~2000 electrons [EEV, 1987]. Extreme levels of radiation can damage the CCD [EEV, 1987] which results in an increase in the dark current and a reduction of charge transfer efficiency. Levels of less than  $10^4$  units are generally acceptable, but levels in the region of  $10^6$  can permanently damage the device.

### 3. 4. 3 The Camera Lens

Choosing the correct lens for the camera is crucial to the success of the sensor. Apart from optical characteristics, the main design criteria are weight and size. Both these parameters have to be kept to a minimum because of the launch cost per kilogram and space limitations in the launch vehicle respectively. The type of glass that the lens is made of should be light with a low heat coefficient. The latter prevents distortions of the lens (and image) during the transitions from very high to very low temperatures in orbit. The size (diameter) of the lens is then the limiting design factor.

In order to make use of interpolation algorithms that give sub-pixel positional accuracy [Grossman, 1984], the image of the faintest star should be spread over not less than 4 pixels (2 by 2 grid;  $N_{Pixel} = 4$ ). The lens therefore has to be large enough to detect the radiation from this star spread over the four pixels.

The aperture diameter of the of the lens has the area :

$$A_{lens} = \pi \left( \frac{D}{2} \right)^2 \quad (3.15)$$

One pixel on the CCD has an area of :

$$A_{Pixel} = L_x \cdot L_y \quad (3.16)$$

Combining Equations (3.7), (3.12), (3.13), (3.15) and (3.16) gives the total number of electrons created in the area of the CCD where the stellar image is formed.

$$\begin{aligned} N_{e^- \text{ radiation}} &= \int_0^{\infty} \frac{E_{e\lambda} \left( \frac{A_{Lens}}{A_{Pixel} \cdot N_{Pixel}} \right)}{h\nu} \cdot \eta_{\lambda} \cdot d\lambda \\ &= \int_0^{\infty} \frac{E_{e\lambda} \left( \frac{\pi \cdot \left( \frac{D}{2} \right)^2}{(L_x \cdot L_y) \cdot (N_{Pixel})} \right)}{h\nu} \cdot \eta_{\lambda} \cdot d\lambda \quad (3.17) \end{aligned}$$

The output voltage for a particular star is then (equations 3.8, 3.13 and 3.16) :

$$V_0 = \frac{q \cdot \left( \int_0^\infty \frac{E_{e\lambda} \left( \frac{\pi \cdot \left(\frac{D}{2}\right)^2}{(L_x \cdot L_y) \cdot (N_{Pixel})} \right)}{h\nu} \cdot \eta_\lambda \cdot d\lambda + N e^{-noise} \cdot G \right)}{C_0} \quad (3.18)$$

A Matlab simulation was carried out using Equation (3.18) over a range of lens areas and integration times for the brightest ( $m_v = -1.47$ ) and faintest star ( $m_v = 6.5$ ) that has to be detected using the stellar spectral type A0V<sup>†</sup> [Illingworth, 1994; Glass, 1994]. The spectral response curve of the TC211 CCD was used in conjunction with square pixel dimensions of 10 $\mu$ m by 10 $\mu$ m. Noise was not added to the values plotted in Figures 3.13 and 3.14 in order to show the response from stellar radiation only.

The lens diameter is limited by the size of the microsatellite which is typically not more than 0.125 cubic meters in size. Integration time,  $t_i$ , is limited by the maximum star rate of the image.

Absolute maxima of the lens diameter and integration time are :

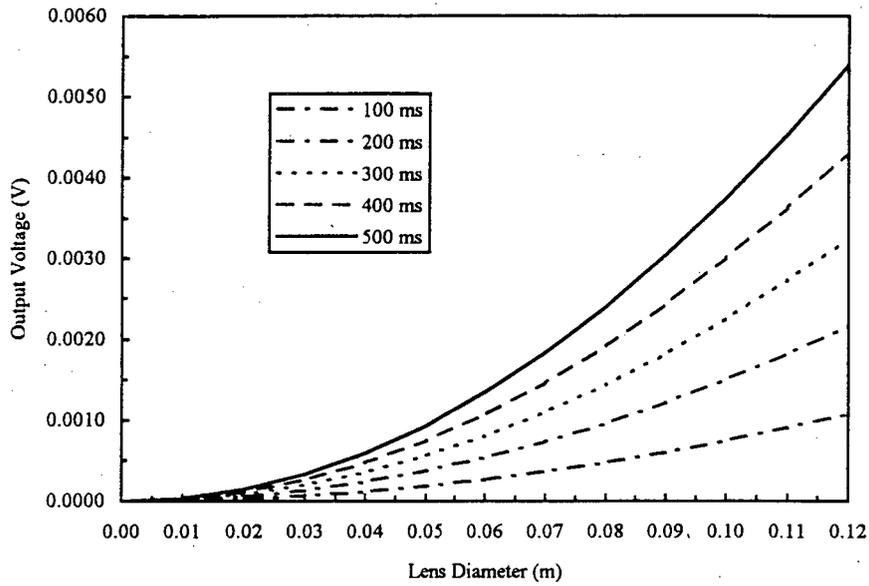
$$D_{lens}(\max) = 0.12 \text{ m}$$

$$t_i(\max) = 0.5 \text{ s}$$

Movement of the image will cause smearing of the stellar images which in turn will cause the centroid determination accuracy to drop (Chapter 4). A further requirement for short integration times is the AD&C system's need for an attitude update every second. If the integration time of the camera is low, more time is available for processing of the data. The electromagnetic radiation from the star was assumed to be spread evenly over 25 pixels (a grid of 5 by 5).

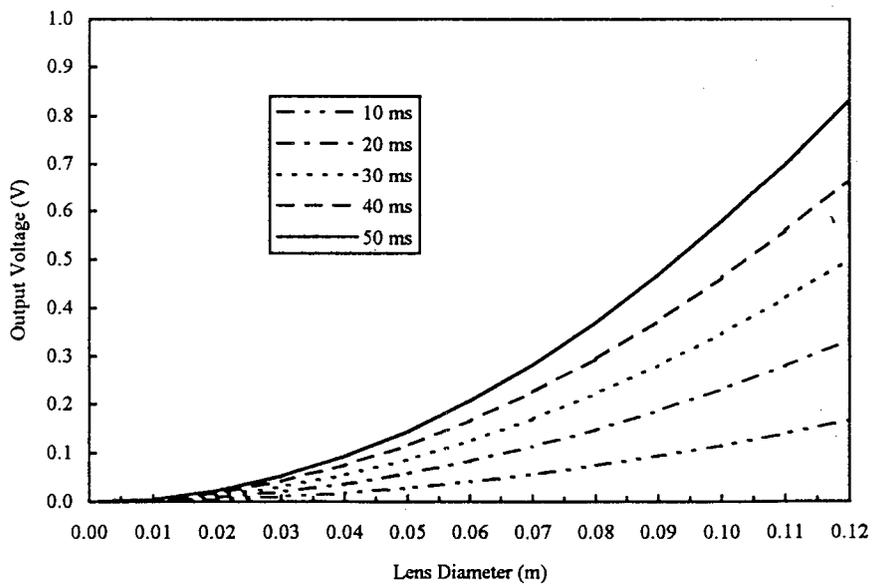
---

<sup>†</sup> Type A0V stars are in the main sequence on the Hertzsprung-Russel [Shu, 1982] diagram which contains 90% of the stellar population.



**Fig. 3.13 Output voltage vs. Lens Diameter vs. Integration time ( $m_v = 6.5$ )**

The graphs for  $m_v = -1.47$  are plotted to accommodate the CCD saturation voltage value of the TC211 CCD ( $V_{sat} = 450$  mV, with antiblooming enabled).



**Fig. 3.14 Output voltage vs. Lens Diameter vs. Integration time ( $m_v = -1.47$ )**

Figure 3.13 shows that the output voltage generated by a star with  $m_v = 6.5$  is 5.5 mV for an integration time of 0.5 sec. This value is quite low in the 450 mV range of the TC211. If 8-bit A/D conversion is used this would represent a value of

$$2^8 \left( \frac{0.0055}{0.450} \right) \cong 3$$

on a scale of 0 to 255. At 0.5 sec the magnitude -1.47 star, on the other hand, would be completely saturated and its image will be of no use to the centroid finding software (Chapter 4).

Using Equation (3.6), the spectral irradiance of a star is:

$$F_{m_v} = 2,65 \cdot 10^{-6} (\sqrt[3]{100})^{-m_v} \quad (3.18)$$

The required dynamic range (brightest to faintest star) of the detector of the star camera has to be:

$$N_D(\text{dB}) = 20 \log((\sqrt[3]{100})^{-(m_{v1}-m_{v2})}) \quad (3.19)$$

where

$$m_{v1} = -1.47$$

$$m_{v2} = 6.5$$

This equates to a dynamic range of 1541.7 or 63.76 dB which, although high, can be met by most of the commercial CCDs. In order to have the pixels of a magnitude 6.5 star register a values in excess of 5 when a star of magnitude -1.47 is also in the FOV the number of A/D bits,  $B$ , has to be at least:

$$2^B = 1541.7(5) = 7708.5$$

$$\Rightarrow B \geq 13$$

Noise voltage contributions as calculated using the model in section 3.4.2 will be in the order of a couple of millivolts if device cooling is used to keep the dark signal contribution as low as possible. The calculations done in this chapter show that the TC211 CCD is lacking in size in terms of the FOV and device sensitivity in terms of the detection of faint stars. Many other CCDs are available and should be studied to find the optimum sensor device.

Exact integration times for the CCD camera can only be found by extensive testing on the final camera design, which is not within the scope of this text.

### 3. 4. 4 Stellar Magnitude Calculation

For the purpose of the pattern recognition algorithms, it is necessary to calculate the visual magnitude of stars [Wertz, 1986]. In the previous section the response of a CCD to starlight was calculated for a certain type of stellar spectra. It was shown from Equation (3.9) that the CCD voltage output is strongly dependent on a star's spectral irradiance. For accurate magnitude calculations from CCD data, the response curve of the particular star should be used in the calculation. However, this is not possible because the star's magnitude is required as a parameter for the pattern recognition algorithm which has yet to identify the star. Another obstacle in magnitude calculation is the integrated intensity of faint background stars which will add a bias to the expected  $m_v$ .

One possible solution would be to use a visual spectrum filter on the camera lens but in this case it is not feasible because of the limited bandwidth and performance of the optical system. Another way to determine the visual magnitude of an observed star is to find a relation between visual magnitude and output voltage of the CCD (for a given lens size and integration time) for a star with an average (in terms of wavelength) spectral irradiance curve. The total intensity of the digitised stellar image can then be used in conjunction with this factor to obtain an estimate of the object's visual magnitude.

Combining Equations (3.17) and (3.18) gives the relation between visual magnitude and output voltage.

Figure 3.15 shows a graph of the output voltage versus visual magnitude for a camera system with the following characteristics :

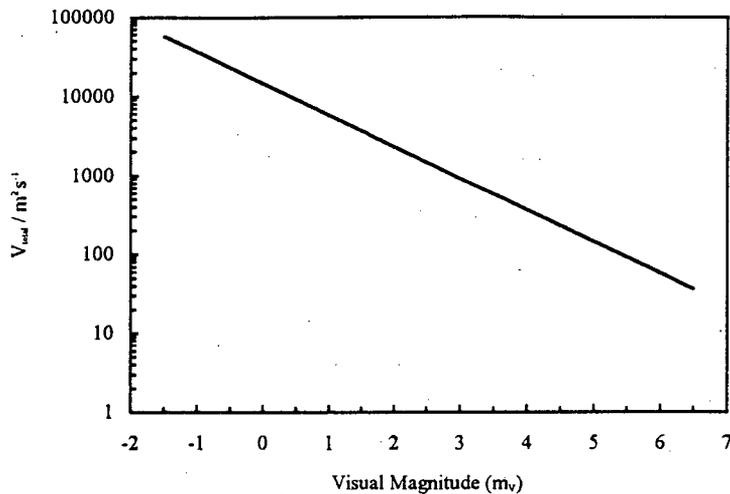
$$K = 1.4 \mu\text{V/e}$$

$$D_{lens} = 0.1 \text{ m}$$

$$t_i = 500 \text{ ms}$$

The star is of spectral type A0V.

Visual stellar magnitude is measured with reference to the response curves of the human eye. Because the response curve of a CCD covers a wider area of the electromagnetic spectrum and has a higher relative peak response [Hecht, 1987; Moller, 1988] the measurement of the magnitude of a star differs from the visual magnitude value supplied by a star catalogue.



**Fig. 3.15 Output voltage vs. Visual magnitude**

The total luminous flux received by a CCD is more than that received by the human eye. It is shown in the fact that photographic magnitude is higher than visual magnitude [Wertz, 1986].

### 3. 5 System Design

The system for this application consists of hardware and software components. Hardware is taken to be the camera and all its components as well as the processing electronics.

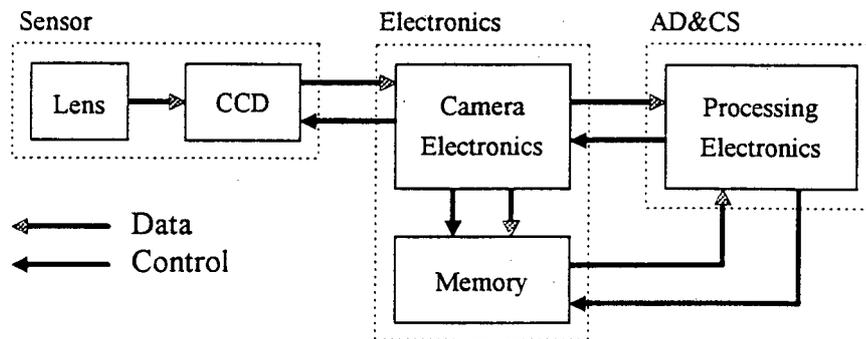
#### 3. 5. 1 Hardware

A CCD camera consisting of a lens and a CCD element was chosen. The lens is of a commercially available compound type, with coated optics and a manually adjustable iris.

The CCD is of the full-frame type. Texas Instruments produces a virtual clock CCD that uses only a single clock signal per line or column which greatly reduces the complexity of the camera electronics [Texas Instruments, 1987]. Electronic sequencing circuitry for this type of CCD is therefore less complex which helps to keep the component count (and cost) down. CMOS technology provides a wide range of low power digital components. Memory requirements for storing the raw image from the CCD are quite low. If 8 bits per pixel are used, the amount of RAM needed is :

$$192 \times 165 \times 8 = 253440 \text{ bits} \cong 248 \text{ kbits} = 32 \text{ Kbytes}$$

As shown in Figure 3.16, the image memory is situated on the camera electronics board. A digital image is loaded into memory after each integration period, from where the processor (which is part of the rest of the AD&CS) can access the data. The type of processor for the AD&CS is presumed to be of the 80C31 family.



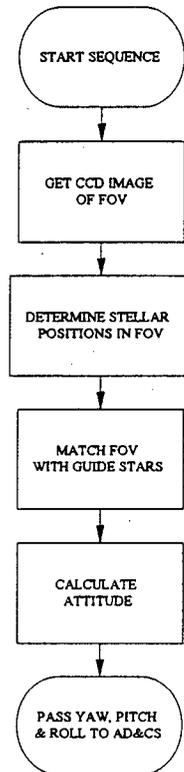
**Fig. 3.16 System hardware**

Control of the camera, such as the length of the integration time and the initialisation of an integrating sequence is done by the AD&CS processor. A typical control sequence is as follows : the processor loads the integration time value into a counter of the camera circuitry. A pulse given by the processor starts the clocking sequence. The values of the CCD pixels are then digitised and clocked into the memory chips on the camera circuit board. Processing of the image in memory then takes place to provide the AD&CS with an estimate of the attitude of the spacecraft. Once the image is in memory, the task of obtaining the attitude of the satellite becomes a software problem.

Hardware imposes definite limits on the accuracy and speed of the system. The camera and electronics can only be as accurate and sensitive as cost and space allow. The quality and performance of the software, on the other hand, is limited largely by the amount of time and effort that is available. Well-written code that implements efficient algorithms can give very good results even if the system hardware has some limitations.

### 3. 5. 2 Software

A high level language such as C or Pascal is suitable for the development of the signal processing and pattern recognition programs. Pascal was chosen because of the researcher's familiarity with the language and the availability of a compiler.



**Fig. 3.17 System software functional flowchart**

Accurate attitude information is required by the AD&CS at least every second. An acquisition sequence is initiated by a signal sent to the CCD camera by the AD&CS processor. After the integration period is over, the CCD image is clocked into the on-board RAM.

A fast, efficient signal processing algorithm is needed to determine the X-Y positions of the stars in the FOV. When the locations and visual magnitudes of the stars in the FOV have been determined, the stars have to be identified. Matching FOV stars with catalogue stars has to be done as fast and accurately as possible.

## 4. Finding Stars in the CCD Image

Once the integration period of the camera has expired, the raw image of the FOV is present in the camera onboard memory. Each pixel's intensity level is represented by a number of bits, usually 8, 12 or 16 depending on the A/D circuitry that is used.

The next step is to obtain the positions, (x,y)-coordinates, of the stars in the FOV. Firstly, stellar images in the FOV are identified. Secondly, the brightness and (x,y)-coordinates of each of the candidate stars are calculated. In this application speed and robustness are the most important criteria. Simple, effective algorithms are used instead of time-consuming digital signal processing and filtering techniques, in order to keep the processing time as short as possible. Stellar image areas are found by using a region growing algorithm in conjunction with a number of stellar image characteristics. A  $(2n + 1) \times (2n + 1)$  interpolation algorithm is used to obtain sub-pixel centroid accuracy.

### 4.1 The Stellar Image Model

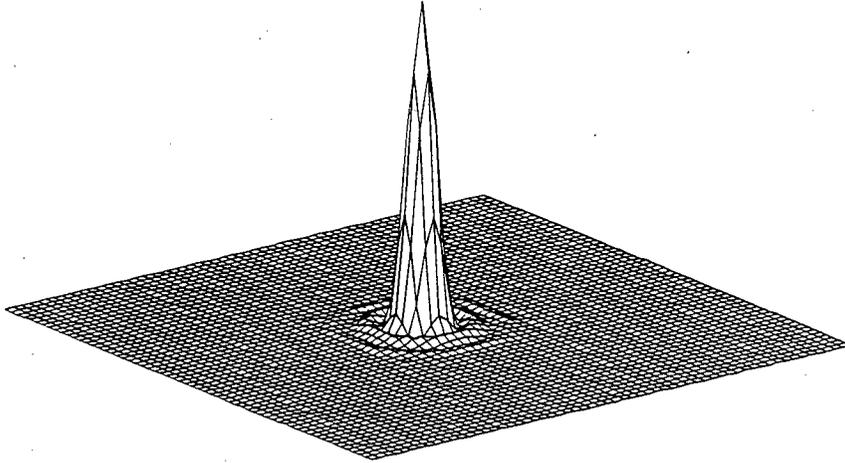
Stellar images are, for all practical purposes, point light sources. Although this may be the case, it cannot be guaranteed that the image of a star, as projected onto the CCD surface by a lens system, will always fall on a single pixel. Firstly, there is always the possibility that a stellar image, however small its diameter, might span the connecting corners of four neighbouring pixels. Secondly, the finite aperture of any optical system causes a diffraction pattern to form at the focal point. In the case of a circular aperture, such as that of an ordinary lens, an Airy disk (Figure 4.1) consisting of a central disk surrounded by dark and light rings is formed. This causes the light to spread over a wider area according to the formula [Moller, 1988] :

$$I(\theta) = I(0) \left[ \frac{2J_1(ka \sin \theta)}{ka \sin \theta} \right]^2 \quad (4.1)$$

where  $J_1$  is the Bessel function of the first order (of the first kind). The centre of the first dark ring of the Airy spot [Hecht, 1987] has a radius of :

$$r \approx 1.22 \frac{f\lambda}{D} \quad (4.2)$$

where  $D$  is the diameter of the lens,  $f$  the focal length and  $\lambda$  is the average wavelength of concern.



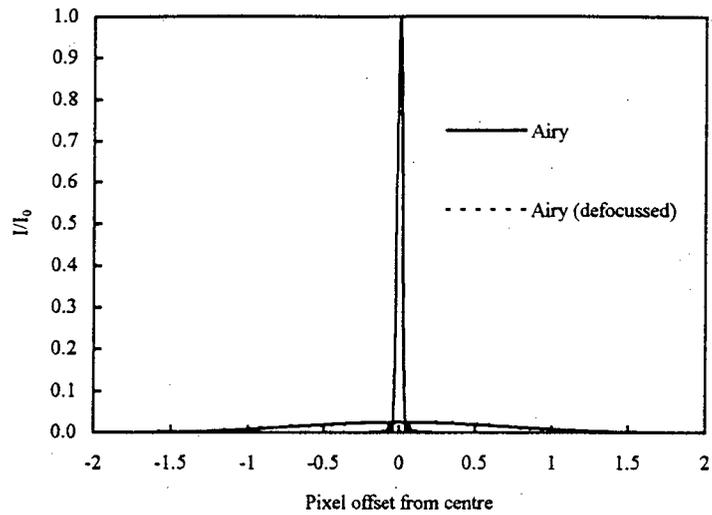
**Fig. 4.1 The Airy function [Moller, 1988]**

Using 555 nm as the average wavelength for visible light and a lens with an  $f/\#$  of 1.4, the diameter of the Airy disk would be 1.89  $\mu\text{m}$ . On a CCD with pixels of 10  $\mu\text{m} \times 10 \mu\text{m}$  such an image could easily fall inside only one pixel. However, for higher accuracy the interpolation techniques, described in Section 4.3, require that the stellar image be defocused over a number of pixels. The minimum resolvable distance that two stars can be apart is set at 2 pixel distances.

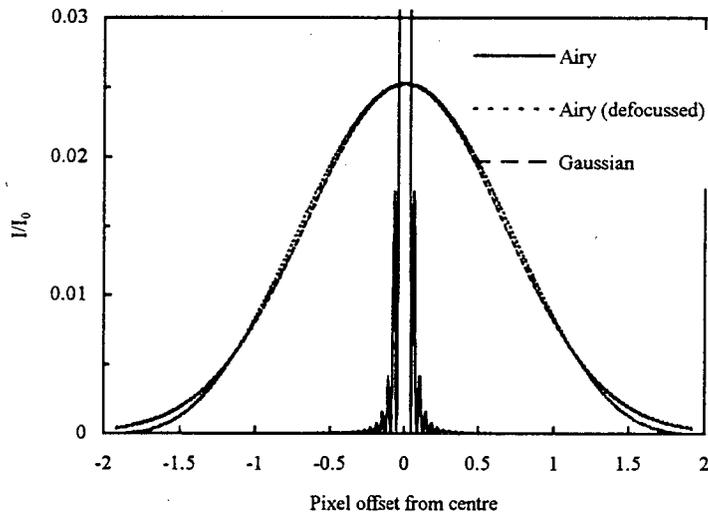
A Gaussian blur spot, however, is mathematically more simple and has been found to be an excellent approximation of the centre of the Airy spot, which is the region of interest [Grossman, 1984]. Another argument in favour of using a Gaussian approximation is that most of the light, [84%; Moller, 1988] is contained within the Airy disk and 91 % within the bounds of the second dark ring. Any smearing or broadening of the blur spot from optical aberrations or a wide spectral band will also make the point spread function tend toward Gaussian [Grossman, 1984]. The intensity of a stellar image is defined as :

$$I = I_0 e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (4.3)$$

where  $I_0$  is the intensity at the centre of the stellar image. The actual defocusing is done before launching of the spacecraft, because the lens has to be fixed at the correct distance from the CCD. In general, blur spots of 2 to 6 detector lengths are used.



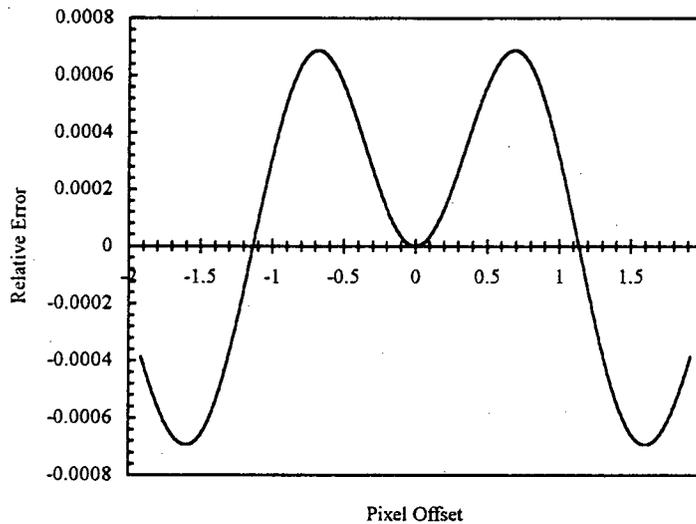
(a)



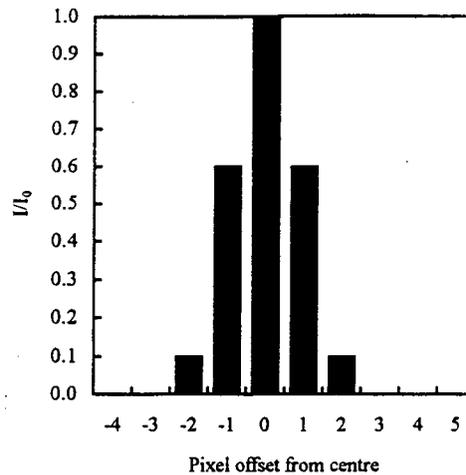
(b)

**Fig. 4.2 The Airy and Gaussian functions [Moller, 1988]**  
**(a) Focused and defocused Airy pattern**  
**(b) Comparison of Airy and Gaussian distributions**

A Matlab program was written to calculate the Airy and Gaussian representations. In the defocused distribution the lens is moved away from the focal distance such that the central Airy disk has its first 0 (at 3.83 for the Bessel function) 2 pixel spacings from the central pixel. The defocusing is modelled by using an intensity of  $I_0/k$  such that the Airy disk has a width of 5 pixels ( $50\mu\text{m}$  for a pixel size of  $10\mu\text{m}$ ). Figure 4.2 depicts a defocused Airy disk and a Gaussian distribution for which  $\sigma = 1.3497$ . At this sigma value the least error between the Airy and Gaussian representations was found for the image contained in the first dark ring of the Airy disk. Figure 4.3 shows that the relative error is less than 0.0008 for the central region of the distribution which makes the Gaussian model an acceptable approximation for this application.



**Fig. 4.3** Relative error between Airy and Gaussian ( $\sigma=1.3497$ ) functions



**Fig. 4.4 Gaussian pixel intensity spread**

The Gaussian distribution is such that the intensity above 30 % of the centre value is spread over a 5 by 5 grid (25 pixels in total). Figure 4.4 illustrates that the pixels in the central 3 by 3 (9 pixel) region have an intensity value of approximately 80 % of the total distribution.

The occurrence of two perturbations of the intensity distribution are expected : over exposure and image smear. The first, over-exposure of the image due to an excessive integration period (which is bound to happen due to the large dynamic range of stellar image intensities), will cause the Airy distribution to tend more towards the Gaussian model in the central region of the stellar image. This will lend more accuracy to the Gaussian model, but only up to the point where the pixels start to saturate.

Image smear, due to movement of the spacecraft during the integration period, will cause the image to be elongated in the direction of motion. This elongation in the plane of the FOV could be in any direction and will decrease the accuracy of the centroid determination algorithms.

## 4.2 Image Segmentation

It is expected that the other sensors (sun and horizon sensors) of the AD&CS will give an attitude measurement accurate to within  $\pm 1^\circ$ . With an FOV coverage of  $10^\circ \times 10^\circ$  and this initial attitude accuracy it is not possible to use any *a priori* stellar position information so that only certain areas of the FOV need to be searched for stars. The whole FOV has to be scanned in order to find all the stellar images.

### 4.2.1 Region Growing Algorithm

Repeatedly accessing all the memory locations of the FOV could be very time-consuming, particularly as the AD&CS expects an attitude update every second. Mathematically complex and time consuming procedures such as spatial filtering and cross correlation are therefore not feasible because of the time limitation. A simple yet robust method is an adaptation of a region growing algorithm [Ballard & Brown, 1982] as shown in Figure 4.7.

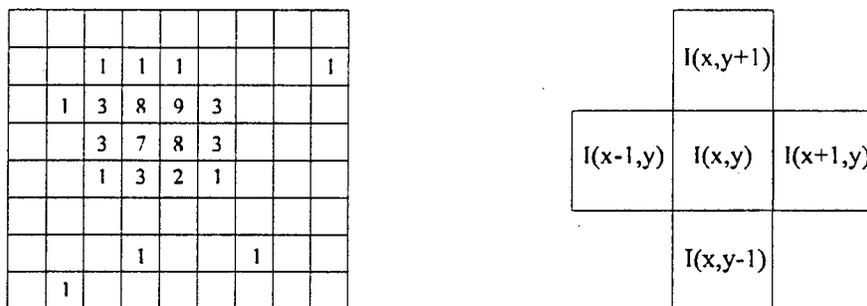
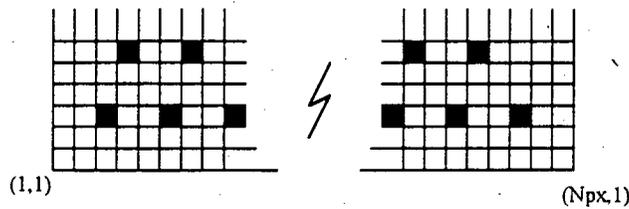


Fig. 4.5 The region growing algorithm [Ballard & Brown, 1982]

The FOV is searched from left to right and top to bottom for pixels that have intensity values above a certain threshold. Each time such a pixel is found, a recursive procedure is started which searches in the directions  $(x-1,y)$ ,  $(y+1,x)$ ,  $(x+1,y)$  and  $(y-1,x)$  as shown by figure 4.5. This procedure sets the value of each non-zero pixel to 0, which effectively erases the object from the image, as it searches in the directions indicated in Figure 4.3. The  $(x,y)$ -coordinate and intensity value of each pixel is stored before it is erased. In this way the characteristics of the object are recorded. Only when no more connected pixels are found is the search through the rest of the image FOV continued. To ensure that no objects are missed, the search is continued from the first pixel of the object that has just been navigated.



**Fig. 4.6 Fast search pattern**

The fact that a stellar image is defocused over a 3 by 3 grid of pixels is used here to speed up the search. Instead of searching the whole  $N_p$  by  $N_p$  grid, only every third pixel has to be evaluated as shown in Figure 4.6. The search is started at pixel (3,3). If its value is below the threshold, pixel (6,3) is checked and so on for the whole row up to pixel number  $N_{px}-2$ . Pixel (4,6) is evaluated next, followed by pixel (7,6) up to pixel number ( $N_{px}-3$ ). In the next row the search is conducted up to pixel ( $N_{px}-2$ ) and in the next up to ( $N_{px}-3$ ) up to the last row in the CCD.

For a CCD with 512 by 512 pixels a total of :

$$\left(\frac{(N_{px}-2)}{3}\right)\left(\frac{(N_{py}-2)}{3}\right)$$

$$= 28900$$

pixels will have to be searched. This is about 11 % of the total number of pixels (26 2144) which will reduce the processing time of the search software substantially.

An object is accepted as a stellar blur spot only when a number of criteria are met, otherwise it is discarded. If the object has a size of less than 7 or greater than 36 elements it is discarded as not being a stellar image. Noise or debris could account for small objects, while the large ones could be due to the sun, moon, planets or even defects in the CCD. When such an out-of-range object is found, the region growing algorithm is completed (thus removing the object from the image), except that the data of the object is discarded before the search for a next object is continued.

When 8 stars have been found the algorithm is halted. At this stage every star is represented by a matrix :

$$S = \begin{bmatrix} x_1 & y_1 & i_1(x,y) \\ \vdots & \vdots & \vdots \\ x_{Ns} & y_{Ns} & i_{Ns}(x,y) \end{bmatrix} \quad (4.4)$$

where  $x$  and  $y$  are the pixel's coordinates,  $i(x,y)$  is its intensity value and  $N_s$  is the total number of pixels in the stellar object.

The following characteristics of an object are used when identifying stellar images:

The first is the total image intensity,  $I$  (the zero<sup>th</sup> moment) [Bokhove *et al.*; 1986]:

$$I = \sum_{ij} f_{ij} \quad (4.5)$$

where  $f_{ij}$  is the intensity value at pixel (i,j)

Secondly, the first moments of the stellar object (the position of its centre)  $\bar{x}$  and  $\bar{y}$  are [Ballard & Brown, 1982; Bokhove, 1986] :

$$\bar{x} = \frac{\sum_{ij} if_{ij}}{I} \quad (4.6)$$

$$\bar{y} = \frac{\sum_{ij} jf_{ij}}{I} \quad (4.7)$$

Lastly, the  $x$ - and  $y$ - widths (second moments) are calculated by [Bokhove, 1986] :

$$\sigma_x^2 = \frac{\sum_{ij} (i - \bar{x})^2 f_{ij}}{I} \quad (4.8)$$

$$\sigma_y^2 = \frac{\sum_{ij} (j - \bar{y})^2 f_{ij}}{I} \quad (4.9)$$

The relationship between the width of the object and the position of its centroid has to be realistic. In this regard the circular symmetry of a stellar image has to be preserved. If the ratio  $\frac{\sigma_x^2}{\sigma_y^2}$  is outside the range  $0.7 < \frac{\sigma_x^2}{\sigma_y^2} < 1.3$ , the object is discarded.

A check for image saturation is also done. If the first two stars that are found have more than 4 pixels that correspond to the saturation signal of the CCD, then the integration period for that FOV is considered to be too long and the FOV image is discarded. A new integration sequence is started with a lowered integration period.

The pixel value threshold of the search routine is crucial to the success of the algorithm and must be variable to accommodate the different integration times of the camera. Herein lies the inherent high noise sensitivity of the algorithm. In an image with a low signal-to-noise ratio the noisy spikes could easily be mistaken for stars. However, in the case of this application, the image generally consists of point light sources on a pitch black background which in conjunction with cooling of the CCD will give a high signal to noise ratio (typically 60 dB at 25 °C [Texas Instruments, 1987]).

## Chapter 4 Finding Stars in the CCD Image

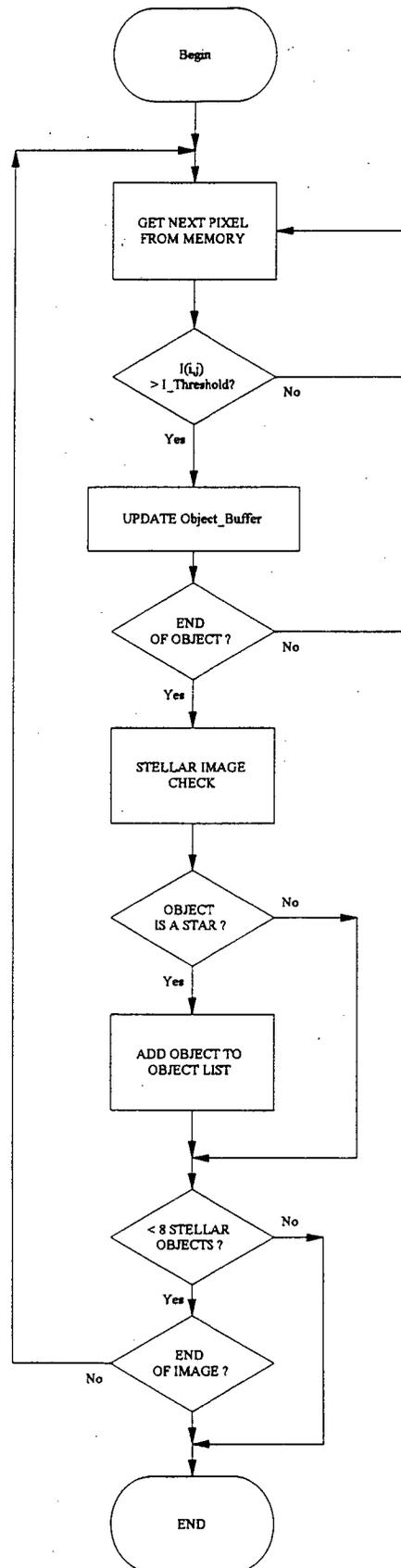


Fig. 4.7 Image Segmentation Flow Diagram

### 4.2.2 Sources of Error

Misidentification of image objects should be kept as low as possible. The pattern recognition algorithms (Chapter 5) rely on the data from the image processing software to determine the spacecraft attitude. Final results of these algorithms can only be as accurate as the stellar positional data that is supplied.

During normal operation in the sun-synchronous orbit, the star sensor should never point directly at the sun. However, during manoeuvres or malfunctioning of the attitude control system, direct imaging of the sun could occur. The CCD will saturate and if the exposure period is too long, could even be damaged. If a saturated image is received from the camera it should be rejected and the operation of the star sensor system suspended. After an elapsed time during which the spacecraft would have rotated through an angle more than that subtended by the sun, the star sensor should become operational once more. This should be an automatic emergency procedure which could also cause a shutter to close the aperture of the CCD camera to prevent damage. The precise implementation of the mechanics and electronics of such a shutter device depends on the available funds and dependence on the star sensor system for success of the mission.

An image containing the moon could still be used. With anti-blooming enabled, only the pixels of the lunar image will be saturated. The region growing algorithm will reject the moon after the pixel amount limit for the algorithm has been reached. Thus the moon will effectively be 'cleared' from the image.

Planets and asteroids on the other hand hold the possibility of actual misidentification. In the sun-synchronous orbit a planet such as Mars subtends an angle of between 2.6 and 8.96 arc minutes depending on its position in orbit around the sun. A planet, especially those farther away, could easily be mistaken for a star. One solution would be to keep ephemerides of all the planets onboard the spacecraft however this is unfortunately not possible in the case of all the minor planets (asteroids) because of their numbers and because some have not yet been found. A fair amount of robustness on the part of the pattern recognition algorithms (Chapter 5) is therefore required.

Electronic and mechanical defects in the CCD are other major sources of error. The CCD should be thoroughly tested before launch. Information about any defective photosites should be recorded and stored on-board the spacecraft. Readings from these pixels could then either be corrected, or ignored in the FOV image. Limited in-flight damage should not hamper the performance of the system. If successive images show certain pixels (or rows, columns or areas of pixels) to have the same value, they should be marked as defective and ignored in subsequent image analyses.

### 4.2.3 Simulation

Matlab and Pascal programs were used to simulate a  $10^\circ$  by  $10^\circ$  FOV. Stars of random brightness were placed randomly in the FOV using the Gaussian intensity distribution of Equation (4.3). The region growing algorithm shown in Figure 4.5 was used to find the first 8 stellar images in the FOV.

Noise was added to the image to determine the signal-to-noise ratio (SNR) performance of the algorithm. It was found that the algorithm is not very immune to noise and that its performance deteriorates rapidly when the SNR drops below 30 dB.

Manufacturing and electronic CCD defects should be detected and corrected before the satellite is launched. An example of such defects is a hot spot (light emitting defects), caused by breakdown between the 'on' and 'off' phases of the vertical clocks that shows up as brighter spots which can saturate at long exposure times. Cosmic rays (as mentioned in Chapter 3) such as muons can generate in the order of 2000 electrons during a normally incident collision with a pixel. At a sensitivity of  $1.4 \mu$  /electron the pixel's value could then increase by about 3 mV.

The success rate of the algorithm is 100 % when the noise amplitude is low and alien objects have sizes of less than 7 and greater than 36 pixels. Defocused planets are the only objects that present a real problem. The point spread function of a defocused planet is very similar to that of a defocused star due to the small angular diameter of planets. The planet Mars, for instance, has an apparent angular diameter of 14 arc seconds at aphelion. It is then not possible to distinguish between the defocused images of stars and Mars on the basis of angular size alone. One solution would be to keep accurate ephemerides of planetary positions onboard the spacecraft and another to use more sophisticated image recognition techniques. Both of these approaches will have to be evaluated in terms of the computing power and storage space that will be available in the microsatellite.

### 4.3 Centroid Determination

In order to reach the attitude error specification of less than 1.5 mrad for the roll and yaw angles of the spacecraft, interpolation techniques are used to calculate stellar centroids to sub-pixel accuracy.

#### 4.3.1 Centroiding Techniques

After an object has been identified as being that of a star, a rough estimate of its centroid is made by choosing the pixel with the highest intensity as the centre. If the highest intensity value is shared by more than one pixel, one of them is chosen as the central pixel while the intensity values of the rest are perturbed in such a way that their values are less than that of the central one. This is done by subtracting a value of 1 to 3 digitisation level values from the pixel values.

The algorithm used for the centroid determination calculates the first moments of the object by using ratios of sums of weighted signals. Only the central  $(2n+1)$  by  $(2n+1)$  pixels, where  $n = 2$ , are used in the calculation. Equations (4.6) and (4.7) are used with  $i, j = -2, -1, 0, 1, 2$  where  $\bar{x}$  and  $\bar{y}$  give the estimate from the centre of the central pixel of the image.

The position of the object  $(x_c, y_c)$  in degrees in the FOV is given by:

$$\begin{aligned} x_c &= (x_{pc} + \bar{x}) \cdot L\theta \\ y_c &= (y_{pc} + \bar{y}) \cdot L\theta \end{aligned} \quad (4.10)$$

where  $(x_{pc}, y_{pc})$  are the coordinates (in pixels) of the central pixel and  $L$  is the detector (pixel) length in degrees.

#### 4.3.2 Sources of Error

A number of sources of error limits the accuracy of the algorithm. Three kinds of errors are considered here. They are the algorithm bias error, signal-to-noise error (including non-uniformity error) and errors caused by distortion of the image.

Algorithm bias error is the most fundamental source of error. It is due to inaccuracy of the algorithm itself and can therefore be eliminated by using a look-up table. That is the calculated centroid (by Eq. (4.6) and (4.7)) is adjusted by subtracting a corresponding value found in the look-up table (Table 4.1).

Noise causes uncertainty in the intensity values of the pixels. This relates to uncertainty in the centroid calculation. The system will have a finite SNR which should be taken into consideration when specifying an attitude accuracy. Non-uniformity error is caused by differing doping levels in the pixels of the device. The variation in response is a function of the quality of the CCD device and can amount to several percent of the peak response value.

Image distortion causes the intensity patterns of stellar images to deviate from the ideal diffraction limited Airy disk. The effect of image smear and distortion in the FOV plane was investigated to determine the sensitivity of the centroid algorithm to these factors.

### 4.3.3 Simulation

A Matlab simulation was carried out using the stellar model described in Section 4.1. The algorithm bias error was first calculated. The position of a star was offset from the centre of its highest value pixel by 0 to 0.5 detector lengths in one dimension (the  $x$ -direction).

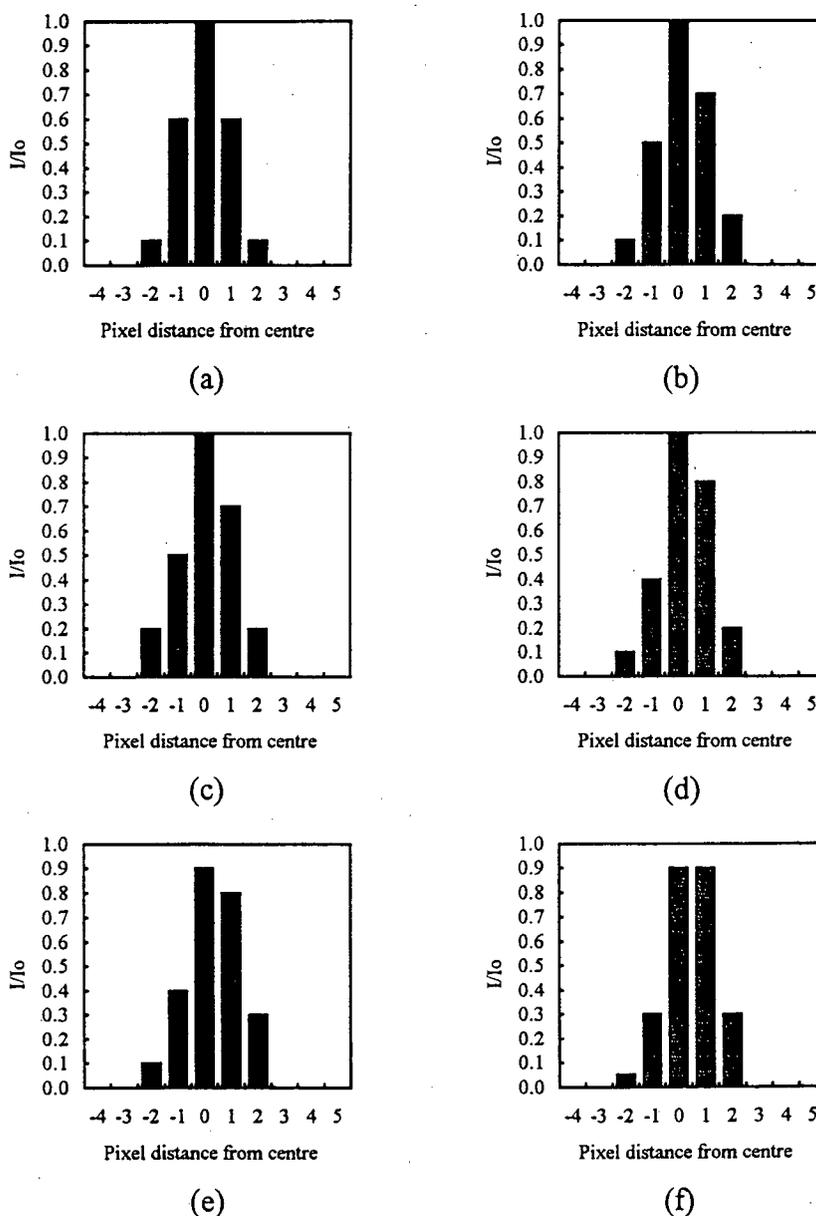


Fig. 4.8 Intensity distribution for offset from 0 to 0.5. (a)  $dx = 0$ , (b)  $dx = 0.1$  (c)  $dx = 0.2$  (d)  $dx = 0.3$ , (e)  $dx = 0.4$ , (f)  $dx = 0.5$

The algorithm bias was calculated for different values of  $\sigma$  for stellar images that were offset by 0 to 0.5 detector spacings from the central pixel. Figure 4.8 shows the intensity distribution of a Gaussian stellar image when the centre of the distribution is offset from 0 to 0.5 detector lengths ( $L$ ) (All further graphs in this chapter have units of  $L$  on both axes) for different values of  $\sigma$ . There is a dependence on  $\sigma$  and the bias with the offset from the centre of the highest value pixel. The graph shows that using a  $\sigma$  of 1.23 for the Gaussian stellar image model gives a near perfect match to the exact Airy model rather than the  $\sigma$  of 1.349 which gives the least error between the Airy and Gaussian functions for the central region of the curve. At  $dx = 0.5 L$  the absolute error (using  $\sigma = 1.23$ ) is  $0.109L$ .

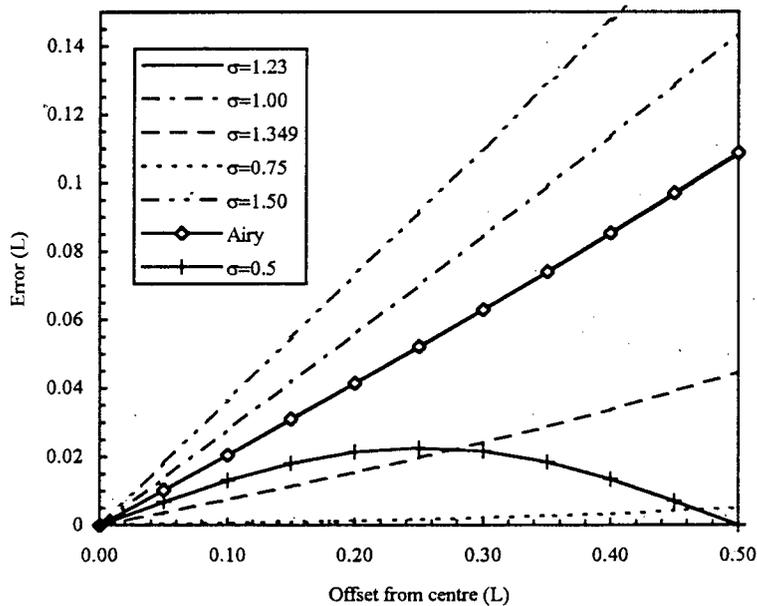
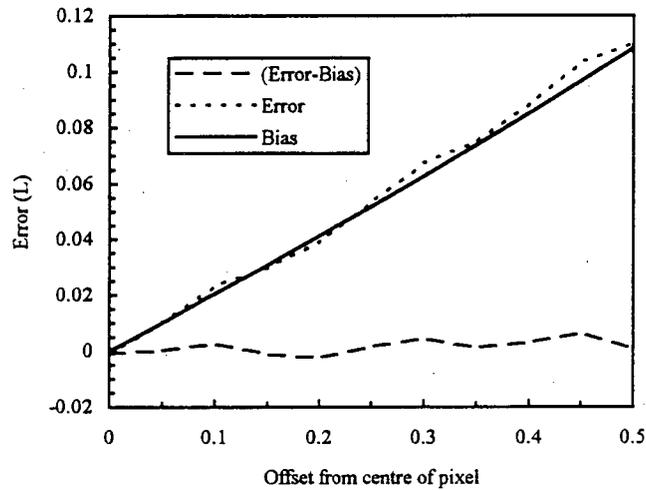


Fig. 4.9 Algorithm bias error

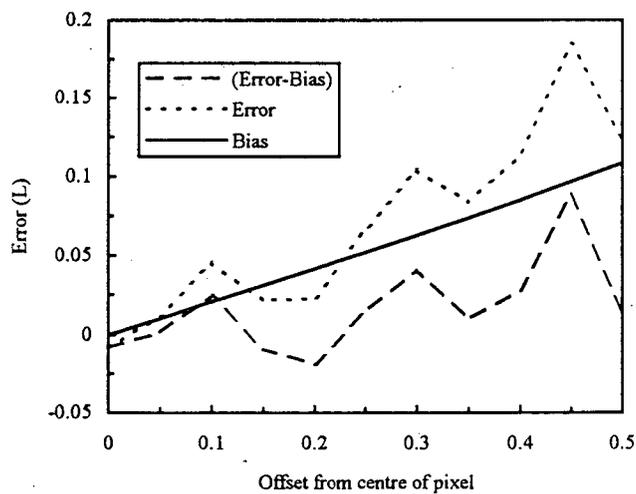
Table 4.1 Algorithm bias error for  $\sigma = 1.23$

Offset	Bias Error
0	0
0.05	0.010284
0.1	0.020604
0.15	0.030994
0.2	0.041490
0.25	0.052125
0.3	0.062934
0.35	0.073949
0.4	0.085203
0.45	0.096728
0.5	0.10855

Noise with a uniform distribution around a mean value was added to the stellar image to simulate random noise sources caused by heat, electronic circuit operation and dark signal non-uniformity (DSNU). Figure 4.10 shows the error for SNR values of 20 dB and 40 dB for a specific Matlab random seed.



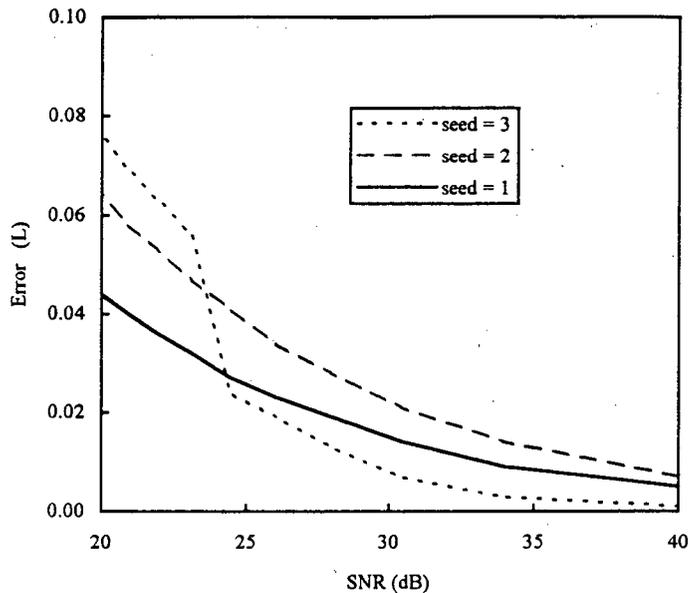
(a)



(b)

**Fig. 4.10 Noise error**  
**(a) SNR = 40 dB (b) SNR = 20 dB**

The maximum and mean error was computed for SNR values between 20 dB and 40 dB. At a 40 dB SNR the maximum absolute error is 0.0063L and at 20 dB 0.088L for this particular seed value.



**Fig. 4.11 Maximum noise error**

Figure 4.11 shows the maximum error values for signal-to-noise ratios of 40dB to 20 dB after bias correction has been applied. A fundamental weakness of the algorithm arose from the case where the seed was set to 3 where a star is positioned at exactly 0.5 detector spacings from the centre of a pixel. In this case there were two pixels with the same intensity value. By just using a simple vector maximum finding procedure such as Matlab's  $\max(N)$  an error can be induced as shown by Fig. 4.11 where the "wrong" central pixel was chosen for SNR values below 25 dB.

In practice there is always the chance that two adjacent central pixels of a stellar image will have the same digitised value. This could be due to: a stellar image having its centre exactly on the boundary between two detectors, saturation of a stellar image, image smear, noise or any other process that modifies pixel values. Figure 4.11 shows that the magnitude of the error produced by choosing the wrong central pixel that is  $0.035L$  which is 3.5 % of the detector spacing in that axis direction. Whether or not these induced errors are significant will depend on the positional accuracy requirement and the noise performance of the system. If this can not be tolerated a more robust coarse centre finding algorithm has to be used.

The influence of smearing of the image was investigated by modelling the motion of the centre of the stellar image according to :

$$\begin{aligned}x_0(t) &= x_0 + s \cdot \cos(\theta) \cdot \left(t - \frac{T}{2}\right) \\y_0(t) &= y_0 + s \cdot \sin(\theta) \cdot \left(t - \frac{T}{2}\right)\end{aligned}\quad (4.11)$$

where  $s$  is the rate of motion in pixels per second and  $\theta$  the direction over the exposure time  $T$ .

Simulations were conducted for a rate of motion of 4, 8, 10 and 20 pixels/s over an integration period of 0.5 seconds. Figure 4.12 presents the error in the x-direction for different values of smear angle. The graph shows that the centroid algorithm is very sensitive for image smear. The influence of choosing the wrong central pixel can also be seen as 'spikes' in the graph.

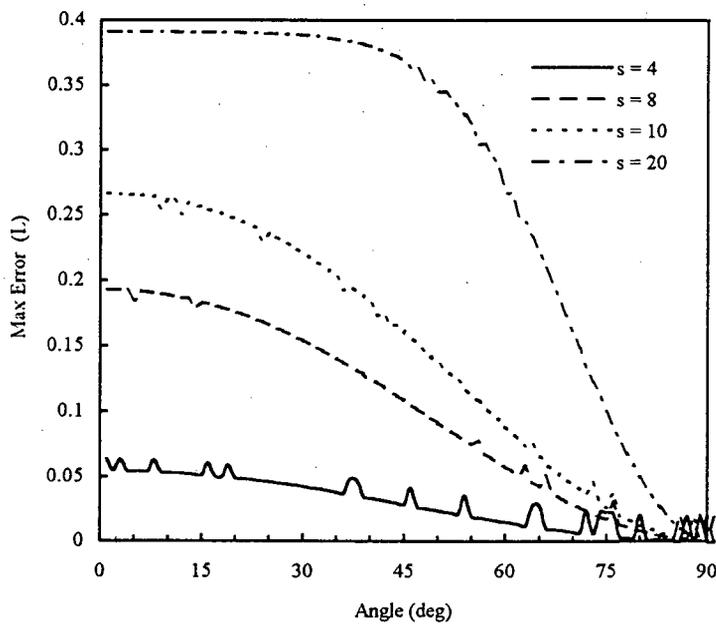
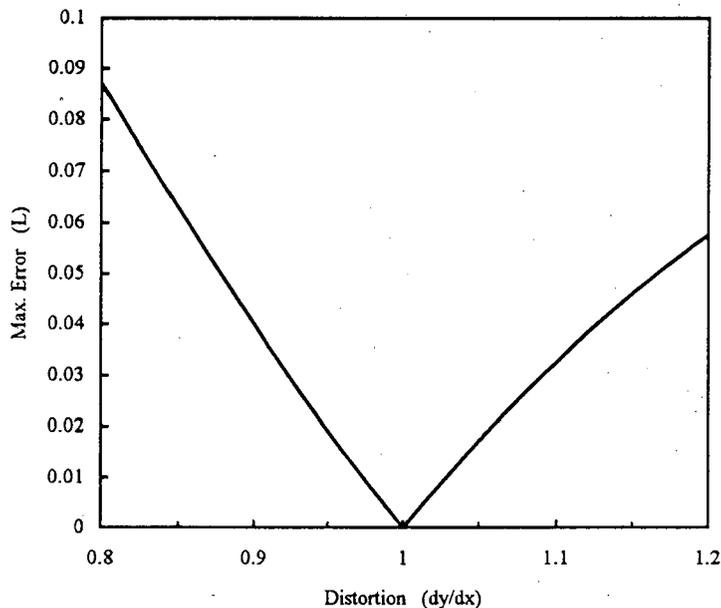


Fig. 4.12 Maximum error vs. smear direction

Distortion of the lens, CCD and mechanical housing of the camera due to sudden thermal changes could also take place. Distortion was modelled as the function :

$$d_r = \frac{dy}{dx} \quad (4.12)$$

At small values of  $d_r$  elongation of the image takes place in the x-direction.



**Fig. 4.13 Maximum error vs. distortion**

Figure 4.13 shows that the centroid algorithm is also sensitive to this form of image distortion. A value of nearly  $0.09L$  is reached at a distortion of only 0.8 (or 20 %). The errors caused by choosing the wrong central pixel as found for the noise and image smear cases are not present in this simulation because of the symmetric nature of the simulated distortion. More complex distortion tests should be carried out if higher positional accuracies are to be considered.

Simulation results show that the highest probable positional error (in the x or y dimensions of the CCD) is less than  $0.1L$  or  $L/10$  using the  $2 \times 2$  first moment algorithm without any corrections. A CCD of size  $192 \times 165$  pixels with an FOV of  $10^\circ$  would give a detector length (in degrees) of :

$$\begin{aligned} L &= \frac{10^\circ}{192} \\ &= 0.0521^\circ \end{aligned}$$

This would require an accuracy of only  $L/2$  to reach the specification. The results indicate that the techniques presented in this chapter will provide the pattern recognition software with stellar positions to an accuracy of at least  $L/10$  even with an SNR of 20 dB, image smear of 4 pixels/s over 0.5 seconds in any direction and image distortion of 50 %. Table 4.1 shows the resulting accuracy in degrees for some commercially available CCDs.

**Table 4.2 Accuracies for different CCDs**

CCD	Format (H x V)	Accuracy in seconds (1 dimension)
TC211	192 x 165	21.8 (0.006°)
TH 7882	579 x 400	9 (0.025°)
Tektronix	512 x 512	7.03 (0.002°)
P8600	385 x 576	9.35 (0.0026°)

## 5. Obtaining the Spacecraft Attitude

Recognition of the stars in the FOV is essential to the functioning of the star sensor system. The robustness and speed of the pattern recognition algorithm determines, to a great extent, what success the attitude determination system will have.

### 5.1 Star Pattern Recognition

There are a number of techniques for identifying stars and star patterns from FOV image information. They all place extensive demands on processor time and memory resources. Correct modelling of the process and selective use of *a priori* information on the satellite attitude, stars, planets and system parameters are therefore used to minimise use of these resources.

#### 5.1.1 Problem Definition

The other attitude sensing devices of the satellite such as sun, horizon and magnetic field line sensors are used to provide a coarse attitude. A star sensor is then used to provide an attitude estimation to a higher accuracy. Apart from providing higher accuracy, the star sensor has much more flexibility in space because it can find attitude information in almost any direction in which it is pointed. In this case, however, the orbit and general attitude of the satellite is fixed and this eliminates large portions of the celestial sphere from the star database.

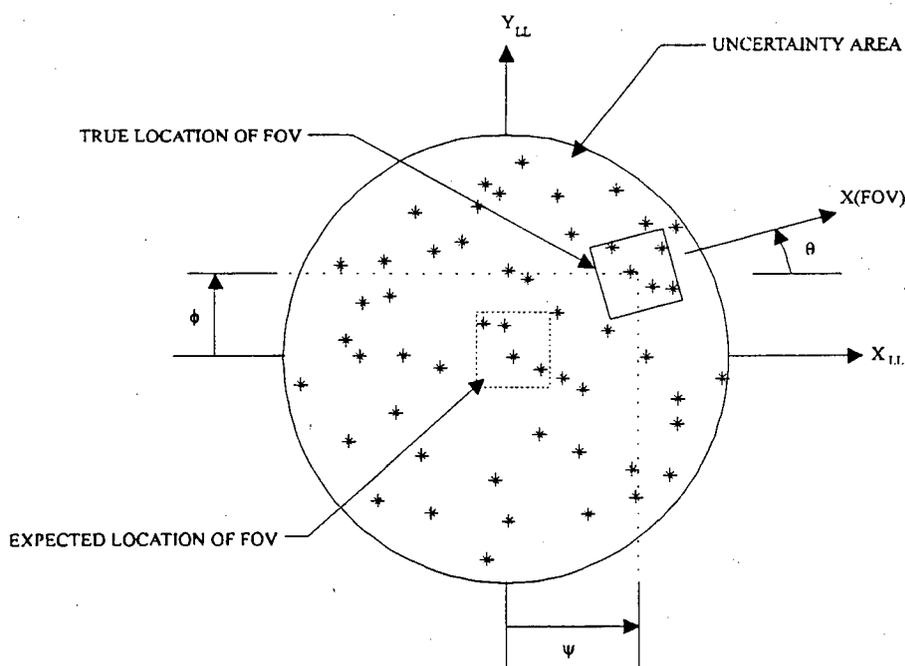


Fig. 5.1 The attitude determination problem [Wertz, 1986]

As mentioned in Chapter 3, the star sensor is mounted on the top ( $-Z_{LL}$ ) facet of the satellite and points into space in the orbit normal  $Y$ -axis direction. The FOV of the camera is modelled as being somewhere in the uncertainty area bounded by the accuracies of the other sensors. By identifying any three stars in the FOV, the 3-axis attitude, (yaw ( $\psi$ ), pitch ( $\theta$ ) & roll ( $\phi$ )) of the satellite can be calculated. The process of identifying stars is done by matching characteristics of the stars in the FOV with those of guide stars in a database.

The use of a smaller uncertainty area speeds up the processing time and reduces the chances of possible misidentification.

### 5.1.2 Recognition Algorithms

Three star identification techniques are: direct match, phase match and angular separation match. The first of these, direct match, dictates that the stars in the FOV must be within a certain tolerance of their counterparts in the guide star database. This technique has the disadvantage that the real attitude must be very close to the predicted attitude. In this case the other sensors are not very accurate, thus eliminating the use of the direct match technique.

Phase matching is usually done when the spacecraft is spinning around a well-defined axis. Observed star azimuths (longitude in the celestial coordinate frame) are compared with those from a star map as the phase between the two coordinate frames are stepped through  $360^\circ$ . In the case of this application, the preliminary specification is that the star sensor will only be used during camera manoeuvres (when the satellite is not spinning) and therefore only the last-mentioned technique will be considered, namely the angular separation technique.

The angular separation technique compares angular measurements of stars in the FOV with those of guide stars. A possible match is encountered when the difference between the two measurements is below a certain threshold. These stars are then given an extra score value and at the end of the process the stars with the highest scores are chosen for the attitude calculation.

This technique has many derivatives. A technique by Van Bezooijen [1990] uses the angular separation of stars, the distance between two stars in the FOV, as well as stellar magnitude calculations to obtain the best match candidates. Another technique by Anderson uses the angles (in the FOV plane) between all the stars in FOV. Yet another algorithm [Cox, 1992] uses the inside angles of triangles (formed by the stars) and matches triangles in the FOV with their counterparts in a star database.

The above angular techniques have been mentioned in the order of the amount of information that is needed. Line lengths are required for calculating angles and angles plus lines are required to represent triangles. The first algorithm, by Van Bezooijen, uses the most fundamental measurement, namely distance, for each match and thus requires the smallest amount of data to be stored on-board the spacecraft.

## 5.2 The Van Bezooijen Algorithm

R.W.H. Van Bezooijen developed an algorithm that has proved to be both robust and fast [Van Bezooijen, 1990]. Two stellar characteristics are used in the algorithm, that is, stellar magnitudes and angular separations. Matching of the FOV image and star map (guide star) information takes place in order to calculate the spacecraft attitude. The accuracy of star database information and pre-processing of stellar data are of the utmost importance. The algorithm can therefore only be successful if the star map information is correct and accurate. Pre-processing of stellar data also plays a major role in improving the performance of the technique. It is claimed that the success rate of the algorithm can be 100 % when guide stars have been carefully selected.

### 5.2.1 The Guide Star Database

The creation of a guide star database is the first step in the pattern recognition process. As much pre-processing as possible has to be done in order to improve the real time performance of the pattern recognition program.

For this application the Yale Star Catalogue is used. It is available on magnetic media (in ASCII format) and is in the public domain. The catalogue contains 9094 stars down to magnitude 8. Each star has 164 bytes of information. The catalogue contains the HD and Durchmusterung [Wertz, 1986] numbers. A small Pascal program was used to extract the RA (to 4 decimal places), declination (to 4 decimal places) and magnitude of each star. This data was then stored as three arrays of real numbers for easy incorporation into the Matlab environment (it has been found that Matlab is particularly useful for the writing of prototype software modules).

When selecting guide stars from the star catalogue, *a priori* information is used to reduce the amount of stellar data that has to be carried on-board the spacecraft. Only stars that will be detected by the camera equipment need to be included in the guide star database. In Chapter 3 it was calculated that only a limited area of the celestial sphere will be visible to the star sensor camera during normal operation.

Another parameter that influences the guide star database is the limit on the angular separations imposed by the FOV dimensions. In the case where the FOV is  $10^\circ \times 10^\circ$  the upper limit on the distance between two observed stars is  $14.1414^\circ$ . A lower bound on angular separations is also imposed by the resolution of the CCD camera and signal processing software. From Chapter 4 the resolving limit of the stars on the CCD is taken as 2 pixel widths. For  $10 \mu\text{m}$  pixel widths, a 512 by 512 pixel CCD and an FOV width of  $10^\circ$  the closest star distance limit for the sensor camera is :

$$\theta_{\min} = \left( \frac{N_{p\min}}{N_{p\max}} \right) \theta_{FOV} \quad (5.1)$$

where

$\theta_{\min}$  = minimum resolvable star pair distance

$\theta_{FOV}$  = FOV width in degrees

$N_{p\min}$  = minimum pixel distance between stars

$N_{p\max}$  = number of pixels per dimension

From equation 5.1 only stars that are more than  $0.039^\circ$  apart can be detected as separate light sources by the system [Hecht, 1987]. In cases where catalogue star pairs have angular separations of less than this limit the dimmer component of the two (the star with the greater visual magnitude value) is discarded from the guide star database.

The number of lines that can be formed by connecting  $N_{stars}$  points is

$$N_{lines} = \frac{N_{stars} (N_{stars} - 1)}{2} \quad (5.2)$$

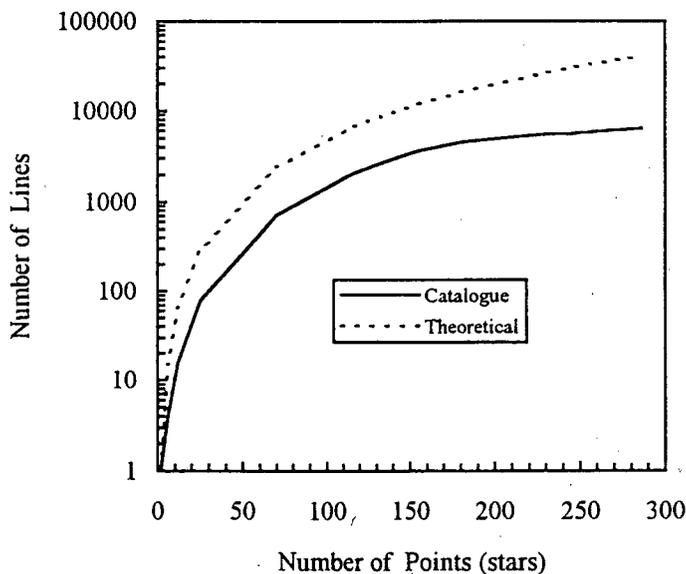


Fig. 5.2 Number of lines versus points

A Matlab program was written to calculate the number of guide stars and lines using the above criteria. For an FOV of  $10^\circ \times 10^\circ$  the number of lines for 133 stars at  $m_v = 6.0$  were found to be 2682. This is about 30 % less than the predicted value of 8778 as shown in Figure 5.2. Because Matlab proved to be too slow (it is an interpreted language) a Pascal program was written for creating the guide star database.

The procedure for selecting guide stars is as follows : The stars in the catalogue are sorted according to RA. Only stars with magnitudes less than the magnitude limit,  $m_{v\_max}$  are then selected and stored in an array **N1**. Next an array, **L1**, containing the angular separations of all the stars in **N1** is created. Only angular separations of less than the hypotenuse of the FOV and greater than the resolution limit of the camera (Chapter 4), 3 arc seconds, are included in **L1**. During this process all stars pairs with a separation of less than the resolution limit are marked. Only the brightest member of each of these marked close pairs is then accepted as a guide star. The other pair is then removed from both the star array, **N1**, and the angular separation (line array) to form the two arrays **N<sub>g</sub>** and **L<sub>g</sub>**. **N<sub>g</sub>** is a numbered list of the (RA, dec) coordinates of all guide stars and **L<sub>g</sub>** is a matrix containing the numbers (as found in **N<sub>g</sub>**) of all the selected stellar pairs :

$$\mathbf{N}_g = \begin{bmatrix} (RA_1, dec_1) \\ (RA_2, dec_2) \\ \vdots \\ (RA_{I_N}, dec_{I_N}) \end{bmatrix} \quad (5.3)$$

$I_N$  = Number of guide stars

$$\mathbf{L}_g = \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_{I_L} \end{bmatrix} \quad (5.4)$$

$$L_i = [n_x \quad n_y]$$

$n$  = Guide star number

$$0 < x < I_N$$

$$0 < y < I_N$$

$$x \neq y$$

$I_L$  = Number of guide star pairs

The algorithm was implemented as shown in Figure 5.3. Lines formed by star that are  $\leq R_{max}$  and  $\geq R_{min}$  are accepted guide star database. Stars that are less than  $R_{MinA}$  degrees from each other can not be resolved by the camera optics. In such a case the dimmer component (higher magnitude star) is marked for deletion. For this calculation  $R_{max} = 14.14^\circ$ ,  $R_{min} = 0.8^\circ$  and  $R_{MinA} = 0.039^\circ$ . The guide star catalogue created in this way for  $m_v = 6.5$  contained 1248 stars. By discarding close star pairs a 5 % reduction in the total number of lines was achieved. Stars were sorted by increasing RA and given a number for easy reference by the software.

**Table 5.1 Number of stars and lines for different magnitudes**

Magnitude	Stars	Lines
5	243	8316
6	767	16493
6.5	1248	32290

In order to simplify mathematical calculations it would be preferable to have the stellar data in integer format. An accuracy of  $0.05^\circ$  requires at least 3 digits after the decimal point. Multiplication by 1000 then transforms all RA and dec values to integers. Storage space required for  $N_g$  and  $L_g$  as calculated above, then requires an array of 1248 integers and an array of 32290 unsigned integers respectively. This equates to a total storage space of 37282 bytes in a Pascal application.

Because of limited on-board memory and in order to optimise the stellar ID algorithms the guide star catalogue has to be divided into a number of zones [Wertz, 1986]. It is then only necessary to search within a zone for a specific star. An example is dividing the sky into zones that overlap in RA [Wertz, 1986]. For this application the zone radius was taken to be the sum of the FOV RA diameter (x-width) and the maximum pointing error caused by the other attitude sensors.

Because the orbit of the satellite is very stable the zones were therefore elongated in the direction of motion of the boresight on the celestial sphere. Provision was made for 30 day's worth of attitude data. Each zone has the dimensions :

$$\theta_{RA} = \theta_{FOV}^\circ + N_{DAY}(\theta_{DAY})^\circ + 2(\theta_{AOS})^\circ \quad (5.5 a)$$

$$\theta_{Dec} = \theta_{FOV}^\circ + 2(\theta_{AOS})^\circ \quad (5.5 b)$$

where

$\theta_{FOV}$  = The maximum angular distance in the FOV

$\theta_{DAY}$  = The angular movement of the satellite boresight in one day

$\theta_{AOS}$  = The accuracy obtained by the other sensors

$N_{DAY}$  = The number of days for which this zone will be used

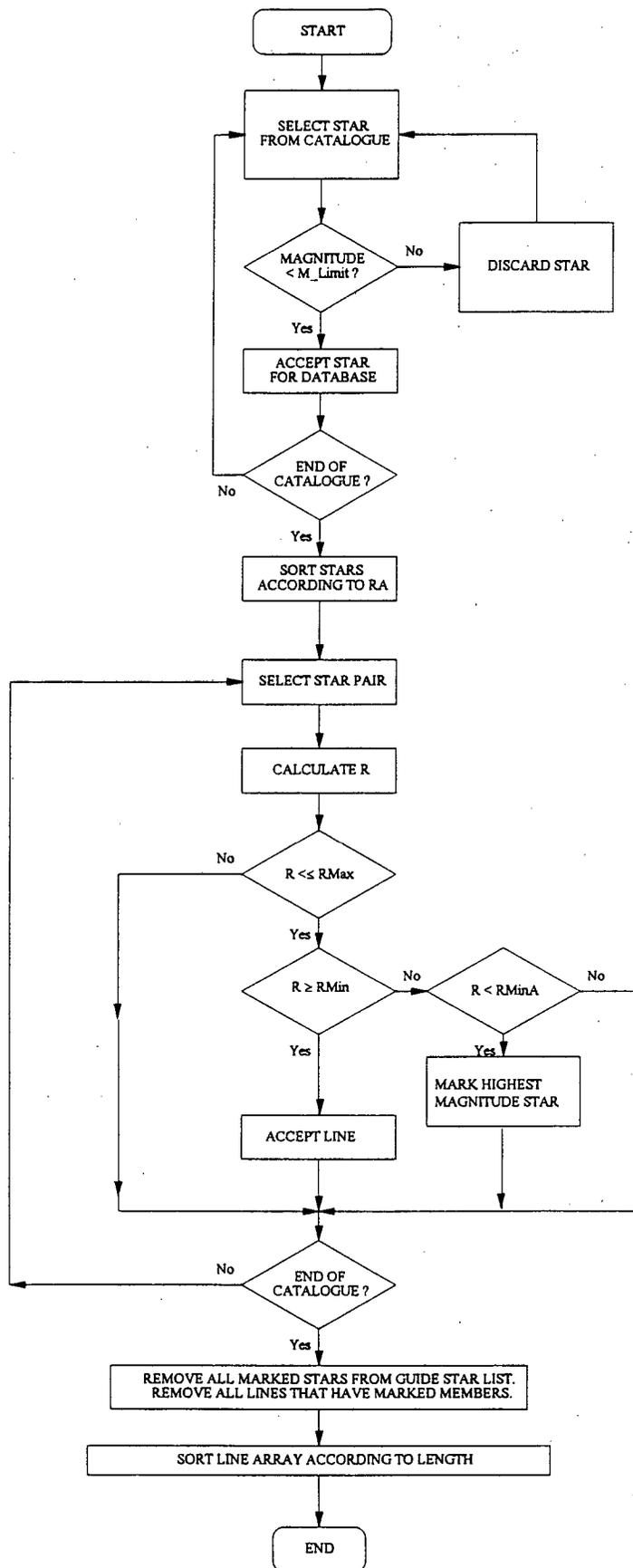


Fig. 5.3 Guide star selection

$\theta_{RA}$  and  $\theta_{Dec}$  are measured in the plane and perpendicular to the plane of the boresight path as described in Chapter 3 respectively. For an FOV of 10 by 10 degrees,  $N_{DAY} = 30$  days and  $\theta_{AOS}$  of  $1^\circ$  the zones have the dimensions :

$$\theta_{RA} = 14^\circ + 30(1)^\circ + 2(1)^\circ = 46^\circ$$

$$\theta_{Dec} = 14^\circ + 2(1)^\circ = 16^\circ$$

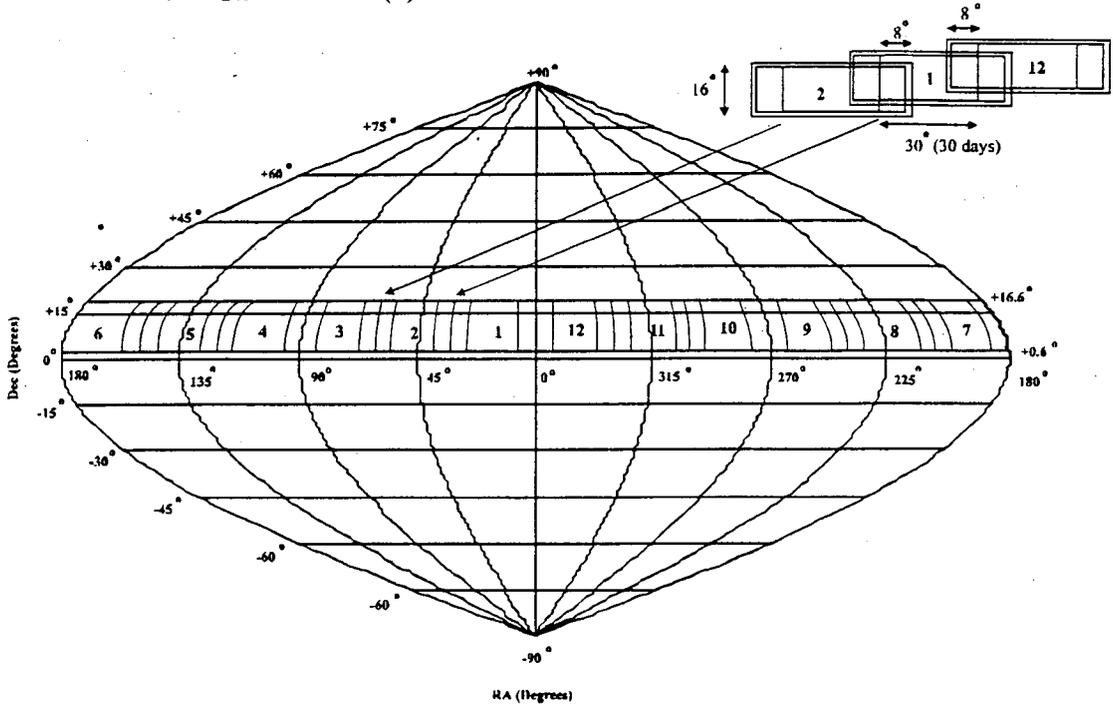


Fig. 5.4 Catalogue Zones

The stellar data is thus divided into 12 zones of  $736^{02}$  each. Matlab calculations showed that the zone with the highest stellar distribution density (zone 3) contains 159 stars (as shown in Figure 5.5 below) at  $m_v = 6.5$  from which 5844 lines were formed.

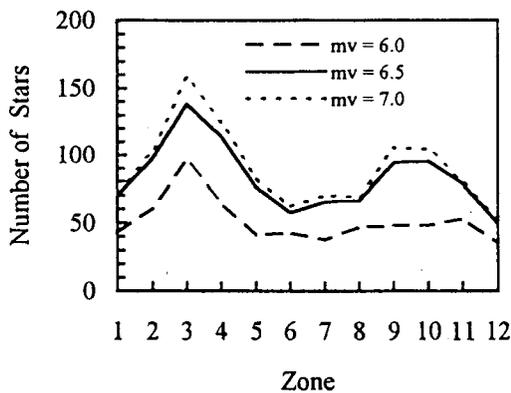


Fig. 5.5 Number of stars in zones

### 5.2.2 Algorithm Description

This section presents a summary of the Van Bezooijen algorithm. References [Van Bezooijen, 1989] and [Van Bezooijen, 1990] can be consulted for further information.

The inputs to the algorithm are the magnitudes and angular separations of all the stars in the FOV. Magnitudes are, as calculated in Chapter 3. In the proposed FOV of  $10^\circ$  by  $10^\circ$  the spatial distortion is negligible, allowing the use of the straight line distance between the two centroids as an approximation of the angular separation. Stellar magnitudes and angular separations of observed stars and guide stars are then compared. The largest group of FOV stars that match a group of guide stars finally presents the solution. All variables referred to in this text are as found in the listing of the match program "match.pas" and its modules in Appendix B. Criteria for a successful match between an observed and guide star group are:

1. The measured angular distance of each pair of observed stars matches the predicted angular distance of the corresponding pair of guide stars to within the distance tolerance (*tdis*).
2. The measured magnitude of each of the observed stars matches the predicted magnitude of the corresponding guide stars to within the magnitude tolerance (*tmag*).
3. The geometry of the group of observed stars is not the mirror image of that of the group of guide stars.

As shown in Figure 5.6 the first step is to generate match groups. These are observed and guide stellar pairs whose angular separations differ by less than the angular separation tolerance. If the match group sizes are large enough, they are processed to find the best match group. This match group is taken to correlate observed stars with guide stars from which the attitude can be calculated.

The observed and guide stars are numbered from 1 to  $N_{total}$ . Three vectors **OMAT** (for Observed stars), **GMAT** (for Guide stars) and **NASS** and one matrix, **MAT**, are created by the match group generation program. Star pair angular separations in the FOV are compared with star pair separations (line lengths) in the Guide star database. Whenever two respective pairs have angular separations that are less than the tolerance the numbers of the observed and guide stars are appended to the vector **OMAT**(*i*) and **GMAT**(*i*), where *i* is the match number. The **NASS** vector contains the weighting value of the match group. **NASS**(*i*) is incremented by 1 every time match group *i* is encountered. Each row, *i*, of the matrix, **MAT**, contains any match groups that are associated with the *i*'th match group (kernel or first match group in **MAT**). Columns (*j*) represent the associated match groups. **MAT** (*i,j*) is appended to the **MAT** matrix when a match is found.

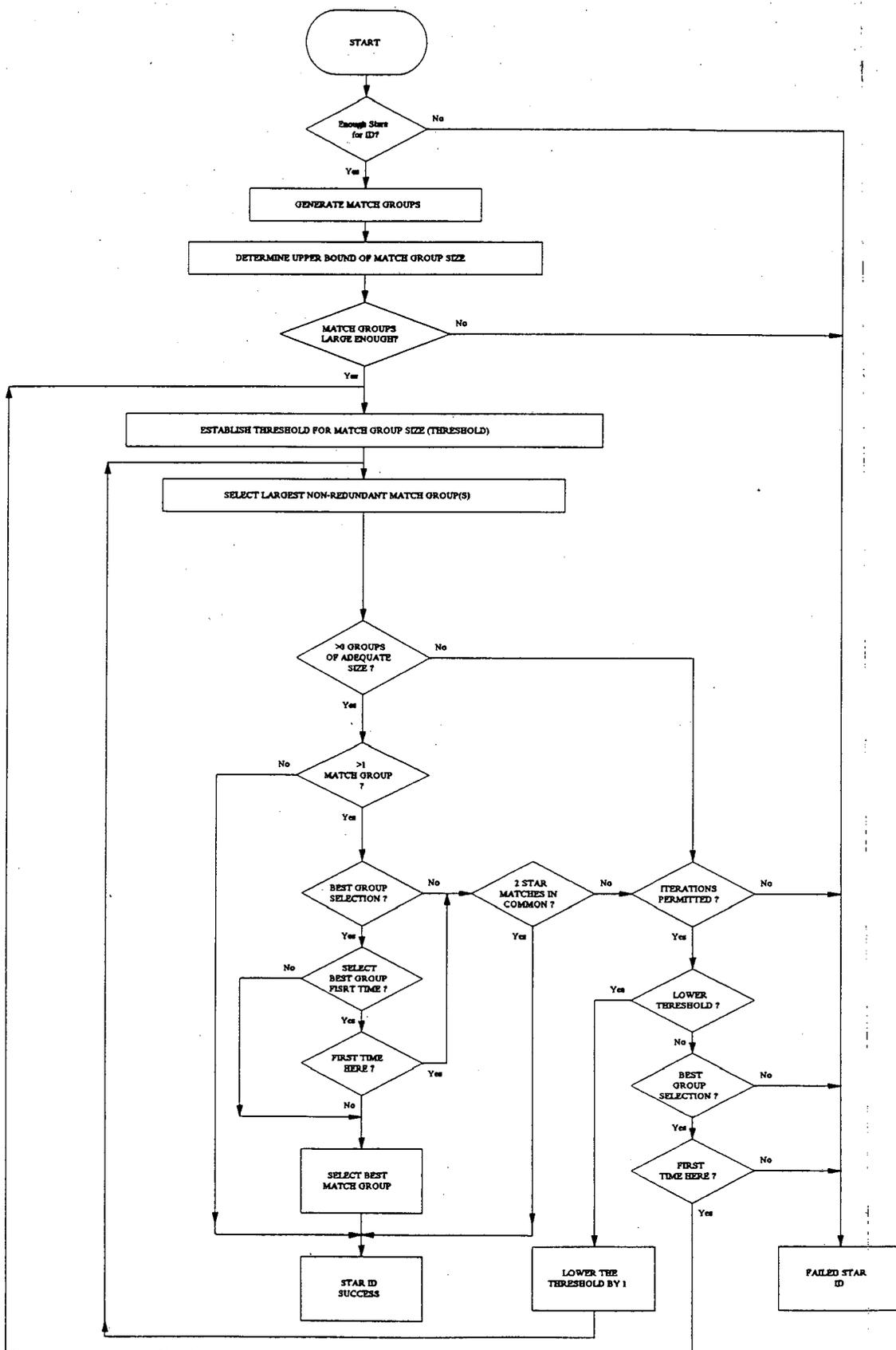


Fig. 5.6 Algorithm Flow Diagram [Van Bezooijen, 1989]

After all the match groups of stars in the FOV have been created, they are evaluated to find the most likely star match pair. An upper size limit has to be determined. If the upper limit is less than the minimum value required (3 or 4 for the acquisition mode) then star ID failure is reported.

A threshold parameter, set at 2 less than the upper limit, on the match group size is used to discard all match groups that are below a certain size. Only those groups above the limit are considered to contain correct star matches. The value of this parameter can be adjusted and is usually 2 below the maximum group size. It is important to note here that all match groups that are associated with any discarded match group are also removed. In this way spurious match groups are eliminated and the amount of possible correct match groups greatly reduced.

Validation of the remaining match groups takes place in the next step. Finally, a mirror test is performed to ensure that the observed and guide star groups are not mirror images of one another. This is done by verifying that the signs vector product of the star pair distances, of the observed and guide stars respectively,  $\underline{a}$  and  $\underline{b}$  and that of  $\underline{a}'$  and  $\underline{b}'$  (as shown in Figure 5.7), are the same.

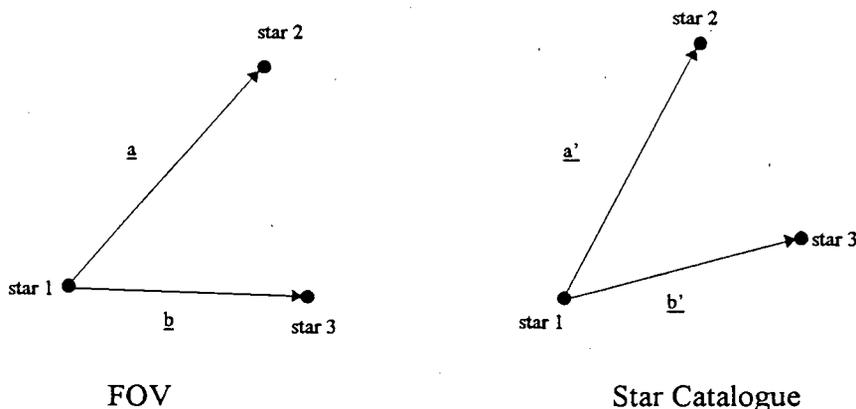


Fig. 5.7 The mirror test

If necessary, iterations are permitted with a lowered threshold. Selection of the best group according to the RSS (root sum squared of the differences) value of the line length errors of each match group is made, where multiple possible match groups are left at the end of match group validation. The match group with the lowest index is then chosen as the most likely match.

The  $(x,y)$  coordinates of the stars of the selected match group are used to calculate the camera boresight direction and from there the spacecraft attitude.

### 5.2.3 Simulation Program Description

A simulation program implementing the Van Bezooijen algorithm was written in Pascal 6.0 on a 40 MHz 386 IBM compatible computer. The suitability of this algorithm to the star sensor system of a microsatellite was evaluated.

A number of parameters have to be set at the start of the program. They are :

- Line length difference tolerance
- Minimum number of stars needed for ID
- Maximum number of stars for ID
- Match group size threshold
- Iterations permitted
- Best group selection used
- The boresight position in RA and declination
- The line length distortion. (An error introduced in the value of the line length)

The output indicates the three stars that were used for the final attitude calculation of each boresight orientation and the position of the camera boresight in right ascension and declination. The yaw, pitch and roll angles of the satellite can also be calculated. After the complete simulation, the average time taken and the success ratio for that section of the particular FOV size and limiting magnitude are calculated.

### 5.2.4 Test Results

A number of passes were made with different parameter values. Without any line length distortion or introduction of image errors such as adding or removing stars from the FOV, the algorithm has a success rate of 100 %. Adding or removing at most two stars from the FOV still gives an acceptable success rate.

Table 5.2 shows the results of simulations that were done in certain sections of the celestial sphere. The regions of high stellar distribution density are the ones where the number of stars in the guide star database zones are the highest. The values show the averages of match sequences done with limiting magnitudes of 6.0, 6.1 and 6.2. Execution time increases logarithmically as the number of guide star pairs (lines) increases with higher limiting magnitude. An overall average of 870 ms for the total match sequence time with a minimum of 330 ms and a maximum of 2360 ms were obtained. Taking into account that the Pascal software implementation of the Van Bezooijen algorithm was not optimised in terms of speed of execution the results show that a complete attitude determination time of 1 second is plausible for the star sensor system if more optimal software is written.

These preliminary results are very encouraging. However, it is clear that a large number of simulations incorporating nearly every possible FOV of the probable area of the celestial sphere should be done before the star sensor system is launched for a particular mission.

**Table 5.2 Van Bezooijen algorithm test results**

Region (h)	Time to generate match groups (ms)	Execution Time (ms)	Number of Lines
1	146	676	1672
5	296	1663	2555
8	203	1026	1540
12	50	380	924
18	106	620	1333
20	183	860	1449

## 5.3 Calculation of the Attitude

Determination of the 3-axis attitude is possible if a pair of observed stars have been matched with a pair of guide stars. Coordinate rotation and translation techniques are used during this procedure.

### 5.3.1 Coordinate Translation and Rotation

Coordinates in the FOV and guide star database are measured relative to different reference points. FOV stellar positions are  $x$  and  $y$  relative to the centre of the FOV (the boresight direction). They can have values between  $-5^\circ$  and  $5^\circ$  (for a  $10^\circ$  by  $10^\circ$  FOV) for both the  $x$  and  $y$  axes. Guide star coordinates, on the other hand, are measured relative to the celestial sphere with the RA range  $0^\circ$  to  $360^\circ$  and the declination range between  $-90^\circ$  and  $90^\circ$  ( $-10^\circ < \text{declination} < 25^\circ$  for this application).

By translating and rotating the guide star coordinates so that they coincide with those of the FOV stars, the yaw, pitch and roll angles can be found. Firstly, the pitch angle is calculated from (in the right handed system) :

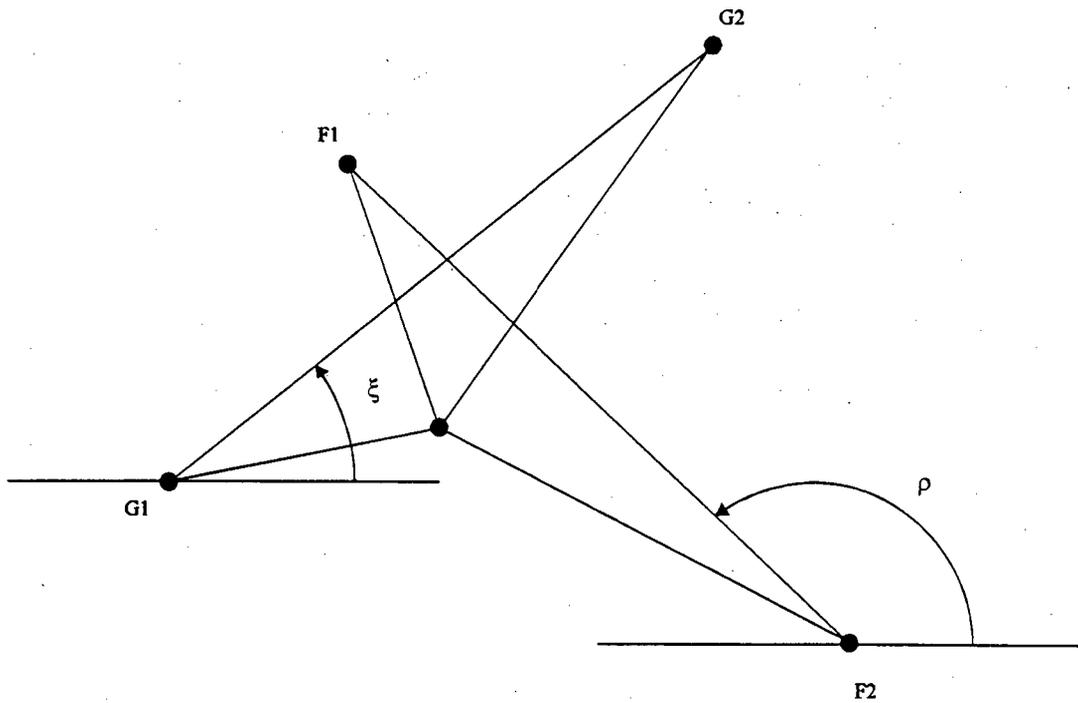
$$\theta = \rho - \xi \quad (5.6)$$

where  $\rho$  is the angle of the line between the two chosen FOV stars and  $\xi$  is the angle between the corresponding guide stars as shown in Figure 5.8. These two angles are calculated by :

$$\xi = \arctan\left(\frac{y_{G2} - y_{G1}}{x_{G2} - x_{G1}}\right) \quad (5.7)$$

$$\rho = \arctan\left(\frac{y_{F2} - y_{F1}}{x_{F2} - x_{F1}}\right) \quad (5.8)$$

In the program use is made of a lookup table to find the correct signs for  $\chi$  and  $\rho$  according to their quadrants.



**Fig. 5.8 Pitch angle calculation**

Secondly the angular separation of the two guide stars (Figure 5.9) and FOV stars are found where roll and yaw movements cause displacement in the y and x axes respectively. Coordinate translation gives these two angles as shown in Figure 5.9:

$$\begin{bmatrix} x_{F1'} \\ y_{F1'} \end{bmatrix} = \Theta \begin{bmatrix} x_{F1} \\ y_{F1} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x_{F2'} \\ y_{F2'} \end{bmatrix} = \Theta \begin{bmatrix} x_{F2} \\ y_{F2} \end{bmatrix} \quad (5.9)$$

where

$$\Theta = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \quad (5.10)$$

The roll error is then :

$$\phi = y_{F2'} - y_{F2} \quad \text{or} \quad y_{F1'} - y_{F1} \quad (5.11)$$

The yaw error is calculated in similar fashion as :

$$\psi = x_{F2'} - x_{F2} \quad \text{or} \quad x_{F1'} - x_{F1} \quad (5.12)$$

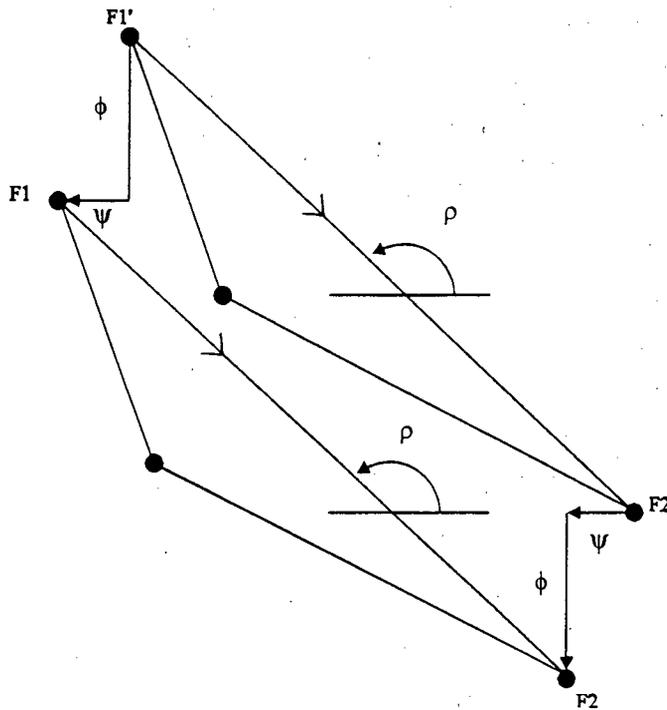


Fig. 5.9 Roll and yaw angle calculations

### 5.3.2 Attitude Errors & Accuracy

The purpose of the star sensor system is to provide extra accuracy over and above that obtained from the other sensors (horizon sensors, sun sensors etc.). An accuracy of  $0.286^\circ$  for the pitch and  $0.025^\circ$  for the yaw and roll axes was specified in Chapter 3.

On the whole the pattern recognition algorithm can only be as accurate as the information supplied by the guide star database. The Yale star catalogue is 90 % complete with RA and dec coordinates of stars correct for epoch 2000 down to 4 decimal places. Stellar magnitudes, on the other hand, are not as accurate. This does not pose a problem because the Van Bezooijen Algorithm relies mainly on positional information to find a positive match. The most likely causes of error are false stellar image identifications by the signal processing software and hardware failure.

Positional accuracy of stellar centroids in the FOV is, as calculated in Chapter 4,  $0.1L$  or  $0.002^\circ$  (for a  $512 \times 512$  CCD and a  $10^\circ$ ) in both the  $X_{FOV}$  (yaw) and  $Y_{FOV}$  (roll) axes. Pitch accuracy depends indirectly on the accuracy of the yaw and roll axes because of the fact that pitch angle is measured in the plane of the FOV from x and y stellar positions.

Figure 5.10 shows that the largest error,  $\theta_{err}$ , in the pitch angle,  $\theta$ , where the positions of two stars in the plane of the FOV are known to an accuracy of  $\delta x$  and  $\delta y$  is :

$$\begin{aligned}\theta_{err} &= \theta_{max} - \theta_{min} & (5.13) \\ &= \text{atan}\left(\frac{y+2\delta y}{x-2\delta x}\right) - \text{atan}\left(\frac{y-2\delta y}{x+2\delta x}\right)\end{aligned}$$

where,

$$y = y_2 - y_1$$

$$x = x_2 - x_1$$

Limits on these parameters are :

$$3(L^\circ) < y, x < 10^\circ$$

$$0^\circ < \delta x, \delta y < 0.002^\circ$$

where

the pixel side length ( $L^\circ$ ) will be about  $0.02^\circ$  for 512 pixels and a  $10^\circ$  FOV.

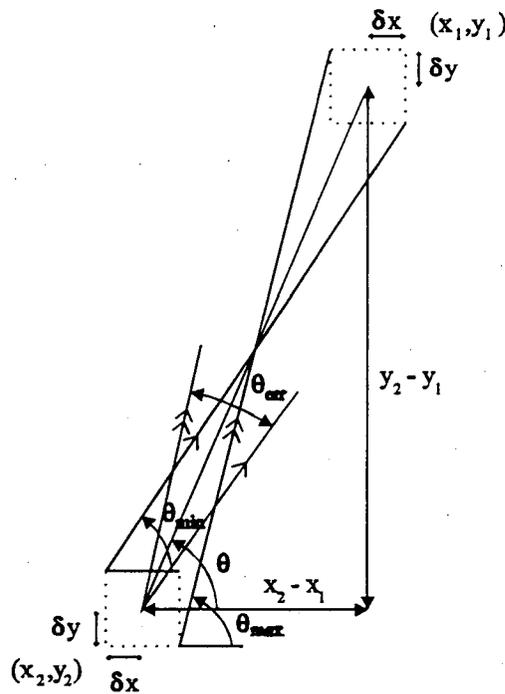
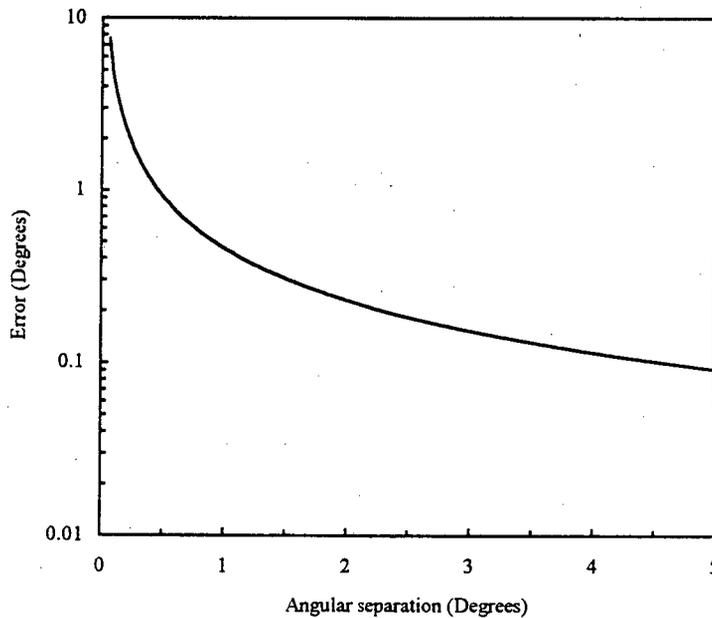


Fig. 5.10 Pitch angle accuracy

Equation 5.13 shows that the pitch angle accuracy is dependent on the angular separation of the stars that are used to calculate the attitude. For a particular centroid accuracy, the maximum error will occur when the stars are at the minimum resolvable distance from each other. The minimum error will be made when stars are separated by the maximum angular distance ( $\sqrt{10 \times 10^\circ}$ ). For  $\delta x, \delta y = 0.002^\circ$ , the minimum error at  $x_{max}, y_{max} = 10^\circ$  will be  $0.0458^\circ$  and maximum error at  $x_{min}, y_{min} = 3(L) = 0.06^\circ$  will be  $7.62^\circ$ .



**Fig. 5.11 Pitch angle accuracy**

It is therefore required that the stars used for attitude calculation should be more than  $1.6^\circ$  apart for a pitch accuracy of  $0.286^\circ$  if the limiting resolution of the sensor camera is  $0.002^\circ$ . This shows the need for more than one star sensor camera if higher guaranteed pitch accuracy is required.

## 6. A CCD Camera Prototype

A CCD camera prototype was developed to evaluate the feasibility of a low-cost star sensor. The TC211 CCD from Texas Instruments was used for the image sensor. A lens from Cosmicar was used to focus the image onto the CCD. The prototype was built on an IBM compatible PC prototype wire-wrap board using HC and HCT technology ICs.

### 6.1 Description

The TC211 is a 192 x 165 pixel full-frame CCD from Texas Instruments. At about R100 per device it is ideal for prototype development. Some of the features of this device are [Barbe, 1976; EEV, 1987; EEV, 1990; Texas Instruments, 1987]:

- Single phase clocking for horizontal and vertical transfers
- Fast clear capability
- High photo response uniformity
- No image burn-in
- No residual imaging
- No image distortion
- No image lag

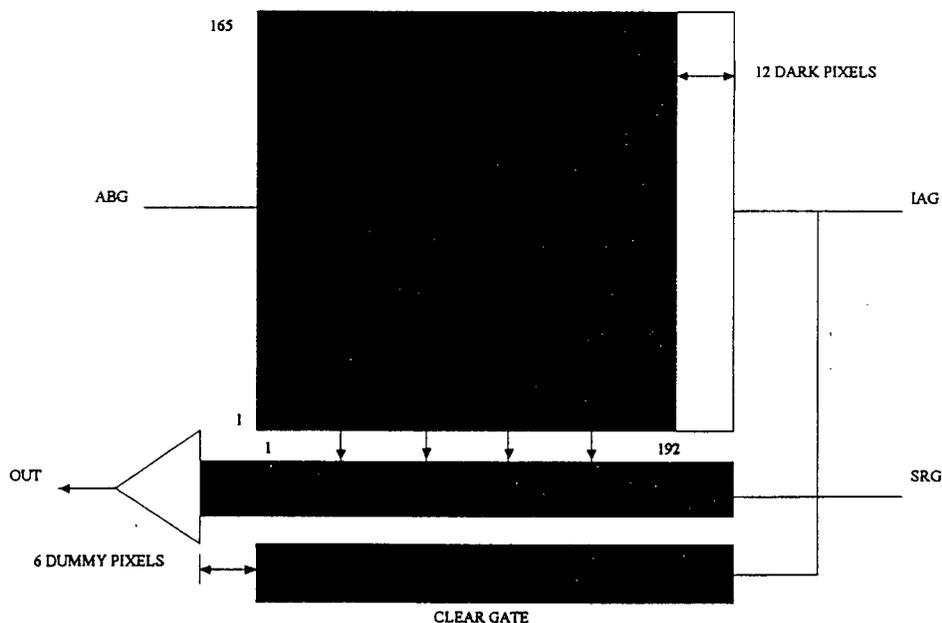


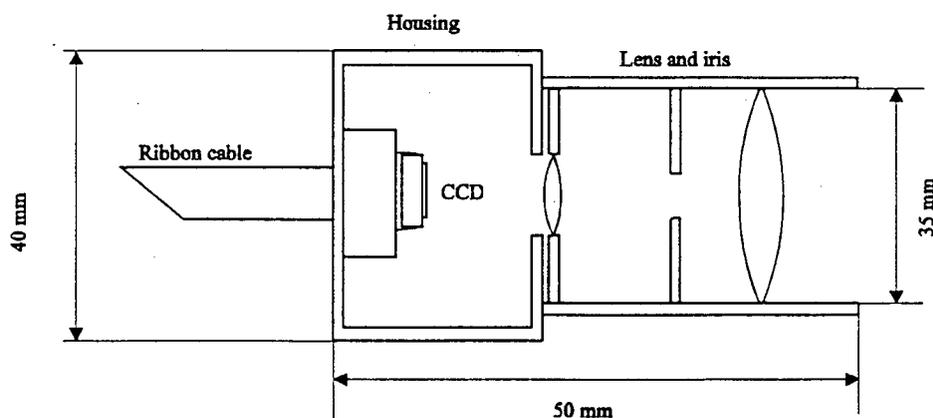
Fig. 6.1 Functional block diagram of the TC211

As shown in Figure 6.1, the image sensing area consists of 165 horizontal lines containing 192 elements each. Each element measures  $13.75 \mu\text{m}$  (H) by  $16 \mu\text{m}$  (V) thus making the image area square. In addition to the 192 horizontal elements there are twelve extra pixels provided at the end of each line to establish dark reference and line clamp. An anti-blooming feature can be activated by supplying clock pulses to the anti-blooming gate which is built into each image-sensing element (pixel). The charge in the device is converted to a signal voltage of  $4 \mu\text{V}$  per electron by a high-performance structure with built-in automatic reset and a voltage reference generator. Furthermore, the signal is buffered by a low-noise, two-stage, source-follower amplifier to provide high output drive capability.

With good spectral response in the range from 400 to 900 nm wavelength, this device is well-suited for optical wavelengths and with its high blue response, it is ideal for the imaging of stellar objects. Its sensitivity is typically  $280 \text{ mV/lux}$  at  $\lambda = 500 \text{ nm}$ .

Light entering the silicon in the image-sensing area causes free electrons to be generated and collected in the potential wells. The amount of charge collected in each pixel is a linear function of the incident light and the exposure time. After exposure (or integration), the charge packets are transferred from the image area to the serial register one row at a time with each clock pulse applied to the image area gate (IAG).

Every time the image area gate is pulsed, an automatic fast clear of the serial register takes place before the row of charges is transferred. The serial register gate (SRG) is then clocked until all the charges have been moved out of the register to the amplifier. The six dummy cells at the front of the serial register are used to transport charges from the register to the input of the amplifier. They are not cleared by the image area gate clock, so care has to be taken to start writing pixel values into memory only after the first six pixels of each line. For more detail the Texas Instruments CCD data sheet and application notes may be consulted.



**Fig. 6.2 CCD and lens configuration**

In the prototype, the CCD chip was mounted in a plastic box (Figure 6.2). A lens mounted on the opposite side of the container has its focal point on the CCD surface. Ribbon cable was used to connect the CCD to its electronics on a wire-wrap prototype board.

## 6.2 Electronic Circuit Design

The main functional components of the camera electronics are the interface, control, memory, sequencer, driver and A/D sections (Figure 6.3).

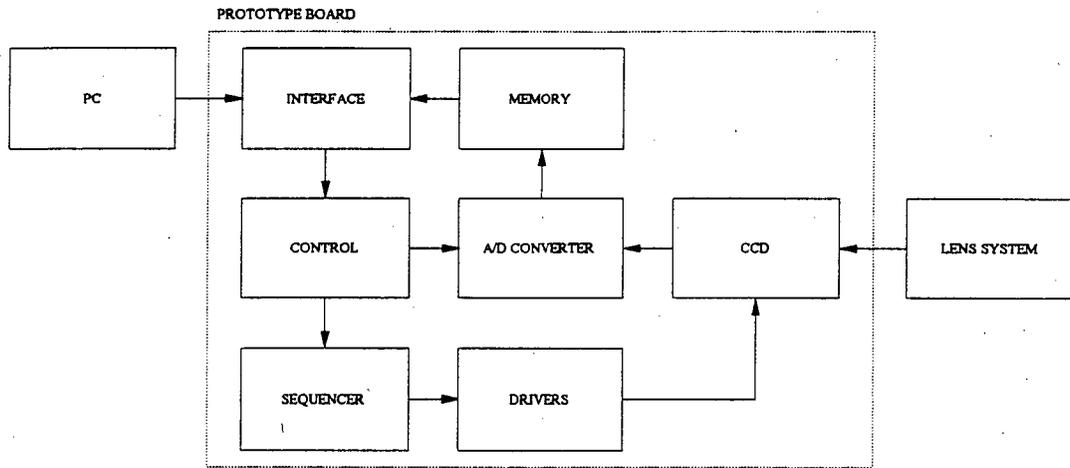


Fig. 6.3 Functional block diagram of the camera electronics

### 6.2.1 PC Interface and Test Environment

The prototype card plugs into the bus of any IBM compatible PC. A 32K memory chip on the prototype board is mapped to the memory addresses A000H:0H to A000H:8000H. This is done by comparison of addresses A16 to A19 with the base address A000H. Address lines A0 to A14 are connected to the card memory through multiplexers. This facilitates the selection of the PC or camera for memory access.

Table 6.1 Control addresses

Address	Name
300H	Camera
302H	PC
304H	Reset
306H	Go

Pascal OUT instructions are used to control the camera from the PC. A PAL (which was later replaced by discrete logic) decodes address lines A1 to A14 to give a possible range of signals of 300H, 302H, 304H and 306H which are not used on the PC. OUT address 300H is used to multiplex the memory to the camera. Once 302H has been written on the I/O bus the PC has access to the memory on the prototype board. Address 304H resets the sequencer circuitry and 306H initiates a sequence. The values written to addresses 300H to 304H are not used. Address 306H however is used to load the integration time (in multiples of the frame time) into the camera electronics. This value is latched into a S/R-latch and used to load the integration time counter, C3.

Power, +5V, +12V, -12V and GND, is supplied from the PC. The clock signal on the bus, CLK4 (4Mhz in this case), is halved and used as the main clock for the counters and the sequencer D-type flip-flops.

### 6.2.2 Clock Generation Circuitry

There are three modes of operation for the CCD. They are Clear, Integration and Clockout as shown in Figure 6.4. In the clear mode the lines of the CCD are transferred one by one into the serial register at the bottom. The serial register is cleared before each transfer. At the end of this operation the potential wells of all the pixels are empty. Next, only power is supplied to the image area of the CCD. Integration of the CCD now takes place. During this time the SRG is kept going to ensure that the bias signal on the serial register is stable to prevent any transitional behaviour when clockout takes place. In the Clockout mode both IAG and SRG are active. Each line is transferred to the serial register from where the pixels are serially clocked out.

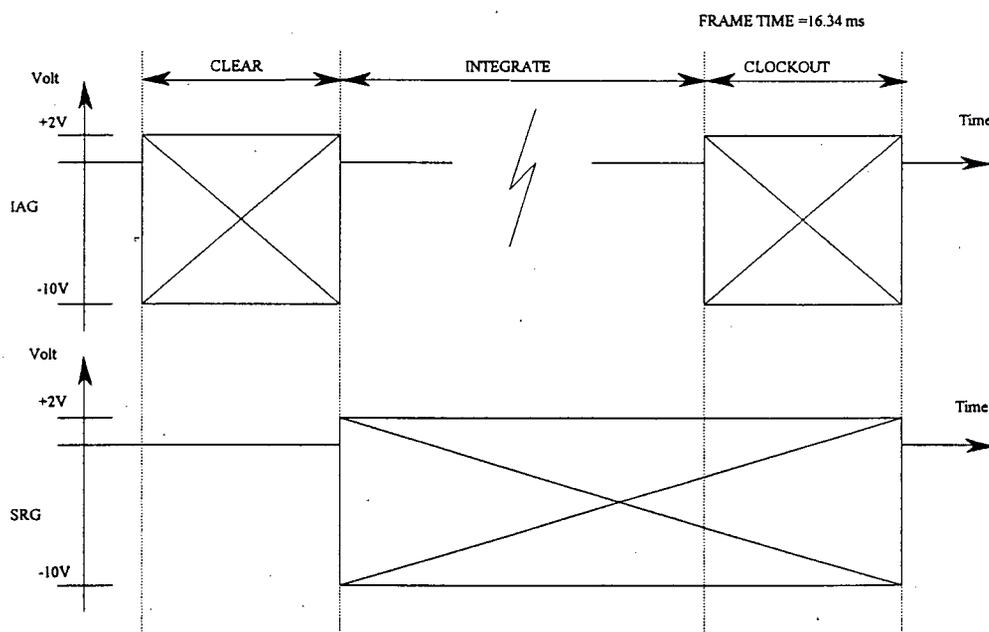


Fig. 6.4 Modes of operation of TC211

One way to implement the clock signals is by using counters. The end of count signal C1 is used as the clock for C2 which counts down from 165 for the rows. C3 is in turn clocked by RCO of C2 and counts down from the desired integration time. From the CCD databook the maximum clock rates for IAG and SRG are 15.7 kHz and 7.61 MHz respectively.

If the length of the IAG count is 208 then the maximum rate of SRG is :

$$15.7 \text{ kHz} \times 208 = 3.26 \text{ MHz}$$

For this implementation an 8 Mhz IBM compatible PC was used. The clock signal of 4 MHz on the extension bus was used for the main clock of the circuit. This signal was divided by two to give an SRG rate of 2 MHz. (Experiments with continuous operation of the CCD showed that the sensitivity of the device improves with a lower clock rate which is mainly due to the longer integration times.) The rate of IAG is therefore :

$$\frac{1}{208 \times 0.5 \cdot 10^{-6}} = 9.6 \text{ kHz}$$

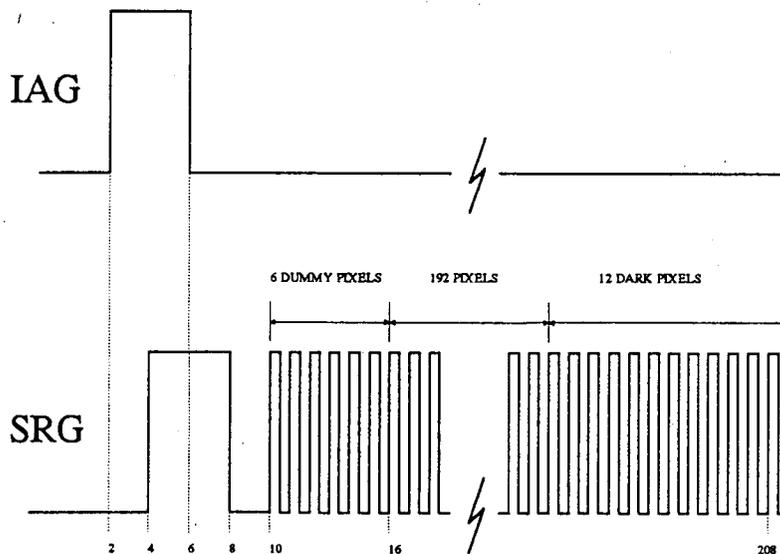


Fig. 6.5 Clock signals IAG and SRG

An Algorithmic State Machine (ASM) [Mano, 1984] chart of the CCD operation was drawn up (Figure 6.5). There are 7 states, numbered T0 to T6, in the ASM chart. The first state, T0, is marked "INITIAL STATE". In this state all the parameters are reset. When the "GO"-signal is given, the sequence starts by transferring operation to state T1. Here the memory counter Cm is reset, the integration counter C3 loaded with the desired value and the signal, A, is set to a high TTL level.

State T2 follows directly on T1. In this state the line counter, C2, is loaded with the value of 165. This counter then counts down from 165 to 0 during the sequence. In state T3, which follows next, the pixel counter, C1, is reset. State T4, where C1 is incremented by one is then repeated 208 times.

During the T4 loop, the logic signals of IAG and SRG are formed. Each time state T5 is reached, counter C2 is decremented by one. If C2 has reached 0 then control is transferred back to T3 to form another line.

At the end of a frame, signal A is reset and T6 is reached. This ensures that IAG is not active during the integration time. If signal B is high at this stage, then the sequence is finished and control is returned to the "INITIAL STATE", T0.

If B is low then the first frame has just been completed and the integration has to start. The integration time counter, C3, is decremented by one and control is transferred to state T2 and another frame is generated. When counter C3 has reached 0 the last frame has been reached. Signals A and B are set to allow SRG and IAG to be active in order to clock out the 192 x 165 pixels of the CCD. After this frame signal B will be set and the sequence will end. From the ASM chart the circuit was designed using standard techniques.

To test the design, a first implementation was done using one flip-flop per state. The circuit was drawn directly from the ASM chart. This has the advantage of easy implementation and debugging.

Using one flip-flop per state is not the optimum implementation in terms of the number of components used. In fact, as far as the number of flip-flops is concerned, it is the maximum implementation. For the satellite implementation where weight and volume are to be kept as low as possible, it is imperative that the component count is kept to a minimum. It should also be kept in mind that maintenance and alterations to the circuitry is not applicable here. The design should rather be optimum than easy to design. A different approach was therefore adopted.

To implement the state machine a design with D-type flip-flops is a good choice. The circuit can be derived by inspection from the state table, and requires less components than the one flip-flop per state implementation.

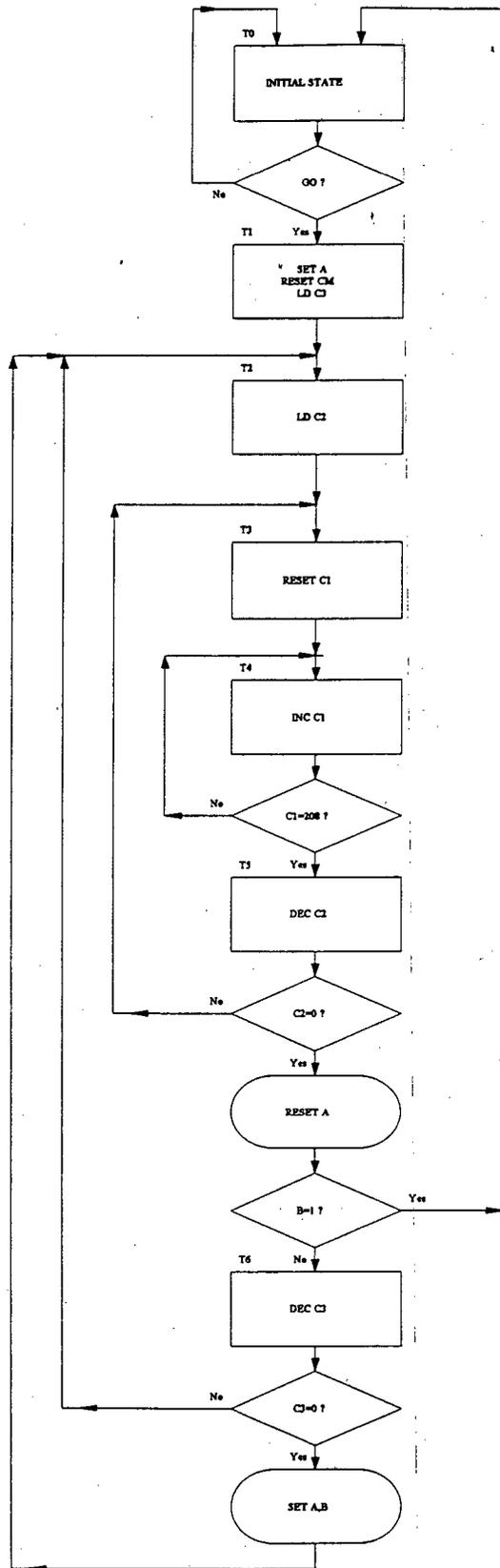


Fig. 6.6 Camera State machine

From the ASM chart the following inputs and outputs are identified :

Inputs : GO, C1Z, C2Z, C3Z and B  
 Outputs : T0 to T6, A<sub>set</sub> and B<sub>set</sub>

**Table 6.2 State table of the ASM chart**

INPUTS							OUTPUTS		
State	Q(t)	GO	C1Z	C2Z	C3Z	B	Q(t+1)	A <sub>SET</sub>	B <sub>SET</sub>
T <sub>0</sub>	000	0	X	X	X	X	000	0	0
T <sub>0</sub>	000	1	X	X	X	X	001	1	0
T <sub>1</sub>	001	X	X	X	X	X	010	0	0
T <sub>2</sub>	010	X	X	X	X	X	011	0	0
T <sub>3</sub>	011	X	X	X	X	X	100	0	0
T <sub>4</sub>	100	X	1	X	X	X	100	0	0
T <sub>4</sub>	100	X	0	X	X	X	101	0	0
T <sub>5</sub>	101	X	X	0	X	X	011	0	0
T <sub>5</sub>	101	X	X	1	X	0	110	0	0
T <sub>5</sub>	101	X	X	1	X	1	000	0	0
T <sub>6</sub>	110	X	X	X	1	X	010	0	0
T <sub>6</sub>	110	X	X	X	0	X	010	1	1

From inspection, the D-type flip-flop next state outputs are found to be :

$$Q1(t+1) = T0.GO + T2 + T4.\overline{C1Z} + T5.\overline{C2Z}$$

$$Q2(t+1) = T1 + T2 + T5 + T6$$

$$Q3(t+1) = T3 + T4 + T5.C2Z.\overline{B}$$

The inputs to the three D-type flip-flops are created from the three equations for Q1, Q2 and Q3. Seven of the output lines of the 3 to 8 decoder are used for the states T0 to T6.

$$A_{set} = T0.GO + T6.\overline{C3Z}$$

$$A_{reset} = T5.C2Z$$

$$B_{set} = T6.\overline{C3Z}$$

$$B_{reset} = T0$$

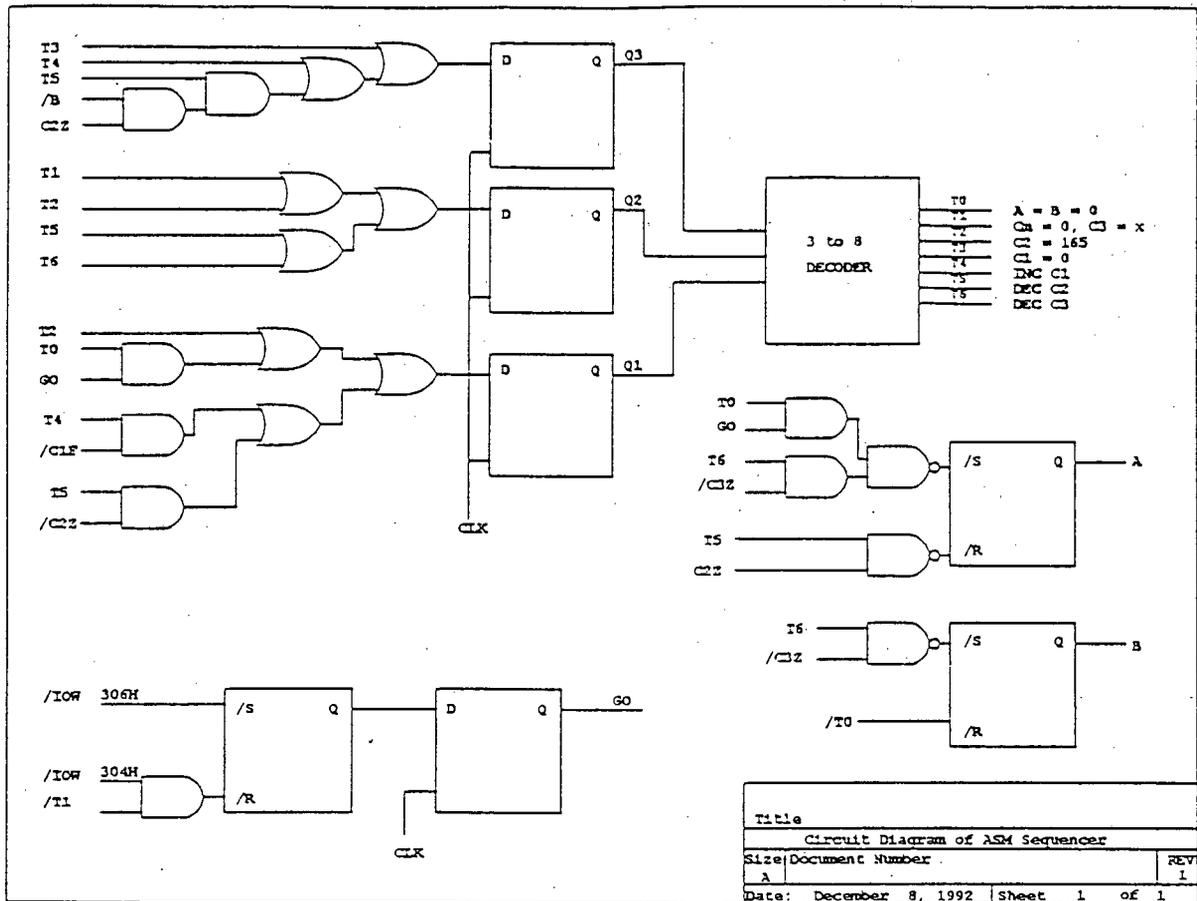


Fig. 6.7 Circuit diagram of ASM implementation

The "GO"-signal is also formed by an S/R-latch.  $\overline{IOW306H}$  sets the latch and  $\overline{IOW304H}$  AND  $T1$  resets it when state T1 is reached. The output of the S/R-latch is connected to the D-input of a D-type flip-flop.

$$GO_{set} = \overline{IOW306H}$$

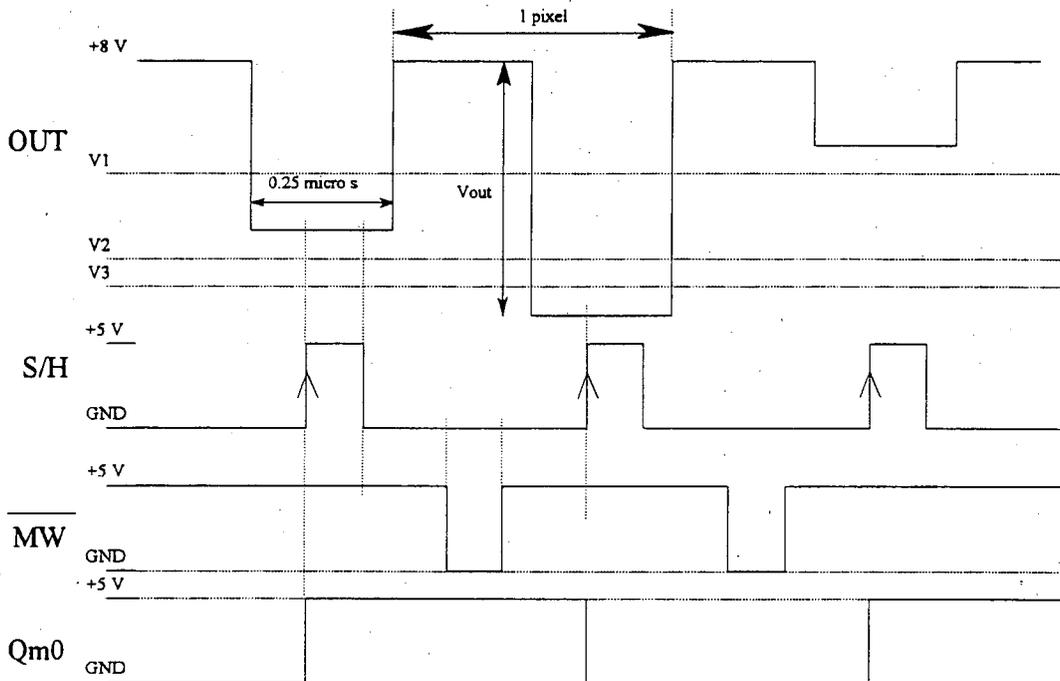
$$GO_{reset} = \overline{IOW304H} \cdot T1$$

The main clock, CLK4, is connected to the latch and ensures that transitions can only occur on the clock pulses.

### 6.2.3 Output Signal Conditioning

The output signal consists of negative going pulses on a DC level (8V for the tested CCD). For each pixel the output signal goes low proportional to the amount of charge that was accumulated in the cell. These levels differ slightly from one device to another but vary greatly with clock frequency. The lower the frequency, the higher the sensitivity but then there is also a greater chance of image smearing. Saturation (after amplification) at the chosen frequency of 2 MHz takes place at about 4V and dark levels are close to 8V.

A simple A/D circuit was employed to register the individual pixels. By combining signals CLK4, Q0 (the first bit of counter C1), A and B a sample-and-hold signal, S/H, was created. This was used to enable a D-latch. When S/H goes high the output of the latch follows the input. The value of the last input is kept when S/H goes low, as shown in Figure 6.8. During the time when S/H is low a level comparator is used to fix the pixel value at the correct TTL level (0 or 1). Halfway between each S/H pulse a /MW signal goes low to write the pixel value into memory. The S/H signal (rising edge) is also used to clock the memory counter which counts from 0 upwards after it has been reset at state T1 (see ASM chart).



**Fig. 6.8 Output and associated signals**

Three voltage levels, that is 4 brightness values are used to convert the analogue value of each pixel to a digital value. The four brightness values can be encoded to 2 bits and stored in memory which results in a 2-bit A/D circuit. Using a 32 K memory chip every pixel can be stored in a byte. The total memory used for a single image is then :

$$192 \times 165 = 31680 \text{ bytes}$$

This is four times the memory required with decoding but saves processing time that would have been taken up by decoding each byte when reading the memory from the PC. This method of A/D was used purely for experimental purposes. The actual CCD camera should rather use a standard A/D IC which could give 8 or 12 bit per pixel values.

Level comparators were used for the prototype so that each digitisation level could be adjusted separately to get the best response. Further tests (which is beyond the scope of this document) on non-linear digitisation level spacing could be done to get the best response for a stellar image.

### 6.3 Prototype Measurements

The wire-wrap prototype was built and tested in a laboratory. Output voltage levels were examined on an oscilloscope.

At first the CCD was run in continuous mode at an integration time of 20 ms. It was found that the device is very sensitive and tends to saturate completely when the manual iris is opened too wide. Some tests were conducted with differing clock signals. The results showed that the device is much more sensitive at lower clock speeds. This technique can however not be used to improve device sensitivity because the image tends to smear when clocked out.

A Pascal program was written to control the CCD camera. Firstly the user is required to enter an integration time. Secondly the user selects the ENTER key on the PC keyboard to start the sequence. The CCD camera then completes the sequence and writes the image data to the memory chip onboard the camera circuitry. The PC then accesses this memory and displays the image on the VGA screen at a resolution of one VGA pixel per CCD photo site.

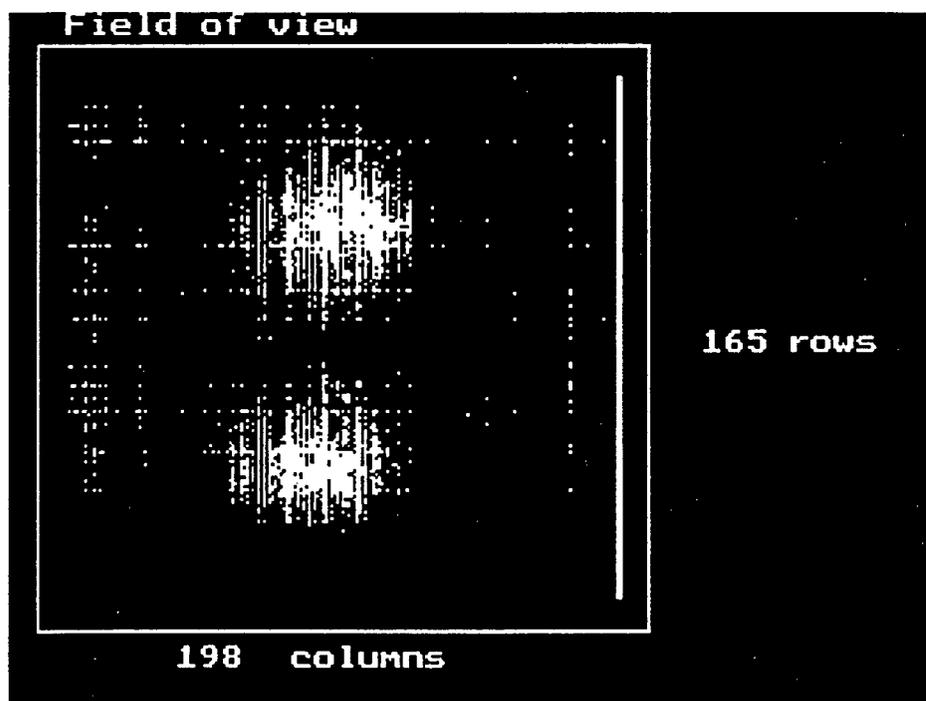
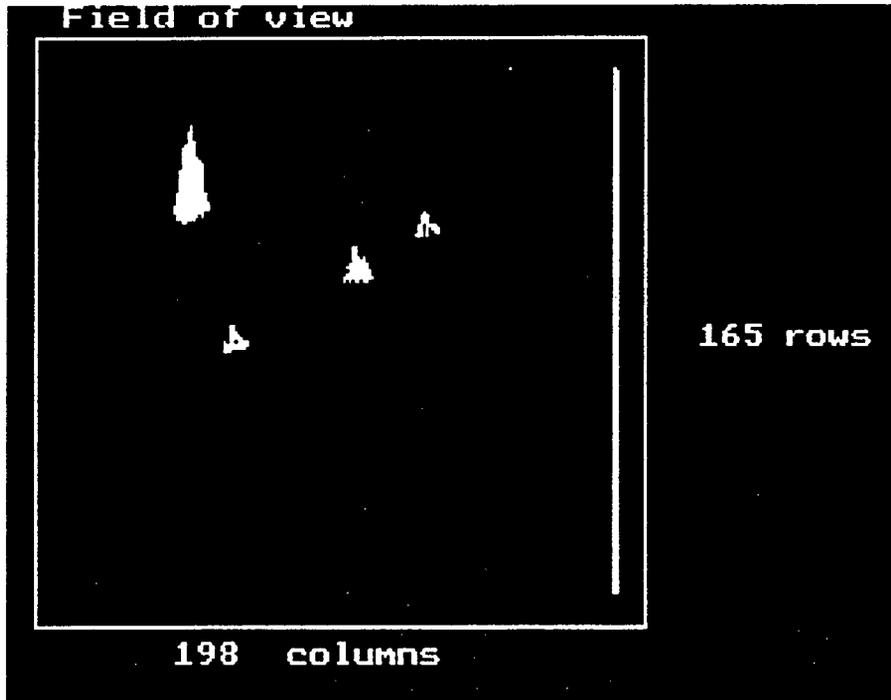


Fig. 6.9 Image with incorrect focus

The camera lens was chosen for image size (1/2 inch format) so that the image and CCD sizes matched. The standard lens with a diameter of only 1.5 cm was consequently much too small to observe real stars. A container with a number of holes drilled in the bottom was used to simulate a stellar FOV. The results are shown in Figures 6.10 to 6.12. The results of using a too slow clocking rate are manifest in the image smear. The noise pattern in the left hand side of the image is similar in all three images indicating poor electrical isolation between the CCD and the rest of the camera electronics.



**Fig. 6.10 Image with slow clock smear**

Although the image quality of these tests are not very refined the results show that a low cost CCD camera with the ability to take snap shots with varying integration times is a feasible proposition for the camera of a microsatellite CCD star sensor system.

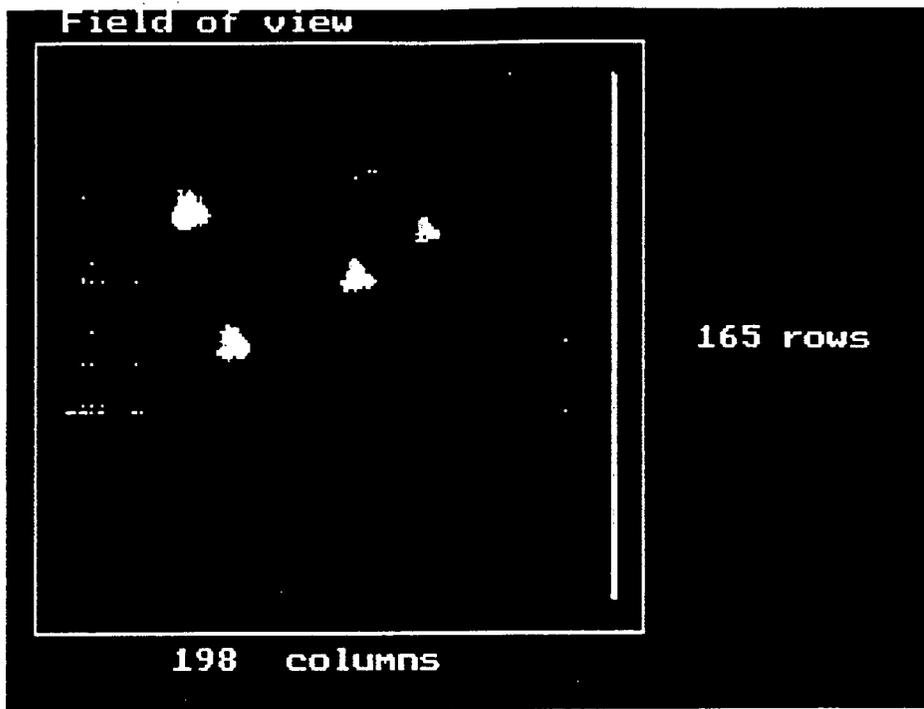


Fig. 6.11 Image with 2MHz clock

## 7. Summary and Conclusions

This paper covers all the most important aspects of the development of a CCD star sensor system for a low earth orbit microsatellite. While not claiming to be comprehensive and exhaustive, the study presents solutions to the problems of each system sub-section. Simulations and calculations conducted and included in the text and appendices show that such an attitude determination system is viable even with a limited budget and resources.

### 7.1 Summary

The different hardware and software sub-systems of a star sensor system were studied as the basis for the implementation of a star sensor attitude determination system on an actual microsatellite. It was found that a mapper type star sensor with no moving parts and a large FOV should be used. CCD technology chosen for the imaging device because of its qualities such as solid state design, high sensitivity, low image distortion and good spectral response to mention but a few. Commercially available HCT technology digital devices can be used for the camera electronics.

In Chapter 3 the satellite orbit was calculated to determine the section of the celestial sphere that the CCD camera would have in its FOV. Stellar data required for the star match routines were assessed. These include star catalogues, stellar distribution density and FOV size and dimension calculations. The responsivity of CCDs to stellar radiation was investigated and quantified. Camera lens diameter calculations were conducted using probable integration times, FOV sizes and stellar magnitude thresholds. Chapter 3 also includes a predicted output voltage versus stellar magnitude calculation for use in estimation of a star's brightness.

Chapter 4 dealt with the determination of star positions in the digital FOV. An FOV model was proposed where a stellar image is modeled as having a Gaussian distribution. Possible sources of error and correction procedure were investigated. An adaptation of a region growing algorithm was implemented for finding the stars in the FOV. Simulations were conducted to test the algorithm. The performance of a weighted centroid determination algorithm was investigated in terms of sensitivity to noise and distortion. Simulation programs were used to obtain and evaluate the results.

In Chapter 5 a pattern recognition star identification was studied. A model of the attitude determination problem was presented and a number of recognition algorithms evaluated. An adaptation of a star pattern recognition algorithm by Van Bezooijen was implemented in a Pascal program and tested for performance and robustness to errors. A technique for

calculation of the attitude once a stellar pair has been identified is also given. A discussion of the accuracy requirements on the yaw and roll axes for a particular pitch accuracy was also done.

A prototype camera was constructed using the TC211 CCD from Texas Instruments and TTL technology on a wire wrap board. Some tests were conducted by controlling the camera from a Pascal program on a PC.

## 7.2 Conclusions

The study has shown that the development and implementation of such a system is within reach of a small project team with limited resources. The results of tests conducted with the prototype camera and computer programs can be used to set down guidelines for an actual project.

The mapper type of star sensor configuration was proposed for this application. The camera is fixed on the top (-Z) facet of the satellite facing in the +Y direction and has no moving parts (other than a possible sun shutter). A solid state area CCD with a size of 512 by 512 pixels should be used for the imaging device. Such a CCD should preferably be radiation hardened and if enough funds are available of the back-illuminated type to improve sensitivity. The choice of CCD should be subject to the calculations in Chapter 3 for device sensitivity and frequency response.

The specification for the imager sensitivity is  $m_v = 6.5$  and an FOV of  $10^\circ$  by  $10^\circ$  in order to have at least 3 stars in the FOV at all times during the orbit. A lens diameter of 12 cm with a manually adjustable iris is specified for the star camera. Another requirement for the lens is that its image size should conform to the physical dimensions of the chosen CCD. Laboratory tests should be conducted before the launch to fix the lens's focus length for the amount of defocusing required to spread a stellar image over a grid of at least 9 (3 by 3) pixels. All mechanical parts of the star camera should be carefully selected for temperature gradient changes in the orbit and possible dimensional distortion during the launch.

The Yale star catalogue is suitable for the creation of a guide star database. This database should be segmented into 12 zones of  $46^\circ$  by  $16^\circ$  to speed up the processing during pattern recognition. Stellar magnitudes can be calculated beforehand for the particular CCD to provide a lookup table which gives the star magnitudes for corresponding output voltage levels.

The adapted region growing algorithm should be implemented to obtain the positions of stars in the FOV. It uses the dimensional characteristics of the stellar image model to search through only 11 % of the image. Some more tests should be conducted to find a

way of discriminating between stellar and planetary images. A suggestion is to use an accurate model of the defocused lens system and light distributions of stars and planets. Two dimensional correlation could then be used on the 5 by 5 grid of each stellar image to determine its conformation to the stellar image model. The centroid determination algorithm proposed here is good enough to provide positional accuracy to  $L/10$  detector lengths even with high noise levels and distortion.

The adapted Van Bezooijen pattern recognition algorithm is both rotation and translation invariant. It is very robust in terms of the interstellar distances, A suggestion for extra speed is to use the digitised  $N$  by  $N$  pixel FOV image directly for the pattern recognition. Only once FOV and Guide star pairs have been matched will it be necessary to calculate their accurate centroids. This would speed up the throughput of the algorithm considerably. An attitude calculation technique is presented which is fast and simple, and provides the three axis attitude of the satellite to the required accuracy.

Areas suggested for further study are:

- The development of more robust attitude determination algorithms and software.
- The use of more than one star sensor in order to obtain a higher attitude accuracy.
- The tracking of more complex celestial objects.

## Appendix A

Transformation of camera boresight vector from camera coordinates to celestial coordinates :

Euler angles are used. Coordinates are transformed from one axis system to another by a yaw, pitch and roll transformation around the first axis system following the z, x and y axes respectively. These three transformations are represented by the following three matrices :

$$[\varphi] = \begin{bmatrix} \cos \varphi & \sin \varphi & 0 \\ -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.1)$$

$$[\theta] = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \quad (A.2)$$

$$[\phi] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \quad (A.3)$$

The coordinate systems used are :

### **FOV coordinates**

The x-axis points to the right when looking at the FOV in the boresight direction. The y-axis points to the top. The z-axis points in the opposite direction from the boresight.

### **Local Level coordinates**

The x-axis points in the direction of spacecraft motion along its orbit. The y-axis points to the right of the x-axis and the z-axis to the earth's centre.

### **Earth coordinates**

The x-axis points in the direction of vernal equinox,  $\Upsilon$ , with the y-axis perpendicular to it. The z-axis points in the direction of the earth's north pole (spinning).

### **Celestial coordinates**

These are as defined in geocentric astronomy. There are two coordinates, Right Ascension (RA) and declination (dec). RA is measured in an anti-clockwise rotation from the vernal equinox along the plane of the equator. dec is measured from the

equator to the North and South poles of the earth. The ranges of these coordinates are

$$0 \leq RA \leq 360^\circ$$

$$-90^\circ \leq dec \leq +90^\circ$$

The boresight vector, in *FOV* coordinates, is :

$$R_{FOV} = \begin{bmatrix} x_{FOV} \\ y_{FOV} \\ z_{FOV} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad (A.4)$$

From *FOV* to *Local Level* coordinates :

$$\begin{aligned} R_L &= [\phi_L] \cdot [\theta_L] \cdot [\varphi_L] R_{FOV} \\ &= M_L \cdot R_{FOV} \end{aligned} \quad (A.5)$$

where :

$$\phi_L = \pi$$

$$\theta_L = -\frac{\pi}{2}$$

$$\varphi_L = 0$$

which gives :

$$[\varphi_L] = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.6)$$

$$[\theta_L] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.7)$$

$$[\phi_L] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (A.8)$$

which gives,

$$M_L = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (A.9)$$

From *Local Level* to *Earth* coordinates :

$$R_E = M_E \cdot R_L \quad (A.10)$$

In this case the local level axis system is moving relative to the solar axis system as the satellite orbits the earth. For a circular Keplerian orbit a number of parameters are used to do this transformation. They are :

$u$  = True angular nodal elongation (The positive angle orientated according to the satellite motion between Earth centre to ascending node axis and the Earth centre to satellite axis,  $u = \nu + \omega$ , where  $\nu$  is the true anomaly (the angle measured at the barycenter between the perigee point and the satellite) and  $\omega$  is the argument of perigee (the angle at the barycenter measured in the orbital plane in the direction of the satellites motion from the ascending node to perigee.)(Wertz, 1986))

$i$  = The inclination of the satellite's orbit relative to the equatorial plane

$\Omega$  = The right ascension of the ascending node of the orbit

First the Local Level axes are rotated around the y-axis through  $u + \frac{\pi}{2}$ . Next a rotation around the z-axis of  $(\beta + \frac{\pi}{2} - \Omega)$ , in the plane of the equator, brings the local level +y axis to the direction of vernal equinox.

The coordinate system is then rotated around the x-axis through  $\frac{\pi}{2} - i$  to a plane perpendicular to the earth's equator.

$$\begin{aligned} \varphi_{E0} &= 0 & \varphi_E &= (\beta + \frac{\pi}{2} - \Omega) \\ \theta_{E0} &= u + \frac{\pi}{2} & \theta_E &= 0 \\ \phi_{E0} &= \frac{\pi}{2} - i & \phi_E &= 0 \end{aligned}$$

which gives :

$$[\varphi_{E0}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.11)$$

$$[\theta_{E0}] = \begin{bmatrix} \cos\left(u + \frac{\pi}{2}\right) & \sin\left(u + \frac{\pi}{2}\right) & 0 \\ -\sin\left(u + \frac{\pi}{2}\right) & \cos\left(u + \frac{\pi}{2}\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.12)$$

$$[\phi_{E0}] = \begin{bmatrix} \cos\left(\frac{\pi}{2} - i\right) & 0 & -\sin\left(\frac{\pi}{2} - i\right) \\ 0 & 1 & 0 \\ \sin\left(\frac{\pi}{2} - i\right) & 0 & \cos\left(\frac{\pi}{2} - i\right) \end{bmatrix} \quad (A.13)$$

$$[\phi_E] = \begin{bmatrix} \cos(\beta + \frac{\pi}{2} - \Omega) & \sin(\beta + \frac{\pi}{2} - \Omega) & 0 \\ -\sin(\beta + \frac{\pi}{2} - \Omega) & \cos(\beta + \frac{\pi}{2} - \Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.14)$$

$$[\theta_E] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.15)$$

$$[\phi_E] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (A.16)$$

which gives,

$$\begin{aligned}
 M_E &= [\phi_E][\theta_E][\varphi_E][\phi_{E0}][\theta_{E0}][\varphi_{E0}] \\
 &= \begin{bmatrix}
 \cos\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) - \sin\left(u+\frac{\pi}{2}\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) + \cos\left(u+\frac{\pi}{2}\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) & -\sin\left(\frac{\pi}{2}-i\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) \\
 -\cos\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) - \sin\left(u+\frac{\pi}{2}\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) & -\sin\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) + \cos\left(u+\frac{\pi}{2}\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) \\
 \cos\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right) & \cos\left(\frac{\pi}{2}-i\right) \\
 \sin\left(u+\frac{\pi}{2}\right)\sin\left(\frac{\pi}{2}-i\right) & & 
 \end{bmatrix} \quad (A.17)
 \end{aligned}$$

The complete transformation from *FOV* to *Earth* coordinates is then :

$$\begin{aligned}
 R_C &= M_E \cdot M_L \cdot R_{FOV} \\
 &= M_{Total} \cdot R_{FOV} \quad (A.18)
 \end{aligned}$$

where,

$$M_{Total} = \begin{bmatrix}
 -\cos\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) + \sin\left(u+\frac{\pi}{2}\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(\frac{\pi}{2}-i\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) & -\sin\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) - \cos\left(u+\frac{\pi}{2}\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) \\
 \cos\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) + \sin\left(u+\frac{\pi}{2}\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) & -\sin\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) & \sin\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) - \cos\left(u+\frac{\pi}{2}\right)\cos\left(\beta+\frac{\pi}{2}-\Omega\right) \\
 -\cos\left(u+\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}-i\right)\sin\left(\beta+\frac{\pi}{2}-\Omega\right) & -\cos\left(\frac{\pi}{2}-i\right) & -\sin\left(u+\frac{\pi}{2}\right)\sin\left(\frac{\pi}{2}-i\right)
 \end{bmatrix} \quad (A.19)$$

From *earth* to *celestial* coordinates :

$$RA = \text{atan}\left(\frac{y_E}{x_E}\right) \quad (A.20)$$

$$dec = \text{asin}(z_E) \quad (A.21)$$

From (A.18), (A.19), (A.20) and (A.21) :

$$RA = \text{atan}\left(\frac{xFOV \cdot (-\cos\left(u + \frac{\pi}{2}\right) \cos\left(\frac{\pi}{2} - i\right) \cos\left(\beta + \frac{\pi}{2} - \Omega\right) + \sin\left(u + \frac{\pi}{2}\right) \sin\left(\beta + \frac{\pi}{2} - \Omega\right) + yFOV \cdot (\sin\left(\frac{\pi}{2} - i\right) \cos\left(\beta + \frac{\pi}{2} - \Omega\right)) + zFOV \cdot (-\sin\left(u + \frac{\pi}{2}\right) \cos\left(\frac{\pi}{2} - i\right) \cos\left(\beta + \frac{\pi}{2} - \Omega\right) - \cos\left(u + \frac{\pi}{2}\right) \sin\left(\beta + \frac{\pi}{2} - \Omega\right)}{xFOV \cdot (\cos\left(u + \frac{\pi}{2}\right) \cos\left(\frac{\pi}{2} - i\right) \sin\left(\beta + \frac{\pi}{2} - \Omega\right) + \sin\left(u + \frac{\pi}{2}\right) \cos\left(\beta + \frac{\pi}{2} - \Omega\right)) + yFOV \cdot (-\sin\left(\frac{\pi}{2} - i\right) \sin\left(\beta + \frac{\pi}{2} - \Omega\right)) + zFOV \cdot \sin\left(u + \frac{\pi}{2}\right) \cos\left(\frac{\pi}{2} - i\right) \sin\left(\beta + \frac{\pi}{2} - \Omega\right) - \cos\left(u + \frac{\pi}{2}\right) \cos\left(\beta + \frac{\pi}{2} - \Omega\right)}\right) \quad (A.22)$$

$$dec = \left( xFOV \cdot (-\cos\left(u + \frac{\pi}{2}\right) \cos\left(\frac{\pi}{2} - i\right) \sin\left(\beta + \frac{\pi}{2} - \Omega\right)) + yFOV \cdot (-\cos\left(\frac{\pi}{2} - i\right)) + zFOV \cdot (-\sin\left(u + \frac{\pi}{2}\right) \sin\left(\frac{\pi}{2} - i\right)) \right) \quad (A.23)$$

## Appendix B

This appendix contains Matlab script files which were used to generate graphs and results. These files are listed by chapter number.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : bs_path.m
%
% Description  : This program calculates the position of the boresight
%               direction of the star sensor camera on the celestial
%               sphere. The boresight vector is :
%
%               RFOV = [0 0 -1]'
%
%               Coordinate transformations are done from (Field Of View)
%               (FOV) to Local Level (LL) to Earth (E) to Celestial (C)
%               coordinates. The longitude of the ascending node is
%               also rotated through 360 deg to simulate a complete
%               rotation of the earth around the sun (1 year).
%
%               The graph of the boresight path is plotted on a Sanson-
%               Flamsteed projection along with all the stars in the
%               Yale Bright Star Catalogue. The direction of the center
%               of the Milky Way Galaxy and the path of the ecliptic are
%               also given for reference.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear memory and screen

clg;
clc;
hold off;
clear;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Constants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Inclination of earth equator to plane of the ecliptic
Earth_i = (23.44/180)*pi;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Inclination of satellite orbit (Orbit_i)
% It is defined as the angle between the lines formed by
% the line between the north pole and the geocenter and
% the line from the geocenter perpendicular to the plane
% of the satellite orbit
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Orbit_i = (98.6/180)*pi;

% The angle between the sun and the orbital plane
% 30 deg for daylight viewing conditions
Sun_Angle = (30/180)*pi;

j = 1;
R = (360)/(2*pi);
Omega_Res = (10/180)*pi;
Spin_Res = (30/180)*pi;

```

```

##### Conventions #####
%
%   yaw   = psi;
%   pitch = theta;
%   roll  = phi;
%
%   rotation order => 1. yaw   around z-axis
%                   2. pitch around y-axis
%                   3. roll  around x-axis
%
%
#####

disp('Boresight Path on Celestial Sphere');
disp('-----');
disp(' ');

#####
%
%   FOV coordinates:
%
%           +y points upward
%           +x points to the right
%           +z points in the opposite
%           direction to the boresight
%
%           +y      -z (boresight)
%           |      /
%           |     /
%           |----+x
%           /
%          +z
%
%   The boresight points in the -z direction of the
%   FOV axis system. The x,y plane is the FOV plane.
%
#####

Rbs      = input('Camera Boresight direction in FOV Coordinates [x;y;z] :
');

Spin_Res = input('Resolution of orbit rotation (deg): ');
Spin_Res = (Spin_Res/180)*pi;

Omega_Res = input('Resolution of sun rotation (deg): ');
Omega_Res = (Omega_Res/180)*pi;

#####
%
%   Get input for different angle mounting of
%   sensor on satellite body.
%
#####

psi1   = input('Sensor psi angle   (around +z) : ');
thetal = input('Sensor theta angle (around +y) : ');
phi1   = input('Sensor Phi angle   (around +x) : ');
%disp('');
% input for body not aligned to local level
%psi2   = input('Body psi angle to local level : ');
%theta2 = input('Body theta angle to local level : ');
%phi2   = input('Body phi angle to local level : ');
%disp('');

psi1 = (psi1/180)*pi;
thetal = (thetal/180)*pi;
phi1 = (phi1/180)*pi;

[Ayaw,Apitch,Aroll] = mrot(psi1,thetal,phi1);

```

```

Rbs = Aroll*Apitch*Ayaw*Rbs;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Sensor to Body Coordinates
%
% Body coordinates:
%
%      +x points to the front of the
%          satellite (direction of motion)
%      +z points to the top
%      +y points to the left if the satellite
%          is viewed from the top
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

psiB = pi;
thetaB = 0;
phiB = pi/2;

[AyawB,ApitchB,ArollB] = mrot(psiB,thetaB,phiB);

Rb = ArollB*ApitchB*AyawB*Rbs;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Body to Local Level Coordinates
%
% Local Level Coordinates:
%
%      +x points in the direction of the satellite motion
%          along its orbit
%      +z points to the centre of the earth
%      +y points to the right of the +x direction if the
%          z-axis points downwards (this is also the boresight
%          direction)
%
%          +x (direction of motion)
%          /
%          /
%      ----- +y (boresight direction)
%          |
%          |
%          +z (to earth centre)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

psiLL = 0;
thetaLL = 0;
phiLL = pi;

[AyawLL,ApitchLL,ArollLL] = mrot(psiLL,thetaLL,phiLL);

RLL = ArollLL*ApitchLL*AyawLL*Rb;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Rotate Omega for one Earth Orbit of Sun %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for Omega = 0:Omega_Res:( (360+(Omega_Res/pi*180) )/180)*pi;

% spin once around orbit

for spin = 0:Spin_Res:(360/180)*pi;

    psiSpin = 0;
    thetaSpin = spin;
    phiSpin = 0;

    [AyawSpin,ApitchSpin,ArollSpin] = mrot(psiSpin,thetaSpin,phiSpin);

    Rspin = AyawSpin*ArollSpin*ApitchSpin*RLL;

```

```

*****
%
% Local Level to Earth Coordinates
%
% Earth Coordinates:
%
%   +y points to vernal equinox.
%   +z points in the direction of the north pole of
%       the earth.
%   +x points to the right of the +y direction if the
%       z-axis points upwards
%
%
%   (to north pole of earth)
%
%           +z
%           | / +y (vernal equinox)
%           | /
%           |----- +x
%
*****

psiE = -(Orbit_i-(pi/2));
thetaE = 0;
phiE = 0;

[AyawE,ApitchE,ArollE] = mrot(psiE,thetaE,phiE);

Omega/pi*180;
RE = ArollE*ApitchE*AyawE*Rspin;

psiE = 0;
thetaE = 0;
phiE = Sun_Angle+(pi/2)-Omega;

[AyawE,ApitchE,ArollE] = mrot(psiE,thetaE,phiE);

Omega/pi*180;

RE = ArollE*ApitchE*AyawE*RE;

*****
%
% The Ecliptic vector points to the sun and
% RECL is in sun coordinates with the ecliptic
% vector pointing in the +y direction.
%
*****

RECL = [0;1;0];

psiECL1 = -Omega;
thetaECL1 = 0;
phiECL1 = 0;

[AyawECL1,ApitchECL1,ArollECL1] = mrot(psiECL1,thetaECL1,phiECL1);

RECL1 = ArollECL1*ApitchECL1*AyawECL1*RECL;

psiECL2 = 0;
thetaECL2 = -Earth_i;
phiECL2 = 0;

[AyawECL2,ApitchECL2,ArollECL2] = mrot(psiECL2,thetaECL2,phiECL2);

RECL2 = ArollECL2*ApitchECL2*AyawECL2*RECL1;

```

```

#####
%
% Earth to Celestial Coordinates
%
% Celestial Coordinates:
%
%           RA is measured from the +y-axis of the Earth
%           coordinate system.
%           dec is the angle measured from the x,y plane
%           of the Earth coordinate system
%
% Looking from earth north pole, +z axis:
% (R is a unit vector in the direction of
% the boresight vector. R is not in the x,y plane)
%
%           +y (Earth)
%           |
%           |-----|
%           | \      \
%           |  \      \ RA
%           |   \      \
%           |    R     \
%           |-----|
%           |-----|----- +x (Earth)
%
%           RA = atan ( - )
%                       Y
%
% Looking perpendicularly at R:
%
%           +z (Earth) (North pole direction)
%           |
%           | \      \
%           |  \      \ R
%           |   \      \
%           |    \      \
%           |     \      \
%           |      dec   \
%           |-----|
%           |-----|-----
%           x,y plane in Earth coordinates
%
#####

    alfa    = atan2(-RE(1),RE(2));
    delta   = asin(RE(3));

    alfa_E  = atan2(-RECL2(1),RECL2(2));
    delta_E = asin(RECL2(3));

    RA_Ecliptic = alfa_E/pi*180;
    dec_Ecliptic = delta_E/pi*180;

    if alfa < 0;
        alfa = alfa + 2*pi;
    end;

    if alfa_E < 0;
        alfa_E = alfa_E + 2*pi;
    end;

    RA_bs(j)    = (alfa/pi)*180;
    Dec_bs(j)   = (delta/pi)*180;

    RA_E(j)     = (alfa_E/pi)*180;
    Dec_E(j)    = (delta_E/pi)*180;

    j = j + 1;

axis([0 360 -90 90]);
plot(RA_bs,Dec_bs,'.g');
hold on;
plot(RA_E,Dec_E,'*w');
title('Ecliptic (white) & Boresight (green)');
hold on;

end;

```

## Appendix B Program Listings

```

end;

text(0.2,0.3,'Press any key','sc');
pause;
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Sort RA & Dec for Plotting %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[RA_bs,I]      = sort(RA_bs);
Dec_bs        = Dec_bs(I);

[RA_E,IE]     = sort(RA_E);
Dec_E        = Dec_E(IE);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Get stars from Yale Catalogue %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load yalecat.mat;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Grid for RA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

i = 1;

Dec_Grid0 = [-90/180*pi:1/180*pi:90/180*pi];

for j = -180:45:180;
    RA_Grid(i,:) = j/180*pi*ones(1,length(Dec_Grid0));
    Dec_Grid(i,:) = Dec_Grid0;
    i = i + 1;
end;

RA_Grid_Lat0 = [-180/180*pi 180/180*pi];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Grid for Dec %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

i = 1;

for j = -90:15:90;
    RA_Grid_Lat(i,:) = RA_Grid_Lat0;
    Dec_Grid_Lat(i,:) = [j/180*pi j/180*pi];
    i = i + 1;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Shift coordinates for chart %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

axis([0 360 -90 90]);
plot(RA_bs,Dec_bs,'*g');
hold on;
plot(RA_E,Dec_E,'*w');
grid;
title('Ecliptic (white) & Boresight (green)');

pause;

clf;
hold off;

s = input('Save boresight path ? (y/n) : ','s');

if s == 'y';
    save bs_path RA_bs Dec_bs RA_E Dec_E;
    disp('Saved boresight path as bs_path.mat');
end;

for a = 1:length(RA_bs);
    if RA_bs(a) < 180;
        RA_bs_A = a;
    end;
    if RA_E(a) < 180;
        RA_E_A = a;
    end;
end;

```

```

end;

RA_bs = [- (RA_bs(1:RA_bs_A)-180)-180 -RA_bs(RA_bs_A+1:length(RA_bs))+360];
RA_E = [- (RA_E(1:RA_E_A)-180)-180 -RA_E(RA_E_A+1:length(RA_E))+360];
RAMap = [- (RAMap(1:4608)-180)-180; -RAMap(4609:9094)+360];

[RA_bs,I] = sort(RA_bs);
Dec_bs = Dec_bs(I);

[RA_E,IE] = sort(RA_E);
Dec_E = Dec_E(IE);

for a = 1:length(RA_bs);
    if RA_bs(a) < 180;
        RA_bs_A = a;
    end;
end;

%%% For plotting a continuous curve with Matlab %%%%%%%%%%%%%%%

RA_bs = [RA_bs(1:RA_bs_A)';-180;RA_bs(RA_bs_A+1:length(RA_bs))']';
Dec_bs = [Dec_bs(1:RA_bs_A)';Dec_bs(RA_bs_A+1);Dec_bs(RA_bs_A+1:length(Dec_bs))']';

RA_E = [-180;RA_E(1:length(RA_E))']';
Dec_E = [Dec_E(length(Dec_E))';Dec_E(1:length(Dec_E))']';

%%%%%%%%%%%%%% Convert to Radians %%%%%%%%%%%%%%%

RA_bs = RA_bs/180*pi;
Dec_bs = Dec_bs/180*pi;

RA_E = RA_E/180*pi;
Dec_E = Dec_E/180*pi;

RAMap = RAMap/180*pi;
Decmap = Decmap/180*pi;

%%%%%%%%%%%%%% Sanson Flamsteed mapping %%%%%%%%%%%%%%%

x_bs = R*RA_bs.*cos(Dec_bs);
y_bs = R*Dec_bs;

x_ecliptic = R*RA_E.*cos(Dec_E);
y_ecliptic = R*Dec_E;

x_Grid_long = (R*RA_Grid.*cos(Dec_Grid));
y_Grid_long = R*Dec_Grid;

x_Grid_Lat = (R*RA_Grid_Lat.*cos(Dec_Grid_Lat));
y_Grid_Lat = R*Dec_Grid_Lat;

x_stars = (R*RAMap.*cos(Decmap));
y_stars = R*Decmap;

%%%%%%%%%%%%%% Plot results %%%%%%%%%%%%%%%

hold off;
clg;

plot(x_stars,y_stars,'w. ');
hold on;

plot(x_Grid_long',y_Grid_long', '-w');
plot(x_Grid_Lat',y_Grid_Lat', '-w');
plot(x_bs,y_bs,'g-');
plot(x_ecliptic,y_ecliptic,'w-');
xlabel('RA (deg)');
ylabel('Dec (deg)');
title('Ecliptic (white) & Boresight (green)');

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : mrot.m
% Description   : Matrix rotation function called by bs_path.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Ayaw,Apitch,Aroll] = mrot(psi,theta,phi)

Ayaw = [ cos(psi)  sin(psi)  0
        -sin(psi)  cos(psi)  0
          0         0         1];

Apitch = [ cos(theta)  0  -sin(theta)
           0            1  0
           sin(theta)  0  cos(theta)];

Aroll = [ 1  0  0
          0  cos(phi)  sin(phi)
          0 -sin(phi)  cos(phi)];
```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : dense_01.m
%
% Description  : This program calculates the stellar distribution
%               density
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear memory and screen %%%%%%%%%%

clear;
clg;
hold off;

load yalecat.mat;

mag_limit = 9.0;

D = zeros(33/2,364/4);

Ls = length(RAmap);

for i = 1:Ls;
    if magmap(i) <= mag_limit;
        if Decmap(i) <= 8.6+15;
            if Decmap(i) >= 8.6-15;
                D(Decmap(i)/2+8/2, RAmap(i)/4+1) =
D(Decmap(i)/2+8/2, RAmap(i)/4+1)+1;
            end;
        end;
    end;
end;

disp('OK. ');

load bs_path.mat;

RA_bs = [RA_bs';360]';
Dec_bs = [Dec_bs';8.6]';

clg;
subplot(211);
axis([0 360 8.6-15 15+8.6]);
plot(RAmap, Decmap, 'w');
hold on;
plot(RA_bs, Dec_bs, 'w-');
plot(RA_E, Dec_E, 'w-');
hold off;
axis([0 360 8.6-15 15+8.6]);
mesh(D(11:-1:1,:),.^2,[17 50]);

```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : dense_02.m
%
% Description  : This program calculates the stellar distribution density
%               of stars along the bore sight path on the celestial
%               sphere. The path is calculated by bs_path.m and used
%               by this program.
%
%               The input parameters for this program are the magnitude
%               limit, FOV size and resolution for the density
%               calculation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear memory and screen %%%%%%%%%%%%%%%

clear;
clg;
clc;
hold off;

%theta = 0:10:360;

more = 'y';

while more ~= 'n';

load yalecat.mat;
disp('Loading data...');

tel = 0;
clc;
clg;
axis('normal');
clear Star;
x = 1;

mv      = input('magnitude : ');
RAwidth = input('RA Width : ');
marker0 = input('Resolution : ');
angleI0 = input('Orbit inclination angle : ');
marker  = 0;

clc;
disp('');
disp('Just a sec...');
disp('');

BoreSight = angleI0-90;

star0 = 0;

j = 0;

while marker < (360+(RAwidth/2)-marker0) & (x < length(RAmap));
    j = j+1;
    y = 0;
    while (RAmap(x) <= (marker+(RAwidth/2))) & (x < length(RAmap));
        if magmap(x) <= mv;
            if Decmap(x) <= BoreSight + (RAwidth/2);
                if Decmap(x) >= BoreSight - (RAwidth/2);
                    if (RAmap(x) > (marker-(RAwidth/2)));
                        star0 = star0+1;
                    end;
                end;
            end;
        end;
        x = x + 1;
        y = y + 1;
    end;
end;

```

## Appendix B Program Listings

```

Star = [Star;star0];
RA = [RA;marker];

if round(10*marker/360) > 10*marker/360;
    disp(y);
end;

x = x-(5*(marker0/360*length(RAmap)));

if x < 1;
    x = 1;
end;

star0 = 0;
marker = marker + marker0;
end;

clear magmap;

disp('nou');

%Dense = Star./((1/(360/marker0))*14400);

%clear Star;

[mini,i] = min(Star(2:length(Star)-1));

plot(RA(1:length(Star)-1),Star(1:length(Star)-1),'-');
title('Stellar distribution density');
ylabel('Stars');
xlabel('Deg portions');
text(0.4,0.015,num2str(marker0),'sc');
text(0.5,0.49,'min = ','sc');
text(0.6,0.49,num2str(mini),'sc');
text(0.7,0.49,'at','sc');
text(0.75,0.49,num2str(i*marker0),'sc');
text(0.8,0.49,'Deg','sc');
text(0.6,0.55,num2str(mv),'sc');
text(0.5,0.55,'mag = ','sc');
text(0.6,0.52,num2str(angleI0),'sc');
text(0.5,0.52,'i = ','sc');
text(0.7,0.52,'Deg','sc');
grid;

pause;
clc;
s = input('Save graph? : ','s');

if s == 'y';
    f = input('Filename : ','s');
    disp('Saving...');
    meta f;
end;

more = input('More : ','s');

clear RA;
clear Star;

clc;
clg;

end;

end;

```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : ccdmag.m
%
% Description  : This program calculates the CCD visual magnitudes of
%               stars.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear memory and screen %%%%%%%%%%%%%%%

clc;
clg;
clear;
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Equations for output voltage calculation
%
%   Vout = qNeG
%         ----- (Volt)
%         Co
%
%   Ne_Total = Ne_Rad + Ne_Noise (electrons)
%
%           / oo
%   Ne_Rad = | E_eLambda
%           | ----- . Eta_Lambda dLambda (electrons per m^2 per second)
%           |   hv
%           / 0
%
%
%   v = -----
%       Lambda
%
%   Ne_Noise = Ne_Photon + Ne_DarkSignal + Ne_DSNU + Ne_Reset
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clg;
clear;
hold off;

% Flux density distribution for 0th mag star of type A0V (in W/cm^2.um)

Ee0_Lambda = [6.6e-12
              3.6e-12
              2.25e-12
              1.22e-12
              2.92e-13];

Ee0_Lambda = Ee0_Lambda * 100 * 100; % Convert to W/m^2.um

% Get visual magnitude of star

mv = input('Star visual magnitude (mv) : ');

k = (100 ^ (1 / 5)) ^ (-mv); % Factor for mv magnitude star

Ee_Lambda = k * Ee0_Lambda;

% Wavelength in micro meter

Lambda = [0.44
          0.55

```

## Appendix B Program Listings

```

0.64
0.79
1.25];

% Delta wavelength in micro meter for numerical integration
dLambda = [0.1
            (0.55 - 0.44)
            (0.64 - 0.55)
            (0.79 - 0.64)
            (1.25 - 0.79)];

% Quantum Efficiency of CCD (For TI TC211 taken from databook)
Eta_Lambda = [0.2
              0.4
              0.5
              0.38
              0.05];

q = 1.6e-19;      % Electron charge in Coulomb
G = 0.7;         % Output amp gain for source follower MOS transistor
C0 = 0.1e-12;    % Output capacitance in Farad
h = 6.626e-34;  % Planck constant in J s
c = 3e8;        % Speed of light in m/s
L_PixelX = 10e-6; % X-Dimension of a pixel in m
L_PixelY = 10e-6; % Y-Dimension of a pixel in m
N_Pixel_Total = 25; % Total number of pixels that are illuminated
A_Pixel = L_PixelX * L_PixelY; % Area of a pixel in m

ti_max = input('Max. integration time (s) : ');

Vt = [];
VtTI = [];

% Calculate 5 integration time values up to the maximum time
for ti = ti_max / 10:ti_max / 10:ti_max;
    Vout = [];
    for D = 0:0.01:0.12;
        A_Lens = pi * (D / 2)^2; % Area of lens
        v = c ./ (Lambda * 1e-6);
        Ne_Rad = (A_Lens / (A_Pixel * N_Pixel_Total)) * sum((Ee_Lambda .*
Eta_Lambda .* dLambda .* Lambda) ./ (h .* v));
        Ne_Total = Ne_Rad; % Voltage only from stellar radiation
        Vout = [Vout; A_Pixel * ti * ((q * Ne_Total * G) / C0)];
    end;
    Vt = [Vt; Vout'];
end;

plot(0:0.01:0.12, Vt, 'g');
xlabel('Lens Diameter (m)');
ylabel('Output Voltage (V)');
grid;

pause;

mesh(Vt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Constants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
F0 = [6.6e-12
      3.6e-12
      2.25e-12
      1.22e-12
      2.92e-13];

```

## Appendix B Program Listings

```

lambda = [0.44
          0.55
          0.64
          0.79
          1.25];

dlam = [0.1
        0.55-0.44
        0.64-0.55
        0.79-0.64
        1.25-0.79];

QE = [0.1
      0.28
      0.4
      0.38
      0.05];

TE = 1.0;
Lt = 1.0;
Np = 10;
ti = 0.5;
Vs = 1.4e-6;
h = 6.626e-34;
c = 3e8;

%Ne = (QE*1.0*(2.5e-12/2.84e-19)*0.4)
NeB = 1e-6*sum(F0.*QE.*dlam.*lambda)/(h*c);

Nem = NeB*100*100

Vout = (0:0.01:0.12)';

%%%%%%%%%%%% Calculate Vout for integration time ti %%%%%%%%%%%%%

for ti = 0:0.001:0.1;
    clear Vout;
    for D = 0:0.01:0.12;
        A = pi*(D/2)^2;
        Ne = Nem*A*4.36;
        NE = [NE;Ne];
        Vout = [Vout;((Ne*ti*Vs)/Np)];
    end;
    Vt = [Vt;Vout'];
    disp(ti);
end;

```

```
***** Plot results ****  
plot(0:0.01:0.12,Vt(length(Vout),:),'g');  
xlabel('Lens Diameter (m)');  
ylabel('Output Voltage (V)');  
grid;  
  
pause;  
  
mesh(Vt);  
  
for mv = -2:0.5:6.5;  
    Vtot = [Vtot;Vs*Nem*((2.65e-6*(100^(1/5))^-mv)/(2.65e-6))];  
end;  
  
plot(-2:0.5:6.5,Vtot);  
grid;
```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : ag.m
%
% Description : This program calculates Airy and Gaussian point spread
%              functions and determines a value of sigma for the
%              Gaussian representation that most closely follows that
%              of the Airy for the central region of the point spread
%              function.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear the memory & screen %%%%%%%%%%

clear;
hold off;
clg;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Constants %%%%%%%%%%

x = 0.1:0.1:5;
%sigma = 1.3535;
%sigma = 1.3246;
Error = size(1.3497:0.00001:1.3499);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculate the Airy & Gaussian functions %%%%%%%%%%

i = 0;

for sigma = 1.3497:0.00001:1.3499;
    I_Gauss = exp(-(x.*x)./(2*sigma*sigma));
    I_Airy = ((2*(bessel(1,x)))./(x)).^2;
    Err = I_Gauss-I_Airy;
    i = i + 1;
    Err_max = max(Err);
    Err_min = min(Err);
    Error(i) = abs(Err_max-Err_min);
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot the Airy & Gaussian functions %%%%%%%%%%

plot(x,I_Gauss);
hold;
plot(x,I_Airy);
grid;

pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot the Error between Airy & Gaussian %%%%%%%%%%

clg;
plot(x,Err);
grid;

pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot the Error vs sigma %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% for best value of sigma %%%%%%%%%%

clg;
plot(1.3497:0.00001:1.3499,Error);
grid;

```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : grow.m
%
% Description  : This program simulates the FOV and finds
%               the stars in it.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear Environment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

echo off;
clear;
clg;
clc;
hold off;
rand('seed',11);
rand('uniform');
axis('normal');
format short;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Constants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

X_Max = 192;
Y_Max = 165;

P_X = 13.75e-6;
P_Y = 16e-6;

FOV_Gauss = zeros(Y_Max,X_Max);

Max_Stars = 20;
I0        = 10;
rho       = 1.5;
a         = 0.75;
D         = 0.03;
lambda    = 555e-9;
width     = 5;
s_width  = 5;
n_algo    = 3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Input Parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bits = input('Sampling Bits : ');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Choose Random Stellar Positions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i = 1:Max_Stars;
    rand;
    xr = rand*10;
    yr = rand*10;
    if xr < (10*10/192);
        xr = 10.5*10/192;
    end;
    if yr < (10*10/165);
        yr = 10.5*10/165;
    end;

    if xr > (150*10/192);
        xr = 150.5*10/192;
    end;
    if yr > (150*10/165);
        yr = 150.5*10/165;
    end;

    Stars_r = [Stars_r;[xr,yr]];

    xp = round((xr/10)*192);
    yp = round((yr/10)*165);
end;

```

```

xpr = rem(xr, (1/19.2));
ypr = rem(yr, (1/16.5));

if (xp-(xr/10*192)) < 0;
    xp = xp + 1;
end;
if (yp-(yr/10*165)) < 0;
    yp = yp + 1;
end;

Stars_p = [Stars_p;[xp,yp]];
delta_x = (xr-((xp-0.5)/192*10))/10*192;
delta_y = (yr-((yp-0.5)/165*10))/10*165;
Delta_X = [Delta_X;delta_x];
Delta_Y = [Delta_Y;delta_y];
P_Size = 5;

sigma = 1+round(rand);

SIGMA = [SIGMA;sigma];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Gaussian Stellar image model %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for xx = -P_Size:P_Size;
    for yy = -P_Size:P_Size;
        x = xx - delta_x;
        y = yy + delta_y;
        I_Gauss = 10*exp(-((x^2)+(y^2))/(2*sigma^2));
        FOV_Gauss(Y_Max-(yp+yy),xp+xx) = I_Gauss;
    end;
end;

disp(i);
end;

disp('Stars have been placed');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot FOV %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%axis('square');
%contour(FOV_Gauss);
%title('8 Stars in FOV');
%grid;
%pause;
%
%keep = input('Save Screen ? : (y/n) ','s');
%if keep == 'y';
%    meta gfov1;
%end;
%
%subplot(121);
%
%x1 = Y_Max-(yp-width);
%x2 = Y_Max-(yp+width);
%y1 = xp-width;
%y2 = xp+width;
%
%mesh(FOV_Gauss(x1:-1:x2,y1:y2));
%title('Gaussian Model');
%contour(FOV_Gauss(x1:-1:x2,y1:y2));
%title('Contour Plot');
%grid;
%
%pause;
%
%keep = input('Save Screen ? : (y/n) ','s');
%if keep == 'y';
%    meta g1;
%end;
%

```

## Appendix B Program Listings

```

##### Add Noise #####
t = 1:(s_width*2)+1;
rand('normal');

##### Main loop for b stars #####
disp('Starting main loop');

for b = 1:Max_Stars;

    disp(b);

    Star0 = FOV_Gauss(Y_Max-(Stars_p(b,2)-s_width):-1:Y_Max-
(Stars_p(b,2)+s_width), Stars_p(b,1)-s_width:Stars_p(b,1)+s_width);

    for i = 1:length(Star0);
        n = 0.0*rand(t);
        Star(i,:) = (n.*Star0(i,:)+Star0(i,:));
    end;

    S = size(Star);

##### Sample image for number of bits #####

Out = 0;
Smax = (2^bits);
step = I0/Smax;

for i = 1:S(1,1);
    for j = 1:S(1,2);
        k = Smax;
        while Out == 0;
            k = k-1;
            if (Star(i,j)/step) < 0.5;
                Star_d(i,j) = 0;
                Out = 1;
            end;
            if (Star(i,j)/step) >= k;
                Star_d(i,j) = k;
                Out = 1;
            end;
            if k == 1;
                Out = 1;
            end;
        end;
        Out = 0;
    end;
end;

##### Show star #####
%
%clg;
%hold off;
%subplot(121);
%mesh(Star);
%title('Star');
%
%mesh(Star_d);
%title('Digitized Star');
%
%pause;
%

end;

```

## Appendix B Program Listings

```

##### Find stars in the FOV #####
n = 0;
Num_Stars = 0;

for i = 1:X_Max;
    for j = 1:Y_Max;
        if FOV_Gauss(i,j) > 0;
            LogStar(i,j,FOV_Gauss);
            if size(Box) > 4;
                Num_Stars = Num_Stars + 1;
                Star_List = [Star_List;Box'];
            end;
        end;
    end;
end;

Num_Stars

pause;

##### Calculate Centroid for continuous star #####
Num_Stars = 2;

for b = 1:Num_Stars;

[d1,i1] = max(Star);
[d2,i2] = max(d1);

x0 = i1(i2);
y0 = i2;

X0 = [X0;x0];
Y0 = [Y0;y0];

Ix1 = sum(sum(Star(x0-n_algo:x0+n_algo,y0-n_algo:y0+n_algo)));
Iy1 = sum(sum(Star(x0-n_algo:x0+n_algo,y0-n_algo:y0+n_algo)));

XI_Sum1 = sum(Star(x0-n_algo:x0+n_algo,y0-n_algo:y0+n_algo))*(-
n_algo:n_algo)';
YI_Sum1 = sum(Star(x0-n_algo:x0+n_algo,y0-n_algo:y0+n_algo))*(-1:-
n_algo)';

Mx1 = XI_Sum1/Ix1;
My1 = YI_Sum1/Iy1;

MX1 = [MX1;Mx1];
MY1 = [MY1;My1];

Ix2 = sum(sum(Star(x0-(n_algo-1):x0+(n_algo+1),y0-n_algo:y0+n_algo)));
Iy2 = sum(sum(Star(x0-n_algo:x0+n_algo,y0-(n_algo+1):y0+(n_algo-1))));

XI_Sum2 = sum(Star(x0-(n_algo-1):x0+(n_algo+1),y0-n_algo:y0+n_algo))*(-
n_algo:n_algo)';
YI_Sum2 = sum(Star(x0-n_algo:x0+n_algo,y0-(n_algo+1):y0+(n_algo-
1)))*(-1:-n_algo)';

Mx2 = XI_Sum2/Ix2;
My2 = YI_Sum2/Iy2;

MX2 = [MX2;Mx2];
MY2 = [MY2;My2];

Xs0 = Stars_p(b,1)-0.5;
Ys0 = Stars_p(b,2)-0.5;

XS0 = [XS0;Xs0];
YS0 = [YS0;Ys0];

Xs1 = Xs0;
Ys1 = Ys0;

```

```

Xs1 = Xs0 + Mx1;
Ys1 = Ys0 + My1;

Xs = Xs1;
Ys = Ys1;

Xsd = (10/X_Max)*Xs;
Ysd = (10/Y_Max)*Ys;

error_x = [error_x;abs(Stars_r(b,1)-Xsd)];
error_y = [error_y;abs(Stars_r(b,2)-Ysd)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Digital Section %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[d1_d,i1_d] = max(Star_d);
[d2_d,i2_d] = max(d1_d);

x0_d = i1_d(i2_d);
y0_d = i2_d;

X0_d = [X0_d;x0_d];
Y0_d = [Y0_d;y0_d];

Ix1_d = sum(sum(Star_d(x0_d-n_algo:x0_d+n_algo,y0_d-
n_algo:y0_d+n_algo)));
Iy1_d = sum(sum(Star_d(x0_d-n_algo:x0_d+n_algo,y0_d-
n_algo:y0_d+n_algo)));

XI_Sum1_d = sum(Star_d(x0_d-n_algo:x0_d+n_algo,y0_d-
n_algo:y0_d+n_algo))*(-n_algo:n_algo)';
YI_Sum1_d = sum(Star_d(x0_d-n_algo:x0_d+n_algo,y0_d-
n_algo:y0_d+n_algo))*(n_algo:-1:-n_algo)';

Mx1_d = XI_Sum1_d/Ix1_d;
My1_d = YI_Sum1_d/Iy1_d;

MX1_d = [MX1_d;Mx1_d];
MY1_d = [MY1_d;My1_d];

Ix2_d = sum(sum(Star_d(x0_d-(n_algo-1):x0_d+(n_algo+1),y0_d-
n_algo:y0_d+n_algo)));
Iy2_d = sum(sum(Star_d(x0_d-n_algo:x0_d+n_algo,y0_d-
(n_algo+1):y0_d+(n_algo-1))));

XI_Sum2_d = sum(Star_d(x0_d-(n_algo-1):x0_d+(n_algo+1),y0_d-
n_algo:y0_d+n_algo))*(-n_algo:n_algo)';
YI_Sum2_d = sum(Star_d(x0_d-n_algo:x0_d+n_algo,y0_d-
(n_algo+1):y0_d+(n_algo-1)))*(n_algo:-1:-n_algo)';

Mx2_d = XI_Sum2_d/Ix2_d;
My2_d = YI_Sum2_d/Iy2_d;

MX2_d = [MX2_d;Mx2_d];
MY2_d = [MY2_d;My2_d];

Xs0_d = Stars_p(b,1)-0.5;
Ys0_d = Stars_p(b,2)-0.5;

XS0_d = [XS0_d;Xs0_d];
YS0_d = [YS0_d;Ys0_d];

Xs1_d = Xs0_d;
Ys1_d = Ys0_d;

Xs1_d = Xs0_d + Mx1_d;
Ys1_d = Ys0_d + My1_d;

Xs_d = Xs1_d;
Ys_d = Ys1_d;

Xsd_d = (10/X_Max)*Xs_d;
Ysd_d = (10/Y_Max)*Ys_d;

```

```
error_x_d = [error_x_d;abs(Stars_r(b,1)-Xsd_d)];
error_y_d = [error_y_d;abs(Stars_r(b,2)-Ysd_d)];

end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculate errors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
error_r = sqrt(error_x.^2+error_y.^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot errors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clg;
hold off;
axis('normal');
plot([error_x error_y error_r], 'w');
title('Centroid Errors');
grid;

pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculate errors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
error_r_d = sqrt(error_x_d.^2+error_y_d.^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot errors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clg;
hold off;
axis('normal');
plot([error_x_d error_y_d error_r_d], 'w');
title('Digital Centroid Errors');
grid;
```

## Appendix B Program Listings

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%  
% Filename      : logstar.m  
%  
% Description   : Region growing function for grow.m  
%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
function [Box,FOV_Gauss] = LogStar(i,j,FOV_Gauss)  
disp(FOV_Gauss(i,j))  
if FOV_Gauss(i,j) > 0;  
    Box = [Box;[i j]];  
    FOV_Gauss(i,j) = 0;  
    LogStar(i,j-1,FOV_Gauss);  
    LogStar(i,j+1,FOV_Gauss);  
    LogStar(i-1,j,FOV_Gauss);  
    LogStar(i+1,j,FOV_Gauss);  
end;
```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : centre.m
%
% Description  : This program simulates a CCD FOV image and calculates
%               the centroids of all the stars in the FOV.
%
%               The positional errors for x,y and r are calculated and
%               plotted.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear Environment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

echo off;
clear;
clg;
clc;
hold off;
rand('seed',5);
rand('uniform');
axis('normal');
format short;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Constants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

X_Max = 200;
Y_Max = 200;

P_X = 13.75e-6;
P_Y = 16e-6;

Max_Stars = 20;
IO         = 10;
sigma1     = 1.325;
a          = 0.75;
D          = 0.03;
lambda     = 555e-9;
width      = 5;
s_width    = 5;
n_algo     = 2;
dist       = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set sigma2 for distortion %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sigma2 = dist*sigma1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Main loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for b = 0:0.01:0.5;

    FOV_Gauss = zeros(Y_Max,X_Max);
    clear(Star);

    xp = 10;
    yp = 10;

    Stars_p(1,1) = 10;
    Stars_p(1,2) = 10;

    Size = 2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set error for smear %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% smear
% b = b + s*cos(90/180*pi)*(t-(T/2));

```

## Appendix B Program Listings

```

##### Gaussian/Airy Stellar image model #####
    for xx = -Size:Size;
        for yy = -Size:Size;
            x = xx - b;
            y = yy;
            I_Gauss = I0*exp(-((x^2+y^2)/(2*sigma1*sigma2)));
##### Airy #####
%
%           I_Airy = I0*((2*(bessel(1,sqrt(x^2+y^2))) )./(sqrt(x^2+y^2))).^2;
%
#####
            FOV_Gauss(Y_Max-(yp+yy),xp+xx) = I_Gauss;
        end;
    end;

    Star0 = FOV_Gauss(Y_Max-(Stars_p(1,2)-s_width):-1:Y_Max-
(Stars_p(1,2)+s_width),Stars_p(1,1)-s_width:Stars_p(1,1)+s_width);

```

## Appendix B Program Listings

```

##### Add noise #####
n = 0.1*(rand(ones(s_width*2+1))-0.05);
Star = n+Star0;
S = size(Star);

##### Calculate Centroid for continuous star #####
[d1,i1] = max(Star);
[d2,i2] = max(d1);

x0 = i1(i2);
y0 = i2;

X0 = [X0;x0];
Y0 = [Y0;y0];

##### Display 3D graphs of stars #####

clg;
subplot(221);
plot(Star(x0,y0-3:y0+3),'g');
grid;
mesh(Star0);
mesh(Star);
contour(Star0,10);
grid;
contour(Star,10);
grid;

bar(Star(x0,:));
grid;
pause;

br = [br;Star(x0,:)];

##### Calculate centroid #####

Ix1 = sum(sum(Star(x0-n_algo:x0+n_algo,y0-n_algo:y0+n_algo)));
Iy1 = sum(sum(Star(x0-n_algo:x0+n_algo,y0-n_algo:y0+n_algo)));

XI_Sum1 = 0;
for i = -2:2;
    XI_Sum1 = XI_Sum1 + i*sum(Star(y0-2:y0+2,i+x0));
end;

YI_Sum1 = 0;
for i = -2:2;
    YI_Sum1 = YI_Sum1 + i*sum(Star(i+y0,x0-2:x0+2));
end;

Mx1 = XI_Sum1/Ix1;
My1 = YI_Sum1/Iy1;

MX1 = [MX1;Mx1];
MY1 = [MY1;My1];

error_x = b-Mx1;
error_y = b-My1;

T_errory = [T_errory;error_y];
T_errorx = [T_errorx;error_x];

end;

clg;

Max_Error = max(abs(T_errorx-N_errorx));

```

```
##### Plot error results #####  
plot(0:0.01:0.5,N_errorx,'w');  
hold on;  
plot(0.1:0.05:0.5,T_errory,'g');  
grid;
```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : guide.m
%
% Description  : This program creates the guide star database for the
%                star pattern recognition algorithm.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear Environment %%%%%%%%%%

clear;
clg;
clc;
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Load star catalogue data %%%%%%%%%%

load yalecat.mat;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Get parameters %%%%%%%%%%

Mag_Max      = input('Magnitude Limit : ');
FOV_Width    = input('FOV Width : ');
Line_Length_Max = 2*(FOV_Width * FOV_Width);
Line_Length_Min = input('CCD Resolution Limit : ');
Line_Length_Min = Line_Length_Min * Line_Length_Min;
Orbit_Angle  = input('Angle between orbit and ecliptic : ');
Sense_Ack    = input('Other sensor accuracy : ');

Dec_Max = Orbit_Angle+(FOV_Width/2)+Sense_Ack;
Dec_Min = Orbit_Angle-(FOV_Width/2)-Sense_Ack;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Filter on magnitude %%%%%%%%%%

Star_Length_0 = length(Decmap);
Guide_Stars_1 = [];
i = 1;
j = 1;

disp('Busy with magnitude filtering...');
disp('');

while i < Star_Length_0;
    if magmap(i) <= Mag_Max;
        if Decmap(i) < Dec_Max;
            if Decmap(i) > Dec_Min;
                Guide_Stars_1(j,:) = [RAMap(i) Decmap(i)];
                Guide_Mags_1(j) = magmap(i);
                j = j + 1;
            end;
        end;
    end;
    i = i + 1;
end;

Star_Length_1 = length(Guide_Stars_1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear redundant variables %%%%%%%%%%

clear RAMap;
clear Decmap;
clear magmap;

disp('Magnitude filtering complete. ');
disp(Star_Length_1);
disp('Press any key. ');
disp('');

pause;

```

```

i = 1;
j = 1;
k = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculate angular separations od stars %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Busy with line length calculation...');

while i < (Star_Length_1-1);
    j = i + 1;
    if i < (Star_Length_1-30);
        l = j + 30;
    else
        l = Star_Length_1;
    end;
    while j < l;
        R2 = ((Guide_Stars_1(i,1)-Guide_Stars_1(j,1))*(Guide_Stars_1(i,1)-
Guide_Stars_1(j,1)))+((Guide_Stars_1(i,2)-
Guide_Stars_1(j,2))*(Guide_Stars_1(i,2)-Guide_Stars_1(j,2)));
        if R2 < Line_Length_Max,
            if R2 > 0;
                Guide_Lines_1(k) = R2;
                Guide_Line_Stars_1(k,:) = [i j];
                k = k + 1;
            end;
        end;
        j = j + 1;
    end;
    j = 1;
    i = i + 1;
end;

[Guide_Lines_1,I] = sort(Guide_Lines_1);
Guide_Line_Stars_1 = Guide_Line_Stars_1(I,:);

Guide_Lines_1 = Guide_Lines_1';

Guide_Lines_1 = sqrt(Guide_Lines_1);

save L65 Guide_Lines_1;
save N65 Guide_Line_Stars_1;

Line_Length_1 = length(Guide_Lines_1);

disp('Line length calculation complete. ');
disp(Line_Length_1);
disp('Press any key. ');
disp(' ');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Check for close pairs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Checking for close pairs...');

for i = 1:Line_Length_1;
    if Guide_Lines_1(i) < Line_Length_Min;
        if Guide_Mags_1(Guide_Line_Stars_1(i,1)) >
Guide_Mags_1(Guide_Line_Stars_1(i,2));
            Discard_1 = [Discard_1;Guide_Line_Stars_1(i,2)];
        else
            Discard_1 = [Discard_1;Guide_Line_Stars_1(i,1)];
        end;
    end;
end;

Discard_Length_1 = length(Discard_1);

disp('Close pair check complete. ');
disp(Discard_Length_1);
disp(' ');

```

## Appendix B Program Listings

```

##### Fix up Discard array #####
Discard_1 = sort(Discard_1);

k = 1;
i = 1;

Discard_2(k) = Discard_1(i);

while i < Discard_Length_1;
    while Discard_1(i) == Discard_2(k) , i < Discard_Length_1;
        i = i + 1;
    end;
    if i <= Discard_Length_1;
        k = k + 1;
        Discard_2(k) = Discard_1(i);
        i = i + 1;
    end;
end;

Discard_2 = Discard_2';

Discard_Length_2 = length(Discard_2);

##### Discard stars from Guide_Stars array #####

if Discard_Length_2 > 0;

disp('Discarding dim close partners...');

Out = 0;
Ok = 0;
j = 1;

for i = 1:Star_Length_1;
    while Out == 0;
        if i == Discard_2(j);
            Out = 1;
        end;
        j = j + 1;
        if j > Discard_Length_2;
            if Out == 1;
                Ok = 0;
            else
                Out = 1;
                Ok = 1;
            end;
        end;
    end;
    if Ok == 1;
        Guide_Stars_2 = [Guide_Stars_2;Guide_Stars_1(i,:)];
    end;
    j = 1;
    Out = 0;
    Ok = 0;
end;

Star_Length_2 = length(Guide_Stars_2);

disp('Stars discarded from Guide_Stars array');
disp(Star_Length_2);
disp('');

##### Discard by sorting #####

Out = 0;
j = 1;

disp('Discarding stars from Guide_Line_Stars matrix...');
disp('');

```

```

for i = 1:Line_Length_1;
    for j = 1:Discard_Length_2;
        if Discard_2(j) == Guide_Line_Stars_1(i,1);
            Guide_Line_Stars_1(i,1) = 0;
            disp(i);
        end;
        if Discard_2(j) == Guide_Line_Stars_1(i,2);
            Guide_Line_Stars_1(i,1) = 0;
            disp(i);
        end;
    end;
end;

j = 1;

for i = 1:Line_Length_1;
    if Guide_Line_Stars_1(i,1) > 0;
        Guide_Line_Stars_2(j,1) = Guide_Line_Stars_1(i,1);
        Guide_Line_Stars_2(j,2) = Guide_Line_Stars_1(i,2);
        Guide_Lines_2(j) = Guide_Lines_1(i);
        j = j + 1;
    end;
end;

Line_Length_2 = length(Guide_Lines_2);

disp('Stars discarded from Guide_Lines array and Guide_Line_Stars
matrix. ');
disp(Line_Length_2);
disp('');

end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Display results %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(Star_Length_1);
disp(Line_Length_1);
disp('');
disp(Star_Length_2);
disp(Line_Length_2);

```

## Appendix B Program Listings

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Filename      : pitch.m
%
% Description : This program calculates the accuracy that can be
%               achieved for a pitch angle when the x and y accuracies
%               are specified.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Clear the memory & screen %%%%%%%%%%%%%%%

clear;
clg;
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set Constants & Initialise variables %%%%%%%%%

i = 1;
xmax = 10/180*pi;
ymax = 10/180*pi;
dxmax = 0.0022/180*pi;
dymax = 0.0022/180*pi;

dx = dxmax;
dy = dymax;

x = 30*dxmax;
y = 30*dymax;

x = 5/180*pi;
y = 10/180*pi;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculate pitch angle from x,y,dx and dy %%%%%%%%%

for dx = 0.00001:0.00001:dxmax;
    for dy = 0.00001:0.00001:dymax;
        for x = 30*dxmax:0.01:xmax;
            for y = 30*dymax:0.01:ymax;
                Error(i) = atan((y+2*dy)/(x-2*dx)) - atan((y-2*dy)/(x+2*dx));
                i = i + 1;
            end;
        end;
    end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot results %%%%%%%%%

plot(Error/pi*180);
grid;

```

```

Unit AGuide;

Interface

Uses

Graph, Crt, AShare;

Var

niks          : integer;

procedure Get_Cat_Data;
procedure Calc_Cat_Lines;
procedure Sort_Cat_Lines;
procedure Sort_2_Cat_Lines;
procedure Show_Cat_Lines(total:integer);

Implementation

var

p          : integer;

{*****}
{ Get star data from database and display }
{*****}

procedure Get_Cat_Data;

var

tel,x1          : integer;
yl              : integer;
Index           : integer;
VecLimit       : integer;
Count          : integer;
RA,Dec, Mag    : real;
s              : string;

begin

  Assign(starfile, 'c:\tp\data\starmap.dat');

  tel:=0;
  x1:=0;

  WCat :=WFOV*3;

  MagLimit1:=3;

  Index := 3*(round((9094/360)*(RACatC-WFOV-20)));

  VecLimit := 200;
  Count := 0;

  reset(StarFile);
  seek(StarFile, Index);

  read(StarFile, RA, Dec, Mag);

  while (RA<(RACatC+(WCat/2))) do
  begin
    if (Mag<MagCatLimit) and (Dec<(DecCatC+(WFOV/2)+Senacc)) and
    (Dec>(DecCatC-((wfov/2)+senacc)))
    and (RA>(RACatC-(WCat/2))) and (count<200) then

```

## Appendix B Program Listings

```

begin
  count := count+1;
  RACat[count] := RA;
  DecCat[count] := Dec;
  MagCat[count] := Mag;

  writeln(count, '. RA = ', RA:3:3, ' Dec = ', Dec:3:3, ' mag = ', mag:3:3);

  XCat[count] := RA-RACatC;
  YCat[count] := Dec-DecCatC;

  end;
  read(StarFile, RA, Dec, mag);
end;

NCat := count;

CatStarLength:=NCat;
CatStarLengthI:=CatStarLength;

writeln;
writeln('CatStarLength= ', CatStarLength);
readln;

write(week, CatStarLengthI);

Close(StarFile);

end;

{*****}
{ Swap procedure for sorting procedure }
{*****}

procedure swap(var a1,b1:real);

var

t1: real;

begin

  t1:=a1;
  a1:=b1;
  b1:=t1;

end;

{*****}
{ Swap procedure for sorting procedure }
{*****}

procedure Swapi(var a,b:integer);

var

t: integer;

begin

  t:=a;
  a:=b;
  b:=t;

end;

{*****}

```

```

{ Sort the lines in ascending order of length
{*****}

procedure Sort_Cat_Lines;

var

j,k      : integer;
sorted  : boolean;

begin

  k := newlength-1;

  { writeln('k = ',newlength);
  readln;      }

  sorted := false;
  while (k>0) and (not sorted) do
  begin
    sorted := true;
    for j:= 1 to k do
    begin
      if LCat[j]>LCat[j+1] then
      begin
        Swap(LCat[j],LCat[j+1]);
        Swapi(SNCat[j,1],SNCat[j+1,1]);
        Swapi(SNCat[j,2],SNCat[j+1,2]);
        {readln;}
        sorted := false
      end;
    end;
    k := k-1
  end;

end;

procedure Show_Cat_Lines(total : integer);

var

i          : integer;
ttotal    : integer;

begin

{ writeln('Line   ', 'Star 1  ', 'X  ', ' Y  ', ' mv ', ' Star 2  ', 'X  ', '
Y', ' mv');
}

  ttotal := 0;
  for i := 1 to total do
  begin
    if ((MagCat[SNCat[i,1]]<>10) and (MagCat[SNCat[i,2]]<>10)) then
    begin
      ttotal:=ttotal+1;
      LCat[ttotal]:=LCat[i];
      SNCat[ttotal,1]:=SNCat[i,1];
      SNCat[ttotal,2]:=SNCat[i,2];

      { writeln(tttotal, ' ',LCat[i]:3:3, ' ',SNCat[i,1], '
',RACat[SNCat[i,1]]:3:3, ' ',DecCat[SNCat[i,1]]:3:3, ' ',
      MagCat[SNCat[i,1]]:3:3, ' ',SNCat[i,2], ' ',RACat[SNCat[i,2]]:3:3, '
',DecCat[SNCat[i,2]]:3:3, ' ',MagCat[SNCat[i,2]]:3:3);
      }

    end;
  end;
end;

```

```

{ writeln;
  writeln('There are ',tttotal,' Map lines');
}

newlength:=tttotal;

end;

procedure Show_Cat_Stars;

var

i : integer;

begin

  for i := 1 to length do
  begin
    writeln(i,'. ',RACat[i]:3:3,' ',DecCat[i]:3:3,' ',MagCat[i]:3:3);
  end;

end;

procedure Calc_Cat_Lines;

var

i,j          : integer;
LCatL        : real;
tel          : integer;
s            : string;

begin

  tel:=0;
  length:=NCat;

  (
  writeln('length = ',length);
  readln;
  )

  for i := 1 to length do
  begin
    for j := 1 to length do
    begin
      LCatL:=sqrt(((RACat[i]-RACat[j])*(RACat[i]-RACat[j]))+((DecCat[i]-
DecCat[j])*(DecCat[i]-DecCat[j]))));
      if ((LCatL < Closer) and (i<>j)) then
      begin
        if MagCat[i]>MagCat[j] then
        begin
          MagCat[i] := 10;
        end
        else
        begin
          MagCat[j]:=10;
        end;
      end;
      if ((LCatL < RCatLimit) and (LCatL >= Closer)) and (tel<1000) then
      begin
        tel:=tel+1;
        LCat[tel]:=LCatL;
        SNCat[tel,1]:=i;
        SNCat[tel,2]:=j;
      end;
    end;
  end;
end;

```

```

{
writeln;
}

newlength := tel;

{
str(tel,s);
OutTextXY(350,100,s);
}

{
writeln('There are ',tel,' Map lines');
readln;
}

end;

procedure Sort_2_Cat_Lines;

var

i          : integer;
c          : integer;

begin

c:=1;

for i:=1 to round(newlength/2) do
begin
LCat[i]:=LCat[c];
SNCat[i,1]:=SNCat[c,1];
SNCat[i,2]:=SNCat[c,2];

{writeln(LCat[i]);}
{readln;}

c:=c+2;
end;

for i:=1 to round(newlength/2) do
begin
LCat[i]:=LCat[i];
SNCat[i,1]:=SNCat[i,1];
SNCat[i,2]:=SNCat[i,2];

writeln(i,'. LCat = ',LCat[i]:3:3,' SNCat['',i,',',1] = ',SNCat[i,1],',
SNCat['',i,',',2] = ',SNCat[i,2]);
end;

newnewlength:=i;

CatLineLength:=i;

{
WriteLn('OK');
readln;
}

end;

:

procedure rotation;

var
lamdaT,phiT,s,c,ls,lc : real;
i,ii                  : integer;
RAs,Decs,RAp,Decp    : array[1..10] of real;
RAc,Decc              : real;

```

```

begin

for i := 0 to ii do
  begin
    if (RAs[i]<=180) then RAs[i]:=-RAs[i];
    if (RAs[i]>180) then RAs[i]:=360-RAs[i];
    RAs[i]:=(RAs[i]/180)*pi;
    Decs[i]:=(Decs[i]/180)*pi;
  end;

  if (RAc<=180) then RAc:=- (RAc/180)*pi;
  if (RAc>180) then RAc:=(360-RAc)/180*pi;

  Decc:=(Decc/180)*pi;

  lamdaT:=RAc;
  phiT:=pi/2+Decc;

  for i:=0 to ii do
    begin
      s:=((sin(Decs[i])*sin(phiT)) +
          (cos(Decs[i])*cos(phiT)*cos(RAs[i]-lamdaT)));
      c:=sqrt(1-s*s);
      Decp[i]:=arctan(s/c);

      ls:=sin(sin(RAs[i]-lamdaT)*cos(Decs[i]))/cos(Decp[i]);
      lc:=cos((cos(Decs[i])*sin(phiT)*cos(RAs[i]-lamdaT)) -
          (sin(Decs[i])*cos(phiT)))/cos(Decp[i]);
      RAp[i]:=arctan(ls/lc);

    end;

    RAc:=(RAc/pi*180);
    Decc:=(Decc/pi)*180;

  end;

end.

```

## Appendix B Program Listings

```

unit match;

interface

uses

Dos, Crt, Graph, AShare;

var

OMAT                : array[1..250] of integer;
GMAT                : array[1..250] of integer;
NASS, NASS1        : array[1..250] of integer;
{LdL               : array[1..250] of real;}
MAT, MAT1          : array[1..250, 1..20] of integer;
NassLength         : integer;
NassMax            : integer;
SizeFactor         : integer;
Threshold          : integer;
best               : boolean;
firsttime1, firsttime2 : boolean;
IDIndex            : integer;
s1, s2, s3, s4, s5, s6, s7 : string;
s                  : string;

procedure BezMatch;
procedure Attitude;

Implementation

var

iii                : integer;

{ Forward procedures }

procedure Choice2; forward;
procedure Choice3; forward;
procedure Choice4(Nm : integer); forward;
procedure Choice5; forward;
procedure Choice6; forward;
procedure Choice7; forward;
procedure Choice8; forward;
procedure Choice9; forward;
procedure Choice10; forward;
procedure Choice11; forward;
procedure Choice12; forward;
procedure selectbest; forward;
procedure select; forward;
procedure IDFail; forward;
procedure generate; forward;
procedure lower; forward;
{procedure results; forward;}

{*****
PROCEDURE NAME : BezMatch
DATE: 7/7/93
GENERAL DESCRIPTION: Start Van Bezooijen algo.
ARGUMENTS:
RETURN VALUE:
PSEUDO CODE:
NOTES:

```

```

*****}

procedure BezMatch;

var

i,j           : integer;
dL           : real;

begin

if NFOV>=3 then
begin

  ( Clear variables )

  IDIndex:=0;

  for i:=1 to 250 do
  begin
    for j:= 1 to 20 do
    begin
      MAT[i,j]:=0;
      MAT1[i,j]:=0;
    end;
  end;

  NassMax:=0;
  SizeFactor:=0;
  NassLength:=0;

  for i:= 1 to 250 do
  begin
    OMAT[i]:=0;
    GMAT[i]:=0;
    NASS[i]:=0;
    NASS1[i]:=0;
  end;

  {
  writeln('CatStarLength = ',CatStarLength);
  writeln('FOVStarLength = ',FOVStarLength);
  writeln;

  writeln('CatLineLength = ',CatLineLength);
  writeln('FOVLineLength = ',FOVLineLength);
  writeln;

  writeln;
  writeln('Starting matches...');
  writeln('tdis= ',tdis:3:3);
  }

  { ClrScr;}

  generate;

end
else
begin

IDFail;

end;

```

```
end;
```

```
(*****
  PROCEDURE NAME : Generate
  DATE: 21/7/93
  GENERAL DESCRIPTION: Generate match groups.
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
  *****)
```

```
procedure generate;
```

```
{ 1 Generate match groups }
```

```
var
```

```
i,j          : integer;
dL           : real;
index       : integer;
count      : integer;
match1,match2 : integer;
flag1,flag2 : boolean;
```

```
begin
```

```
index:=0;
```

```
{
writeln('Generate');
}
```

```
count:=0;
Flag1:=False;
Flag2:=False;
```

```
for i:= 1 to FOVLineLength do
begin
```

```
  for j:= 1 to CatLineLength do
```

```
  begin
```

```
    dL := abs(Lfov[i]-LCat[j]);
```

```
    if (dL<=TDis) then
```

```
    begin
```

```
      index:=0;
```

```
      if count<250 then
```

```
      begin
```

```
        count:=count+1;
```

```
{    LdL[count]:=dL;}
```

```
        flag1:=False;
```

```
        flag2:=False;
```

```
      end;
```

```
    {
```

```
      writeln(count,'. ','I found one!');
```

```
      writeln('O= ',SNfov[i,1],', ', G= ',SNCat[j,1]);
```

```
      writeln('O= ',SNfov[i,2],', ', G= ',SNCat[j,2]);
```

```
      writeln;
```

```
    }
```

```
  while (flag1=False) and (index<250) do
```

```
  begin
```

```
    index:=index+1;
```

```
    if OMAT[index]=0 then
```

```
    begin
```

```
      OMAT[index]:=SNfov[i,1];
```

```
      GMAT[index]:=SNCat[j,1];
```

```
      NASS[index]:=1;
```

```
      NassLength:=index;
```

## Appendix B Program Listings

```

    {
      writeln('A:NASS=[' , index, ']=' , NASS[index]);
    }

    flag1:=True;
  end;
  if (OMAT[index]=SNfov[i,1]) and (GMAT[index]=SNCat[j,1]) and
(flag1=False)
  and (NASS[index]<19) then
  begin
    NASS[index]:=NASS[index]+1;

    {
      writeln('B:NASS=[' , index, ']=' , NASS[index]);
    }

    flag1:=True;
  end;
end;

match1:=index;

index:=0;

{
  writeln;
}

while flag2=False do
begin
  index:=index+1;
  if OMAT[index]=0 then
  begin
    OMAT[index]:=SNfov[i,2];
    GMAT[index]:=SNCat[j,2];
    NASS[index]:=1;
    NassLength:=index;

    {
      writeln('C:NASS=[' , index, ']=' , NASS[index]);
    }

    flag2:=True;
  end;
  if (OMAT[index] = SNfov[i,2]) and (GMAT[index]=SNCat[j,2]) and (flag2
= False)
  and (NASS[index]<19) then
  begin
    NASS[index]:=NASS[index]+1;

    {
      writeln('D:NASS=[' , index, ']=' , NASS[index]);
    }

    flag2:=True;
  end;
end;

match2:=index;

MAT[match1,NASS[match1]]:=match2;
MAT[match2,NASS[match2]]:=match1;

{
  writeln('End of search');
  writeln;
}

end;
end;
end;

```

```

writeln('NassLength = ',NassLength);

{ 2 Determine upper bound of match group size }
NassMax:=0;
for i:= 1 to NassLength do
begin
  {
  writeln(i);
  }

  if NASS[i]>NassMax then
  NassMax:=NASS[i];
end;

for i:= 1 to NassLength do
begin
  if NASS[i]>NassMax-1 then
  begin
    SizeFactor:=SizeFactor+1;
  end;
end;

writeln;
writeln('NassMax = ',NassMax);
writeln('SizeFactor = ',SizeFactor);

readln;

Choice2;

end;

{*****
PROCEDURE NAME : setthreshold
DATE: 7/7/93
GENERAL DESCRIPTION: Calculate threshold for minimum
                    match group size.
ARGUMENTS:
RETURN VALUE:
PSEUDO CODE:
NOTES:
*****}

procedure setthreshold;

{ 4 Establish threshold for match group size }
begin
  {
  writeln('setthreshold');
  }

  Threshold:=NassMax-2;
  select;

end;

{*****}

```

## Appendix B Program Listings

```

PROCEDURE NAME : Choice2
DATE: 7/7/93
GENERAL DESCRIPTION: Check whether the largest match
                    group is large enough.

ARGUMENTS:
RETURN VALUE:
PSEUDO CODE:
NOTES:

```

```
*****}
```

```

procedure Choice2;

{ 3 Match groups large enough? }

begin

```

```
    writeln('Choice2');
```

```

    s1:='Choice2';
    write(logfile,s1);

```

```

    if NassMax>=SizeFactor then
    begin
        setthreshold;
    end
    else
    begin
        IDFail;
    end;

```

```
end;
```

```

{*****
  PROCEDURE NAME : select
  DATE: 7/7/93
  GENERAL DESCRIPTION: Select largest non-redundant
                      match group(s)

  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
*****}

```

```

procedure select;

{ 5 Select largest non-redundant valid match group(s) }

var

i,j,k,l           : integer;
OK                : boolean;
kernel           : integer;
flag1,flag2      : boolean;
Vprodmap,VprodFOV : real;

```

```
begin
```

```
    writeln('select');
```

```

for i:=1 to NassLength do
begin
  {
  write(i,'. ',OMAT[i],' ',GMAT[i],' * ',NASS[i],' * ');
  }

  NASS1[i]:=NASS[i];
  for j:=1 to 19 do
  begin
    {
    write(MAT[i,j],' ');
    }

    MAT1[i,j]:=MAT[i,j];
  end;

  {
  writeln;
  }

end;

{
writeln;
writeln('These are the calculated match groups. ');
writeln;
writeln;
writeln('Threshold = ',Threshold);
writeln;
}

{ 5.1 Eliminate match groups smaller than Threshold }
for i:=1 to NassLength do
begin
  if NASS1[i]<Threshold then
  begin
    NASS1[i]:=0;

    {
    writeln('i = ',i);
    }

    for j:=1 to NassLength do
    begin
      if NASS1[j]>0 then
      begin
        for k:= 1 to 19 do
        begin
          if MAT1[j,k]=i then
          begin
            MAT1[j,k]:=-1;
          end;
        end;
      end;
    end;
  end;

end;

for i:=1 to NassLength do
begin
  if NASS1[i]>0 then
  begin
    for j:=1 to 19 do
    begin
      if MAT1[i,j]=-1 then
      begin
        NASS1[i]:=NASS1[i]-1;
      end;
    end;
  end;
end;

```

```

    end;
end;

for i:=1 to NassLength do
begin
    {
    write(i, ' ', OMAT[i], ' ', GMAT[i], ' * ', NASS1[i], ' * ');
    }

    for j:=1 to 19 do
    begin
        {
        write(MAT1[i,j], ' ');
        }

    end;

    {
    writeln;
    }

end;

{
writeln;
writeln('After eliminating small match groups. ');
writeln;
}

{ 5.2 Validate remaining match groups }

OK:=False;

for i:=1 to NassLength do
begin
    if NASS1[i]>0 then
    begin
        for j:=1 to 19 do
        begin
            if MAT1[i,j]>0 then
            begin
                for k:=j+1 to NassLength do
                begin
                    l:=0;
                    while (OK=False) and (l<>19) do
                    begin
                        l:=l+1;
                        if MAT1[j,l]=k then
                        begin
                            OK:=True;
                        end;
                        if OK=False then
                        begin
                            NASS1[i]:=0;
                        end;
                    end;
                end;
            end;
        end;
    end;
end;
end;
end;
end;
end;
end;

```

```

{ Show results }
for i:=1 to NassLength do
begin

write(i,'. ',OMAT[i],' ',GMAT[i],' * ',NASS1[i],' * ');

for j:=1 to 19 do
begin

write(MAT1[i,j],' ');

end;

writeln;

end;

writeln;
writeln('Matchgroups validated. ');
writeln;

readln;

{ 5.3 Eliminate redundant match groups }

for i:=1 to NassLength do
begin
if NASS1[i]>0 then
begin
writeln('i one = ',i);
readln;
for j:=i+1 to NassLength do
begin
if NASS1[j]>0 then
begin
writeln('NASS1[j] = ',NASS1[j]);
k:=0;
flag2:=False;
flag1:=False;
kernel:=700;
OK:=False;
while (flag2=False) and (k<>19) do
begin
k:=k+1;
l:=0;
while (flag1=False) and (l<>19) do
begin
l:=l+1;
if ((MAT1[i,k]=MAT1[j,l]) and (MAT1[i,k]>0) and (MAT1[j,l]>0)) then
begin
flag1:=True;
OK:=True;
end;
end; (l)
if flag1=False then
begin
if kernel=700 then
begin
kernel:=MAT1[i,k];
end

```

```

    else
    begin
        flag2:=True;
        OK:=False;
    end;
    end; ( if flag1=False)
    OK:=False;
end; (k)
end; (>)

if ((kernel=700) and (NASS1[j]>0)) then
begin
    OK:=True;
end;

if kernel>0 then
begin
    if kernel=j then
    begin
        OK:=True;
    end;
end;

if OK=True then
begin
    NASS1[j]:=0;
    write(' ',j);
end;

end; (j)
readln;
end; (>)
end; (i)

( Show results )
for i:=1 to NassLength do
begin

write(i, '. ',OMAT[i], ' ',GMAT[i], ' * ',NASS1[i], ' * ');

for j:=1 to 19 do
begin

write(MAT1[i,j], ' ');

end;

writeln;

end;

writeln;
writeln('Redundant matchgroups eliminated. ');
writeln;

readln;

( 5.4 Mirror test )

```



```

begin

  writeln('ID Failure!');

  s:='IDFail';

  write(logfile,s);

  ID:=False;
  WriteLn('ATTITUDE FAIL');
  attitude;

end;

(*****
  PROCEDURE NAME : IDOK
  DATE: 7/7/93
  GENERAL DESCRIPTION: ID success.
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
  *****)

procedure IDOK;

begin

  writeln('IDOK');

  s:='IDOK';

  write(logfile,s);

  ID:=True;

  WriteLn('ATTITUDE OK');

  attitude;

  { results;}

end;

(*****
  PROCEDURE NAME : Choice3
  DATE: 7/7/93
  GENERAL DESCRIPTION: Are there more than 0 match
                      groups of adequate size?
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
  *****)

procedure Choice3;

var

Nm,i      : integer;

begin

```

```

writeln('Choice3');

s:='Choice3';

write(logfile,s);

Nm:=0;

for i:=1 to NassLength do
begin
  if NASS1[i]>0 then
  begin
    Nm:=Nm+1;
    IDIndex:=i;
  end;
end;

writeln('Nm = ',Nm);

if Nm>0 then
begin
  Choice4(Nm);
end
else
begin
  Choice9;
end;

end;

{*****
  PROCEDURE NAME : Choice4
  DATE: 7/7/93
  GENERAL DESCRIPTION: Is there more than one match group?
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
  *****}

procedure Choice4(Nm : integer);

begin

writeln('Choice4');

s:='Choice4';

write(logfile,s);

if Nm>1 then
begin
  Choice5;
end
else
begin
  OutTextXY(320,40,'Choice4 OK');
  IDOK;
end;

end;

```

```
{*****  
  PROCEDURE NAME : Choice5  
  DATE: 7/7/93  
  GENERAL DESCRIPTION: Best groupselection?  
  ARGUMENTS:  
  RETURN VALUE:  
  PSEUDO CODE:  
  NOTES:  
*****}
```

```
procedure Choice5;  
begin  
  
  writeln('Choice5');  
  
  s:='Choice5';  
  write(logfile,s);  
  
  if BestSelect=True then  
  begin  
    Choice6;  
  end  
  else  
  begin  
    Choice8;  
  end;  
end;
```

```
{*****  
  PROCEDURE NAME : Choice6  
  DATE: 7/7/93  
  GENERAL DESCRIPTION: Select best match group the first  
                      time?  
  ARGUMENTS:  
  RETURN VALUE:  
  PSEUDO CODE:  
  NOTES:  
*****}
```

```
procedure Choice6;  
begin  
  
  writeln('Choice6');  
  
  s:='Choice6';  
  write(logfile,s);  
  
  if BestFirst=True then  
  begin  
    Selectbest;  
  end  
  else  
  begin  
    Choice7;  
  end;  
end;
```

```

end;

{*****
  PROCEDURE NAME : Choice7
  DATE: 7/7/93
  GENERAL DESCRIPTION: First time in this procedure?
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
*****}

procedure Choice7;

begin

  writeln('Choice7');

  s:='Choice7';

  write(logfile,s);

  if firsttime1=False then
  begin
    Firsttime1:=True;
    Choice8;
  end
  else
  begin
    Selectbest;
  end;

end;

{*****
  PROCEDURE NAME : Choice8
  DATE: 7/7/93
  GENERAL DESCRIPTION: Are there 2 star matches in common?
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
*****}

procedure Choice8;

var

common                : boolean;
i,j,k,l               : integer;
flag1,flag2,flag3     : boolean;
count                 : integer;

begin

  writeln('Choice8');

  s:='Choice8';

```

```

write(logfile,s);

common:=True;

while (flag1=False) and (i<>NassLength) do
begin
i:=i+1;
if NASS1[i]>0 then
begin
flag1:=True;
end;
end;

j:=i;
flag1:=False;

while flag1=False do
begin
j:=j+1;
if NASS1[j]>0 then
begin
k:=0;
while (k<19) and (flag2=False) do
begin
k:=k+1;
l:=0;
count:=0;
while (l<19) and (flag3=False) do
begin
l:=l+1;
if MAT1[i,k]=MAT1[j,l] then
begin
flag3:=True;
count:=count+1;
end;
end;
if count<2 then
begin
flag2:=True;
flag1:=True;
common:=False;
end;
end;
end;
end;

if common=True then
begin
IDOK;
end
else
begin
Choice9;
end;

end;

(*****
PROCEDURE NAME : Choice9
DATE: 7/7/93
GENERAL DESCRIPTION: Are iterations permitted?
ARGUMENTS:
RETURN VALUE:
PSEUDO CODE:
NOTES:
*****)

procedure Choice9;

```

```

begin

  writeln('Choice9');

  s:='Choice9';

  write(logfile,s);

  if iterations=True then
  begin
    Choice10;
  end
  else
  begin
    IDFail;
  end;
end;

end;

{*****
  PROCEDURE NAME : Choice10
  DATE: 7/7/93
  GENERAL DESCRIPTION: May the threshold be lowered?
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
  *****)

procedure Choice10;

var

lowerOK      : boolean;

begin

  writeln('Choice10');

  s:='Choice10';

  write(logfile,s);

  if Threshold>1 then
  begin
    lowerOK:=True;
  end;

  if lowerOK=True then
  begin
    lower;
  end
  else
  begin
    Choice11;
  end;
end;

end;

{*****
  PROCEDURE NAME : Choice11
  DATE: 7/7/93
  *****)

```

```

GENERAL DESCRIPTION: Ok to select best group?
ARGUMENTS:
RETURN VALUE:
PSEUDO CODE:
NOTES:

```

```

*****}

```

```

procedure Choicell;

```

```

begin

```

```

  writeln('Choicell');

```

```

  s:='Choicell';

```

```

  write(logfile,s);

```

```

  if BestSelect=True then

```

```

  begin

```

```

    Choicel2;

```

```

  end

```

```

  else

```

```

  begin

```

```

    IDFail;

```

```

  end;

```

```

end;

```

```

(*****}

```

```

  PROCEDURE NAME : Choicel2

```

```

  DATE: 7/7/93

```

```

  GENERAL DESCRIPTION: First time this procedure?

```

```

  ARGUMENTS:

```

```

  RETURN VALUE:

```

```

  PSEUDO CODE:

```

```

  NOTES:

```

```

*****}

```

```

procedure Choicel2;

```

```

begin

```

```

  writeln('Choicel2');

```

```

  s:='Choicel2';

```

```

  write(logfile,s);

```

```

  if firsttime2=False then

```

```

  begin

```

```

    Firsttime2:=True;

```

```

    setthreshold;

```

```

  end

```

```

  else

```

```

  begin

```

```

    IDFail;

```

```

  end;

```

```

end;

```

```

(*****}

```

```

  PROCEDURE NAME : lower

```

```

  DATE: 7/7/93

```

GENERAL DESCRIPTION: Lower the threshold by one.  
 ARGUMENTS:  
 RETURN VALUE:  
 PSEUDO CODE:  
 NOTES:

\*\*\*\*\*)

procedure lower;

begin

writeln('lower');

s:='Lower';

write(logfile,s);

Threshold:=Threshold-1;

select;

end;

(\*\*\*\*\*

PROCEDURE NAME : selectbest

DATE: 7/7/93

GENERAL DESCRIPTION: Select the best match group.

ARGUMENTS:

RETURN VALUE:

PSEUDO CODE:

NOTES:

\*\*\*\*\*)

procedure selectbest;

var

i,j,k,l : integer;

count : integer;

Sum,Sum0 : real;

index : integer;

begin

writeln('selectbest');

s:='SelectBest';

write(logfile,s);

Sum:=0;

Sum0:=0;

for i:=1 to NassLength do

begin

if NASS1[i]>0 then

begin

for j:= 1 to 19 do

begin

if MAT1[i,j]>0 then

begin

Sum0:=Sum0+1;

end;

```

end;

if Sum0>Sum then
begin
  Sum:=Sum0;
  index:=i;
end;

Sum0:=0;

end;
end;

IDIndex:=index;

IDOK;

end;

(*****
  PROCEDURE NAME : results
  DATE: 7/7/93
  GENERAL DESCRIPTION: Show the match results.
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
*****)

procedure results;

( Show results of matching )

var

tel,i,j,k           : integer;
FinalFlag          : boolean;
flag               : boolean;

begin

  writeln('results');

  (

  writeln;
  writeln('FOV          CAT          NASS');
  writeln;

  )

  flag:=False;

  if IDIndex=0 then
  begin

  tel := 0;
  FinalFlag:=False;
  k:=0;
  i:=0;

  while (i<NassLength) and (FinalFlag=False) do
  begin
    i:=i+1;
    if NASS1[i]>0 then
    begin

```

```

j:=0;
while (tel=0) and (j<19) do
begin
  j:=j+1;
  if MAT1[i,j]>0 then
  begin
    k:=MAT1[i,j];
    tel:=1;
    CatStar1:=GMAT[i];
    CatStar2:=GMAT[k];
    FOVStar1:=OMAT[i];
    FOVStar2:=OMAT[k];
    writeln('i = ',i);
    writeln('k = ',k);
    readln;
    FinalFlag:=True;
  end;
end;
end;
end;

end

else
begin
  CatStar1:=GMAT[IDIndex];
  FOVStar1:=OMAT[IDIndex];

  j:=0;

  while ((j<19) and (flag=False)) do
  begin
    j:=j+1;
    if MAT1[IDIndex,j]>0 then
    begin
      flag:=True;
    end;
  end;

  if flag=True then
  begin
    CatStar2:=GMAT[MAT1[IDIndex,j]];
    FOVStar2:=OMAT[MAT1[IDIndex,j]];
  end
  else
  begin
    writeln('O my ...');
    readln;
  end;

end;

{
writeln(i,'. ',OMAT[i],' ',RAfov[OMAT[i]]:3:3,' ',
GMAT[i],' ',RACat[GMAT[i]]:3:3,' ',NASS1[i],' ** ');

for j:=1 to 19 do
begin

  write(MAT1[i,j], ' : ');

end;

writeln;
writeln(k,'. ',OMAT[k],' ',RAfov[OMAT[k]]:3:3,' ',
GMAT[k],' ',RACat[GMAT[k]]:3:3,' ',NASS1[k],' ** ');

```

```

for j:=1 to 19 do
begin

  write(MAT1[k,j], ' : ');

end;

writeln;
writeln('Finito!');
}

writeln('Huh?');
end;

```

```

(*****
  PROCEDURE NAME : Attitude
  DATE: 8/7/93
  GENERAL DESCRIPTION: Attitude determination from
                      positive star id.
  ARGUMENTS:
  RETURN VALUE:
  PSEUDO CODE:
  NOTES:
*****)

```

```

procedure Attitude;

var

theta, phi          : real;
x1, x2, y1, y2, xa, xb, ya, yb : real;
dp, dy, dr         : real;
x1n, x2n, y1n, y2n : real;
A, B, C, D         : real;
s                  : string;
aa                 : real;
i, j               : integer;
ccc                : integer;

```

```

begin

readln;

if ID=True then
begin

  results;

  y1:=YFOV[FOVStar1];
  y2:=YFOV[FOVStar2];
  x1:=XFOV[FOVStar1];
  x2:=XFOV[FOVStar2];

  ya:=YCat[CatStar1];

```

```

yb:=YCat[CatStar2];
xa:=XCat[CatStar1];
xb:=XCat[CatStar2];

A:=y2-y1;
B:=x2-x1;
C:=yb-ya;
D:=xb-xa;

writeln('A = ',A:3:3);
writeln('B = ',B:3:3);
writeln('C = ',C:3:3);
writeln('D = ',D:3:3);

writeln(x1:3:3,',',y1:3:3,' ',x2:3:3,',',y2:3:3);
writeln(xa:3:3,',',ya:3:3,' ',xb:3:3,',',yb:3:3);
writeln;

writeln(RACat[CatStar1]:3:3,',',DecCat[CatStar1]:3:3,' ',
RACat[CatStar2]:3:3,',',DecCat[CatStar2]:3:3);
writeln(RAFOV[FOVStar1]:3:3,',',DecFOV[FOVStar1]:3:3,' ',
RAFOV[FOVStar2]:3:3,',',DecFOV[FOVStar2]:3:3);
writeln;

theta := arctan((A)/(B));
writeln('theta = ',theta/pi*180:3:3);

phi := arctan((C)/(D));
writeln('phi = ',phi/pi*180:3:3);

{ check quadrants }

if (A>0) and (B>0) and (D>0) then dp := theta-phi;
if (A>0) and (B>0) and (D<0) then dp := -pi+theta-phi;
if (A>0) and (B<0) and (D>0) then dp := pi+theta-phi;
if (A>0) and (B<0) and (D<0) then dp := theta-phi;
if (A<0) and (B<0) and (D>0) then dp := -pi+theta-phi;
if (A<0) and (B<0) and (D<0) then dp := theta-phi;
if (A<0) and (B>0) and (D>0) then dp := theta-phi;
if (A<0) and (B>0) and (D<0) then dp := pi+theta-phi;

if dp>pi then dp := dp-2*pi;

{ **** Old Stuff **** }

if (A>0) and (B>0) then theta := -theta;
if (A>0) and (B<0) then theta := theta-pi;
if (A<0) and (B<0) then theta := theta-pi;

if (C>0) and (D<0) then phi:= -phi-pi;
if (C<0) and (D<0) then phi:= -phi-pi;

phi:=2*pi-phi;

writeln;
writeln('theta = ',theta/pi*180:3:3);
writeln('phi = ',phi/pi*180:3:3);
writeln;

pitch error

dp := (phi-theta);

}

```

```

writeln('dp = ',dp/pi*180:3:3);

x1n:=(cos(dp)*x1)+(sin(dp)*y1);
y1n:=(-sin(dp)*x1)+(cos(dp)*y1);
x2n:=(cos(dp)*x2)+(sin(dp)*y2);
y2n:=(-sin(dp)*x2)+(cos(dp)*y2);

{ roll error }
dr:=y2n-yb;
(dr:=y1n-ya;)

{ yaw error }

dy:=x2n-xb;
(dy:=x1n-xa;)

{
writeln('roll = ',dr:3:6);
writeln('pitch = ',dp/pi*180:3:6);
writeln('yaw = ',dy:3:6);
}

write(tests,dy);
dp:=dp/pi*180;
write(tests,dp);
write(tests,dr);

ccc:=0;

readln;

for i:= 1 to NassLength do
begin
if NASS1[i]>=0 then
begin
ccc:=ccc+1;
write(ccc,'. ',i,' ',NASS1[i],' * ');
for j:= 1 to 19 do
begin
write(MAT1[i,j],' ');
end;
write(' # ',RACat[GMAT[i]]:3:3,', ',RAFOV[OMAT[i]]:3:3);
readln;
end;
end;

writeln;
readln;

end
else
begin
a:=0;
write(tests,a);
write(tests,a);
write(tests,a);
end;

end;

(*****
PROCEDURE NAME :
DATE:
GENERAL DESCRIPTION:

```

ARGUMENTS:  
RETURN VALUE:  
PSEUDO CODE:  
NOTES:

\*\*\*\*\*)

procedure P2;

begin

end;

END.

## Appendix B Program Listings

```

unit Aobserve;

interface

uses

Graph, Crt, AShare, AMatch;

var

niks          : integer;

procedure Get_FOV_Data;
procedure Calc_FOV_Lines;
procedure Show_FOV_Lines(total : integer);
procedure Sort_FOV_Lines;
procedure Sort_2_FOV_Lines;

Implementation

var

Xcom, Ycom          : array[1..10] of real;
xc1d, xc2d, xc3d, xc4d : real;
yc1d, yc2d, yc3d, yc4d : real;
xc1, xc2, xc3, xc4   : integer;
yc1, yc2, yc3, yc4   : integer;
mtopA, mtopB, mtopC  : real;
mbotA, mbotB, mbotC  : real;
s1                   : string;

{*****}
{ Rotate the square FOV area by the angle Bpitch }
{*****}

procedure Calculate_FOV;

begin

xc1d := cos(Bpitch)*(-(wfov/2))+sin(Bpitch)*(+(wfov/2));
xc2d := cos(Bpitch)*(-(wfov/2))+sin(Bpitch)*(-(wfov/2));
xc3d := cos(Bpitch)*(+(wfov/2))+sin(Bpitch)*(-(wfov/2));
xc4d := cos(Bpitch)*(+(wfov/2))+sin(Bpitch)*(+(wfov/2));

yc1d := -sin(Bpitch)*(-(wfov/2))+cos(Bpitch)*(+(wfov/2));
yc2d := -sin(Bpitch)*(-(wfov/2))+cos(Bpitch)*(-(wfov/2));
yc3d := -sin(Bpitch)*(+(wfov/2))+cos(Bpitch)*(-(wfov/2));
yc4d := -sin(Bpitch)*(+(wfov/2))+cos(Bpitch)*(+(wfov/2));

end;

{*****}

Rotate procedure for cross
{*****}

procedure Rotate(var q1, q2:real);

var

q3          : real;

begin

q3:=q1;
q1:= cos(Bpitch)*q3+sin(Bpitch)*q2;

```

```

q2:= -sin(Bpitch)*q3+cos(Bpitch)*q2;
end;

{*****}
{ Swap procedure for sorting procedure      }
{*****}

procedure swap(var a,b:real);
var
t      : real;
begin
t:=a;
a:=b;
b:=t;
end;

{*****}
{ Sort the x co-ords in ascending order of distance }
{*****}

procedure Sort_Coords;
var
j,k      : integer;
sorted   : boolean;
begin
k := 4;
sorted := false;
while (k>0) and (not sorted) do
begin
sorted := true;
for j:= 1 to k-1 do
begin
if Xcom[j]>Xcom[j+1]then
begin
swap(Xcom[j],Xcom[j+1]);
swap(Ycom[j],Ycom[j+1]);
sorted := false
end;
end;
k := k-1
end;
end;

{*****}
{ Calculate the border of the selected FOV      }
{*****}

procedure Calculate_Border_FOV;

begin
if Bpitch<>0 then

```

```

begin
  if Bpitch>0 then
    begin
      mtopA := (Ycom[2]-Ycom[1])/(Xcom[2]-Xcom[1]);
      mtopB := (Ycom[4]-Ycom[2])/(Xcom[4]-Xcom[2]);
      mtopC := mtopB;
      mbotA := (Ycom[3]-Ycom[1])/(Xcom[3]-Xcom[1]);
      mbotB := mbotA;
      mbotC := (Ycom[4]-Ycom[3])/(Xcom[4]-Xcom[3]);
    end
  else
    begin
      mtopA := (Ycom[3]-Ycom[1])/(Xcom[3]-Xcom[1]);
      mtopB := mtopA;
      mtopC := (Ycom[4]-Ycom[3])/(Xcom[4]-Xcom[3]);
      mbotA := (Ycom[2]-Ycom[1])/(Xcom[2]-Xcom[1]);
      mbotC := (Ycom[4]-Ycom[2])/(Xcom[4]-Xcom[2]);
      mbotB := mbotC;
    end;
  end;
end;

(*****
{ Display the actual FOV
}
*****)

procedure Get_FOV_Data;

var

x2,y2,del                : integer;
DecTop,DecBot            : real;
vlag1                    : integer;
dum1,dum2,dum3,dum4,dum5 : real;
C1,C2,C3,C4              : real;
CA,CB,CC,CD,CE,CF       : real;
RAshow                   : real;
Decshow                  : real;
Xarrow,Yarrow            : array[1..6] of real;
XarrowSC,YarrowSC        : array[1..6] of integer;
Xasp,Yasp                : word;
x                         : integer;
count                    : integer;
xB,yB                    : integer;
i                         : integer;
MagFOVLimit1             : real;

begin

  vlag1 := 0;
  MagFOVLimit1 := 3.5;

  { Display borders of commanded FOV on star map}

  calculate_fov;

  { Sort the coords according to x-value }

  Xcom[1] := xc1d;
  Xcom[2] := xc2d;
  Xcom[3] := xc3d;
  Xcom[4] := xc4d;

  Ycom[1] := yc1d;
  Ycom[2] := yc2d;
  Ycom[3] := yc3d;
  Ycom[4] := yc4d;

```

```

Sort_Coords;

( Calculate inclinations of sides )

Calculate_Border_FOV;

if Bpitch>0 then
begin

C1 := Ycom[1]-mtopA*Xcom[1];
C2 := Ycom[2]-mtopB*Xcom[2];
C3 := Ycom[3]-mbotA*Xcom[3];
C4 := Ycom[4]-mbotC*Xcom[4];

CA := C1;
CB := C3;
CC := C2;
CD := C3;
CE := C2;
CF := C4;

if Ycom[3]>Ycom[2] then
begin
vlag1 := 1;
dum1 := mtopA;
dum2 := mtopB;
dum3 := mtopC;

mtopA := mbotA;
mtopB := mbotB;
mtopC := mbotC;

mbotA := dum1;
mbotB := dum2;
mbotC := dum3;

( mtopB := mtopA;
  mbotB := mbotC; )

C1 := Ycom[3]-mtopC*Xcom[3];
C2 := Ycom[1]-mtopA*Xcom[1];
C3 := Ycom[2]-mbotC*Xcom[2];
C4 := Ycom[1]-mbotA*Xcom[1];

CA := C2;
CB := C4;
CC := C2;
CD := C3;
CE := C1;
CF := C3;

end;
end

else
begin

C1 := Ycom[3]-mtopC*Xcom[3];
C2 := Ycom[1]-mtopA*Xcom[1];
C3 := Ycom[2]-mbotC*Xcom[2];
C4 := Ycom[1]-mbotA*Xcom[1];

CA := C2;
CB := C4;
CC := C2;
CD := C3;
CE := C1;
CF := C3;

if Ycom[2]>Ycom[3] then
begin

```

## Appendix B Program Listings

```

vlag1 := 1;
dum1 := mtopA;
dum2 := mtopB;
dum3 := mtopC;

mtopA := mbotA;
mtopB := mbotB;
mtopC := mbotC;

mbotA := dum1;
mbotB := dum2;
mbotC := dum3;

(
    mtopB := mtopA;
    mbotB := mbotC;)

C1 := Ycom[1]-mtopA*Xcom[1];
C2 := Ycom[2]-mtopC*Xcom[2];
C3 := Ycom[1]-mbotA*Xcom[1];
C4 := Ycom[3]-mbotC*Xcom[3];

CA := C1;
CB := C3;
CC := C2;
CD := C3;
CE := C2;
CF := C4;
end;
end;

if Bpitch=0 then
begin
mtopA := 10000;
mbotA := 0;
mtopB := 0;
mbotB := 0;
mtopC := 0;
mbotC := 10000;
end;

{ Corners of commanded FOV }

{ Inclinations of sides of commanded FOV }

{ Find stars in commanded FOV }

count := 0;

writeln('Ncat = ',Ncat);

for i := 1 to Ncat do
begin
if (Racat[i]<Xcom[4]+BRA) and (Racat[i]>Xcom[1]+BRA) then
begin
if (((((Racat[i]<Xcom[2]+BRA) and (Deccat[i]<(((Racat[i]-
BRA)*mtopA)+CA)+BDec) and
(Deccat[i]>(((Racat[i]-BRA)*mbotA)+CB)+BDec)) or
((Racat[i]>Xcom[2]+BRA) and (Racat[i]<=Xcom[3]+BRA) and
(Deccat[i]<(((Racat[i]-BRA)*mtopB)+CC)+BDec) and
(Deccat[i]>(((Racat[i]-BRA)*mbotB)+CD)+BDec)) or
((Racat[i]>Xcom[3]+BRA) and (Racat[i]<Xcom[4]+BRA) and
(Deccat[i]<(((Racat[i]-BRA)*mtopC)+CE)+BDec) and
(Deccat[i]>(((Racat[i]-BRA)*mbotC)+CF)+BDec))) and
(magcat[i]<MagFOVLimit )) and (count<NIDMax)) then
begin
count := count+1;

```

## Appendix B Program Listings

```

    RAFOV[count] := RAcats[i];
    DecFOV[count] := Deccats[i];
    MagFOV[count] := magcats[i];

    RAslow := cos(-Bpitch)*(RAFOV[count]-BRA)+sin(-Bpitch)*(DecFOV[count]-
BDec);
    Decslow := -sin(-Bpitch)*(RAFOV[count]-BRA)+cos(-
Bpitch)*(DecFOV[count]-BDec);

    writeln('XFOV = ',XFOV[count]:3:3,'    YFOV = ',YFOV[count]:3:3);
    XFOV[count]:=RAslow;
    YFOV[count]:=Decslow;

    end;
end;
end;

NFOV:=count;
FOVStarLength:=NFOV;

( repeat until keypressed; )

end;

procedure Show_FOV_Lines(total : integer);

var

i          : integer;
ttotal    : integer;

begin

{
writeln('Line  ', 'Star 1  ', 'X  ', ' Y  ', ' mv ', ' Star 2  ', 'X  ', '
Y', ' mv');
}

ttotal := 0;
for i := 1 to total do
begin
if ((magfov[SNFOV[i,1]]<>10) and (magfov[SNFOV[i,2]]<>10)) then
begin
ttotal:=ttotal+1;
Lfov[ttotal]:=Lfov[i];
SNFOV[ttotal,1]:=SNFOV[i,1];
SNFOV[ttotal,2]:=SNFOV[i,2];

{
writeln(tttotal, ' ', Lfov[i]:3:3, ' ', SNFOV[i,1], '
', RAfov[SNFOV[i,1]]:3:3, ' ', Decfov[SNFOV[i,1]]:3:3, ' ',
magfov[SNFOV[i,1]]:3:3, ' ', SNFOV[i,2], ' ', RAfov[SNFOV[i,2]]:3:3, '
', Decfov[SNFOV[i,2]]:3:3, ' ', magfov[SNFOV[i,2]]:3:3);
}

end;
end;

{
writeln;
writeln('There are ', tttotal, ' FOV lines');
}

FOVnewlength:=ttotal;

end;

{*****}
{ Swap procedure for sorting procedure }

```

```

(*****)

procedure Swapi(var a,b:integer);
var
t: integer;
begin
  t:=a;
  a:=b;
  b:=t;
end;

procedure Sort_FOV_Lines;
var

j,k          : integer;
sorted       : boolean;

begin
  k := FOVnewlength-1;

  {
  writeln('k = ',newlength);
  }

  sorted := false;
  while (k>0) and (not sorted) do
  begin
    sorted := true;
    for j:= 1 to k do
    begin
      if Lfov[j]>Lfov[j+1] then
      begin
        Swap(Lfov[j],Lfov[j+1]);
        Swapi(SNFOV[j,1],SNFOV[j+1,1]);
        Swapi(SNFOV[j,2],SNFOV[j+1,2]);
        sorted := false
      end;
    end;
    k := k-1
  end;

end;

procedure Calc_FOV_Lines;
var

i,j          : integer;
LfovF       : real;
tel         : integer;

begin
  tel:=0;
  FOVlength:=NFOV;

  FOVStarLength:=NFOV;

  {
  writeln('length = ',FOVlength);
  }

  for i := 1 to FOVlength do
  begin
    for j := 1 to FOVlength do
    begin

```

## Appendix B Program Listings

```

    LfovF:=sqrt(((RAfov[i]-RAfov[j])*(RAfov[i]-RAfov[j]))+((Decfov[i]-
Decfov[j])*(Decfov[i]-Decfov[j])));
    if ((LfovF < 0.05) and (i<>j)) then
    begin
        if magfov[i]>magfov[j] then
        begin
            magfov[i] := 10;
        end
        else
        begin
            magfov[j]:=10;
        end;
    end;
    if ((LfovF < 14.1421) and (LfovF >= 0.05)) then
    begin
        tel:=tel+1;
        Lfov[tel]:=LfovF;
        SNFOV[tel,1]:=i;
        SNFOV[tel,2]:=j;
    end;
end;

(
writeln;
)

FOVnewlength := tel;

(
writeln('There are ',tel,' FOV lines');
)

end;

```

```

procedure Sort_2_FOV_Lines;

```

```

var

```

```

i : integer;
c : integer;

```

```

begin

```

```

    c:=1;

```

```

    for i:=1 to round(FOVnewlength/2) do

```

```

    begin

```

```

        Lfov[i]:=Lfov[c];
        SNFOV[i,1]:=SNFOV[c,1];
        SNFOV[i,2]:=SNFOV[c,2];

```

```

        (
        writeln(Lfov[i]);
        )

```

```

        c:=c+2;

```

```

    end;

```

```

    for i:=1 to round(FOVnewlength/2) do

```

```

    begin

```

```

        Lfov[i]:=Lfov[i];
        SNFOV[i,1]:=SNFOV[i,1];
        SNFOV[i,2]:=SNFOV[i,2];

```

```

    end;

```

```

    FOVnewnewlength:=i;

```

```

    FOVLineLength:=i;

```

## Appendix B Program Listings

```
{  
  WriteLn('OK');  
}
```

```
end;
```

```
end.
```

```
unit Aout;

Interface

uses

Dos, Crt, AShare;

procedure Show_Files;

Implementation

procedure Show_Files;

var

wat          : real;
wats         : string;
d            : integer;
ss           : string;
i, code     : integer;

begin

  reset(week);
  reset(tests);
  reset(logfile);

  writeln('week.dat');
  writeln;

  while NOT EoF(week) do
  begin
    d:=0;
    while d<>9 do
    begin
      d:=d+1;
      read(week, wat);
      writeln(wat:3:3, ' ');
    end;
    writeln;
  end;

  readln;

  writeln('tests.dat');
  writeln;

  while NOT EoF(tests) do
  begin
    d:=0;
    while (d<>8) AND (NOT EoF(tests)) do
    begin
      d:=d+1;
      read(tests, wat);
      if ((wat-round(wat))=0) then
      begin
        write(wat:3:0, ' ');
      end
      else
      begin
        write(wat:3:3, ' ');
      end;
    end;
    writeln;
  end;
end;
```

## Appendix B Program Listings

```
readln;

writeln('logfile.dat');
writeln;

while NOT EoF(logfile) do
begin
  read(logfile,wats);
  write(wats);
  writeln;
end;

readln;

end;

end.
```

```

unit Ashare;

Interface

uses

  Crt;

var

  week           : file of real;
  tests          : file of real;
  logfile        : file of string;
  WFOV           : real;
  WCat           : real;
  RACat, DecCat, MagCat : array[1..100] of real;
  RAFOV, DecFOV, MagFOV : array[1..8] of real;
  LCat           : array[1..1500] of real;
  LFOV           : array[1..60] of real;
  SNCat         : array[1..1500,1..2] of integer;
  SNFOV         : array[1..60,1..2] of integer;
  Senacc        : real;
  RACatC, DecCatC, MagCatC : real;
  MagCatLimit   : real;
  CloseR, CloseM : real;
  SelectMethod  : integer;
  RCatLimit     : real;
  Gd, Gm        : integer;
  GraphPath     : string;
  StarFile      : file of real;
  NCat          : integer;
  DX1, DX2      : integer;
  DY1, DY2      : integer;
  XSpace, YSpace : integer;
  MagLimit1, MagLimit2 : real;
  NewLength     : integer;
  Length        : integer;
  NewNewLength  : integer;
  CatStarLength : integer;
  CatStarLengthI : real;
  CatLineLength : integer;
  CatLineLengthI : real;
  BRA, BRA0, BDec : real;
  BPitch        : real;
  MagFOVLimit   : real;
  XFOV, YFOV    : array[1..8] of real;
  XCat, YCat    : array[1..100] of real;
  NFOV          : integer;
  FOVLength     : integer;
  FOVNewLength  : integer;
  FOVNewNewLength : integer;
  FOVStarLength : integer;
  FOVStarLengthI : real;
  FOVLineLength : integer;
  FOVLineLengthI : real;
  TDis          : real;
  TMag          : real;
  NIDMax, NIDMin : integer;
  NSatLimit     : integer;
  InitThreshold : integer;
  SizeLimit     : integer;
  Iterations    : boolean;
  BestSelect    : boolean;
  BestFirst     : boolean;
  TIntegrate    : real;
  DNoise        : real;
  DSNU          : real;
  MinObSize, MaxObSize : integer;
  Oblateness    : real;
  CatStar1, CatStar2 : integer;
  FOVStar1, FOVStar2 : integer;

```

```
ID : boolean;
```

```
procedure Get_Cat_Parameters;
procedure Get_FOV_Parameters;
procedure Get_Match_Parameters;
procedure Get_Development_Parameters;
procedure Get_CCD_Parameters;
procedure Get_Grow_Parameters;
procedure Get_Auto_Parameters;
procedure Default_Parameters;
```

#### Implementation

```
{*****}
{ Get user input for map area }
{ The accuracy of the sensors defines the }
{ size of the surrounding area that has to }
{ searched. }
{*****}
```

```
procedure Get_Cat_Parameters;
```

```
begin
```

```
  ClrScr;
  writeln('Catalogue Parameters');
  writeln('-----');
  writeln;
  write('Accuracy of other sensors : ');
  readln(senacc);
  write('RAc (h) : ');
  readln(RACatC);
  write('Decc (deg) : ');
  readln(DecCatC);
  write('Limiting mag : ');
  readln(MagCatLimit);
  write('Close dist. star radius : ');
  readln(CloseR);
  write('Close mag margin : ');
  readln(CloseM);
  write('Selection method (1 or 2) : ');
  readln(SelectMethod);
  write('Pair distance limit : ');
  readln(RCatLimit);
  RACatC := (RACatC*15);
```

```
end;
```

```
{*****}
{ Get user input for actual FOV }
{*****}
```

```
procedure Get_FOV_Parameters;
```

```
begin
```

```
  ClrScr;
  writeln('Field Of View Parameters');
  writeln('-----');
  writeln;
  write('Boresight RA : ');
  readln(BRA);
  BRA:=BRA*15;
  BRA0:=BRA;
  write('Boresight Dec : ');
```

```

readln(BDec);
write('Pitch angle (deg): ');
readln(BPitch);
Bpitch:=BPitch/180*pi;
write('Magnitude limit : ');
readln(MagFOVLimit);
write('FOV angle (deg) : ');
readln(WFOV);

end;

{*****}
{ Get user input for matching }
{*****}

procedure Get_Match_Parameters;

var

Answer1      : string;

begin

  DNoise:=15;
  DSNU:=15;
  ClrScr;
  writeln('Match Algorithm Parameters');
  writeln('-----');
  writeln;
  write('Max number of stars for ID : ');
  readln(NIDmax);
  write('Saturation limit number : ');
  readln(NSatLimit);
  write('Pair distance tolerance : ');
  readln(TDis);
  write('Magnitude tolerance : ');
  readln(TMag);
  write('Min number of stars for ID : ');
  readln(NIDMin);
  write('Initial match threshold : ');
  readln(InitThreshold);
  write('Match group size limit : ');
  readln(SizeLimit);
  Iterations:=False;
  BestSelect:=False;
  BestFirst:=False;
  write('Iterations permitted (y/n) : ');
  readln(Answer1);
  if Answer1='y' then
  begin
    Iterations:=True;
  end;
  write('Best group selection (y/n) : ');
  readln(Answer1);
  if Answer1='y' then
  begin
    BestSelect:=True;
  end;
  write('Select first time (y/n) : ');
  readln(Answer1);
  if Answer1='y' then
  begin
    BestFirst:=True;
  end;
end;

procedure Get_Development_Parameters;

begin

```

```
{Catalogue}

Senacc:=5;
RACatC:=10*15;
DecCatC:=10;
MagCatLimit:=6;
CloseR:=0.08;
CloseM:=0.77;
SelectMethod:=1;
RCatLimit:=14.147;

{FOV}

BRA:=10.4*15;
BRA0:=BRA;
BDec:=11;
BPitch:=23.45/180*pi;
MagFOVLimit:=6;
WFOV:=10;

{Match}

NIDMax:=8;
NSatLimit:=3;
TDis:=0.2;
TMag:=2;
NIDMin:=3;
InitThreshold:=5;
SizeLimit:=11;
Iterations:=True;
BestSelect:=True;
BestFirst:=True;

{CCD FOV}

DNoise:=15;
DSNU:=15;

end;

procedure Get_CCD_Parameters;

begin

ClrScr;
writeln('Charge Coupled Device Parameters');
writeln('-----');
writeln;
write('Integration time (msec) : ');
readln(TIntegrate);
write('Dark noise amplitude      : ');
readln(DNoise);
write('NDSNU amplitude              : ');
readln(DSNU);

end;

procedure Get_Grow_Parameters;

begin

ClrScr;
writeln('Growing Algorithm Parameters');
writeln('-----');
writeln;
write('Minimum object size (pixels) : ');
readln(MinObSize);
```

## Appendix B Program Listings

```

write('Maximum object size (pixels) : ');
readln(MaxObSize);
write('Object X/Y (=<1)           : ');
readln(Oblateness);

end;

procedure Get_Auto_Parameters;

begin

  ClrScr;

  (
  writeln('Star Match Test');
  writeln('-----');
  writeln;
  write('Center RA           : ');
  readln(RACatC);
  write('Center Dec          : ');
  readln(DecCatC);
  write('Magnitude Limit (Map)  : ');
  readln(MagCatLimit);
  write('Magnitude Limit (FOV)  : ');
  readln(MagFOVLimit);
  write('FOV width              : ');
  readln(WFOV);
  write('Close Star Dist. Limit : ');
  readln(CloseR);
  write('Pair Distance Tolerance : ');
  readln(TDis);

  )

end;

procedure Default_Parameters;

begin

  (Catalogue)

  RACatC:=10;
  DecCatC:=0;
  MagCatLimit:=5.5;
  MagFOVLimit:=5.5;
  WFOV:=10;
  CloseR:=0.08;
  TDis:=0.2;
  Senacc:=5;
  RACatC:=RACatC*15;
  CloseM:=0.77;
  SelectMethod:=1;
  RCatLimit:=Sqrt((WFOV*WFOV)+(WFOV*WFOV));
  Bpitch:=-30.45/180*pi;
  Bpitch:=-Bpitch;

  (Match)

  NIDMax:=8;
  NSatLimit:=3;
  TMag:=2;
  NIDMin:=3;
  InitThreshold:=5;
  SizeLimit:=11;
  Iterations:=True;

```

## Appendix B Program Listings

```
BestSelect:=True;  
BestFirst:=True;  
  
write (week, RACatC, DecCatC, MagCatLimit, MagFOVLimit, WFOV, CloseR, TDis);  
  
end;  
  
end.
```

```

program Starfind;

uses

  Dos, Graph, Crt, Guide, Observe, Match, Results, Share, Demo, CCD;

var

  Answer          : string;
  Flag, Flag1, Flag2 : boolean;

  {*****}
  { Main Program }
  {*****}
  {ww}

BEGIN

  Flag:=False;

  while Flag=False do
  begin

    Clrscr;

    Flag1:=False;
    Flag2:=False;

    writeln('Attitude Simulation');
    readln;

    while Flag1=False do
    begin

      Flag2:=False;

      while Flag2=False do
      begin
        Get_Cat_Parameters;
        writeln;
        write('Accept Catalogue Parameters? : ');
        readln(Answer);
        if (Answer='y') or (Answer='Y') then
        begin
          Flag2:=True;
        end;
      end;
    end;

    Flag2:=False;

    while Flag2=False do
    begin
      Get_FOV_Parameters;
      writeln;
      write('Accept Field Of View Parameters? : ');
      readln(Answer);
      if (Answer='y') or (Answer='Y') then
      begin
        Flag2:=True;
      end;
    end;

    Flag2:=False;

    while Flag2=False do
    begin
      Get_Match_Parameters;

```

```

writeln;
write('Accept Match Parameters? : ');
readln(Answer);
if (Answer='y') or (Answer='Y') then
begin
  Flag2:=True;
end;
end;

writeln;
write('Accept all parameters? : ');
readln(Answer);

if (Answer='y') or (Answer='Y') then
begin
  Flag1:=True;
end;
end;

ID:=False;

{ Get_Development_Parameters;}

Get_Cat_Data;

{ readln;}

Get_FOV_Data;

{ readln;}

Show_CCD_FOV;

{ readln;}

Calc_Cat_Lines;
Show_Cat_Lines(Newlength);
Sort_Cat_Lines;
Show_Cat_Lines(NewLength);
Sort_2_Cat_Lines;
Show_Cat_Lines(NewNewLength);

{ readln;}

Calc_FOV_Lines;
Show_FOV_Lines(FOVNewlength);
Sort_FOV_Lines;
Show_FOV_Lines(FOVNewLength);
Sort_2_FOV_Lines;
Show_FOV_Lines(FOVNewNewLength);

{ readln;}

Show_Cat_Parameters;
Show_FOV_Parameters;

readln;

Display_Cat_Lines;
Display_FOV_Lines;

readln;

CloseGraph;

BezMatch;

Get_Cat_Data;
Get_FOV_Data;
Show_CCD_FOV;
Show_Cat_Parameters;

```

```
Show_FOV_Parameters;
Show_Results;
readln;
CloseGraph;
Flag:=True;

writeln;
write('Again? : ');
readln(Answer);

if (Answer='y') or (Answer='Y') then
begin
  Flag:=False;
  Flag1:=False;
end;

end;

END.
```

## Appendix B Program Listings

```
program auto;

uses

  Dos, Graph, Crt, AGuide, AObserve, AMatch, AShare, AOut;

var

  x          : integer;
  xI         : real;
  dDec       : real;

BEGIN

  assign(week, 'c:\tp\data\week1.dat');
  assign(tests, 'c:\tp\data\test1.dat');
  assign(logfile, 'c:\tp\data\log1.dat');

  rewrite(week);
  rewrite(tests);
  rewrite(logfile);

  reset(week);
  reset(tests);
  reset(logfile);

  Get_Auto_Parameters;
  Default_Parameters;

  Get_Cat_Data;

  Calc_Cat_Lines;
  Show_Cat_Lines(Newlength);

  writeln('Newlength = ', Newlength);
  readln;

  Sort_Cat_Lines;
  Show_Cat_Lines(NewLength);
  Sort_2_Cat_Lines;
  Show_Cat_Lines(NewNewLength);

  CatLineLengthI:=CatLineLength;

  write(week, CatLineLengthI);
  writeln('CatLineLength= ', CatLineLength);
  readln;

  for x:=1 to 10 do
  begin

    writeln(x);

    Randomize;

    BRA:=RACatC-5+x;
    BDec:=DecCatC;

    xI:=x;

    write(tests, xI);
    write(tests, BRA, BDec);

    Get_FOV_Data;

    if NFOV>=3 then
    begin
      Calc_FOV_Lines;
      Show_FOV_Lines(FOVNewlength);
    end;
  end;
end;
```

```
Sort_FOV_Lines;
Show_FOV_Lines(FOVNewLength);
Sort_2_FOV_Lines;
Show_FOV_Lines(FOVNewNewLength);
end;

FOVStarLengthI:=FOVStarLength;
write(tests,FOVStarLengthI);
writeln('FOVStarLength = ',FOVStarLength);

FOVLineLengthI:=FOVLineLength;
write(tests,FOVLineLengthI);
writeln('FOVLineLength = ',FOVLineLength);

str(x,s);

write(logfile,s);

BezMatch;

end;

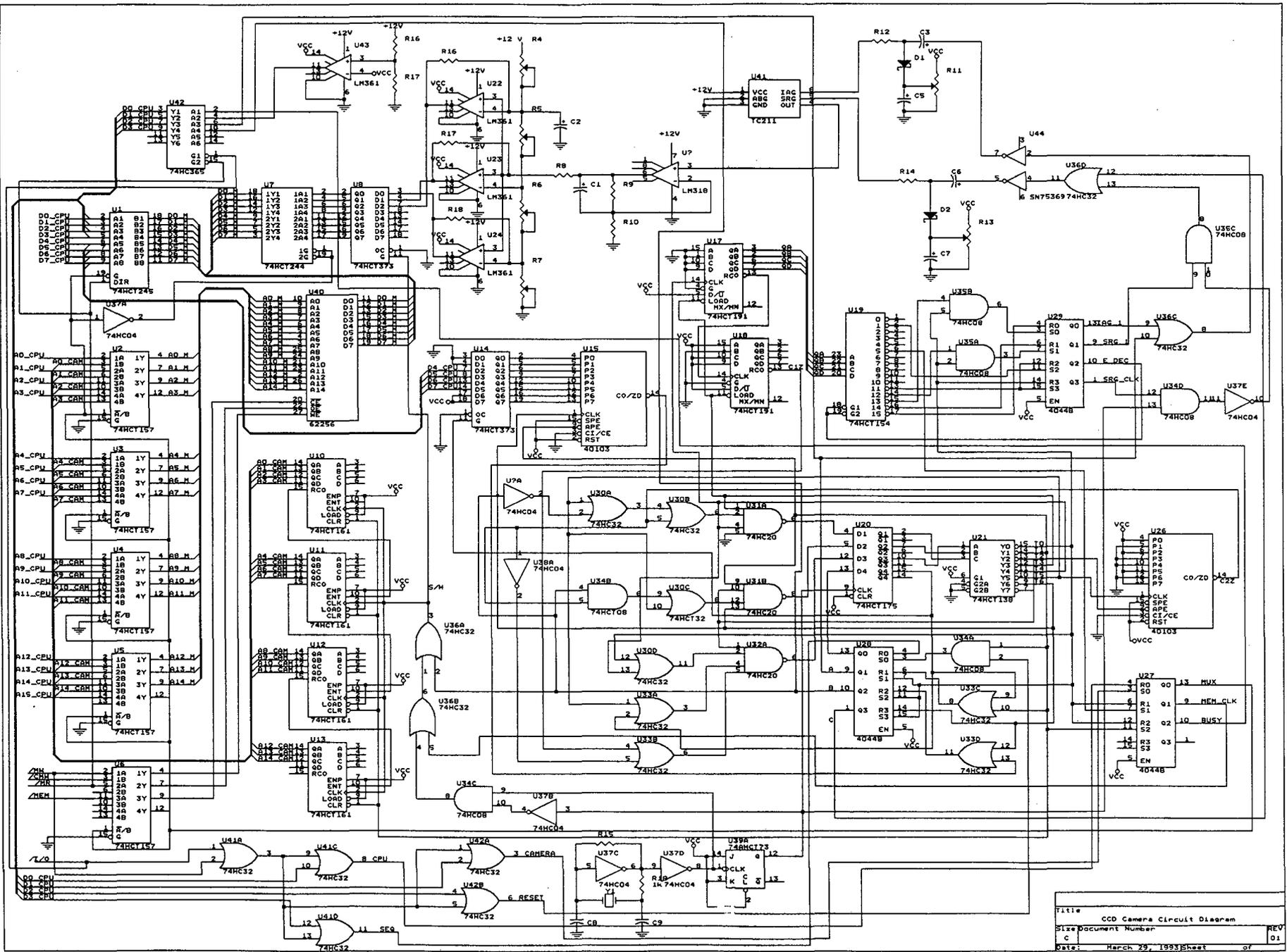
writeln('OK');
readln;

Show_Files;

close(week);
close(tests);
close(logfile);

END.
```

## Appendix C



Title: CCD Camera Circuit Diagram  
S/N: Document Number  
Date: March 29, 1993  
Sheet 01 of

## Table of Authorities

- Anderson D.S., Pollock T.C. and Junkins J.L. [1990]  
“A Testbed for Autonomous Star Pattern Recognition and Attitude Determination”, *AIAA*.
- Ballard D.H. and Brown C.M. [1982]  
*Computer Vision*, Prentice-Hall.
- Barbe D.F. [1976]  
“Imaging with Charge-Coupled Devices”, *AIAA Systems Design Driven by Sensors Conference*.
- Bokhove H, Jonker P.P., van der Kolk L.W. and Roos C. [1986]  
“The TPD Off-Axis Startracker (TOAST)”, *Proceedings of the Second International Symposium on Spacecraft Flight Dynamics*.
- Castleman K.R. [1987]  
*Digital Image Processing*, Prentice-Hall.
- EEV [1987]  
*CCD Imaging III*, EEV.
- EEV [1990]  
*CCD02-06 Series TV Image Sensor*, EEV.
- Chory M.A. [1986]  
“Satellite Autonomous Navigation - Status and History”, *IEEE Position Location and Navigation Symp.* pp. 110-121.
- Comer D.J. [1987]  
*Modern Electronic circuit Design*, Addison Wesley.
- Cox G.S., de Jager G. and Warner B. [1992]  
“A new method of Rotation, Scale and Translation Invariant Point Pattern Matching Applied to the Target Acquisition and Guiding Automatic Telescope”.
- Davidoff M.R. [1985]  
*The Satellite Experimenter's Handbook*, The American Radio Relay League
- Ginati A. [1991]  
“TUBSAT -1 Attitude Control and Stabilisation System”, *Technical University of Berlin*.
- Glass I. [1994]  
Correspondence on the response of CCDs to stellar electromagnetic radiation.

- Grossman S.B. [1984]  
“Fine guidance sensor design optimization for space infrared telescope facility (SIRTF)”, *SPIE Vol. 509 Cryogenic Optical Systems and Instruments*.
- Hecht E. [1987]  
*Optics Second Edition*, Addison Wesley.
- Jones B. [1990]  
“Attitude Determination Concepts for the Space Station Freedom”, *IEEE*.
- Kan D., Yang J., Ye P., Zhu Z. and Guo R [1992]  
“Star Referenced Autonomous Attitude Determination”, *IFAC workshop on Spacecraft Automation and On-Board Autonomous Mission Control*.
- Mano M. [1984]  
*Digital Design*, Prentice-Hall International Editions.
- Milne G.W. [1990]  
Technical Specification for SUNSAT.
- Moller K.D. [1988]  
*Optics*, University Science Books.
- The Math Works inc. [1989]  
*PC-MATLAB User Guide*, The Math Works inc.
- AIAA/USU [1990]  
“Proceedings of the 4th Conference on Small Satellites”, Volume II, AIAA/USU, August.
- Radbone J.M. [1987]  
“The UoSAT- 2 spacecraft CCD imaging and digital store/readout experiments”, *Journal of the Institution of Electronic and Radio Engineers, Vol. 57 No 5 (Supplement), September/October*, pp.S179-S183.
- LI- COR [1979]  
*Radiation Measurement*, LI- COR.
- Riedel J.E., Owen W.M., Stuve J.A., Synnott S.P. and Vaughan R.M. [1990]  
“Optical Navigation During the Voyager Neptune Encounter”, *AIAA-90-2877-CP*.
- Rosengren M. [1992]  
“ERS-1 - An Earth Observer that Exactly Follows its chosen Path”, *ESA bulletin, November 1992*.
- Shu F. [1982]  
*The Physical Universe An Introduction to Astronomy*. University Science Books

- Smith [1990]  
*Modern Optical Engineering*, McGraw-Hill.
- Texas Instruments [1987]  
*Optoelectronics and Image-Sensor Data Book*, Texas Instruments.
- Thomson Composants et Spatiaux [1974]  
*The CCD Image Sensor*, Thomson Composants et Spatiaux.
- Thorne D.J. , Jordan P.R. and Waltham N.R. [1986]  
“Optimisation of CCDs for astronomy - The RGO experience”, *Ex. Proceedings of the ESO workshop on optimization of CCD detectors for astronomy, 17-19 June*.
- Traynar C.P. [1982]  
“The development of a satellite-borne Earth imaging system for UOSAT”, *The Radio Electronic Engineer Vol 52. No 8/9, pp. 398-402*.
- Van Bezooijen R.W.H. [1989]  
“A Pattern Recognition Algorithm for Autonomous Attitude Determination”, *IFAC Automatic Control in Aerospace*.
- Van Bezooijen R.W.H. [1990]  
“Autonomous Star Tracker Development”, *IFAC Automatic Control in Aerospace*.
- Wertz J.R. [1986]  
*Spacecraft Attitude Determination and Control*, D. Reidel Publishing Company  
Boston U.S.A., Reprint 1986.