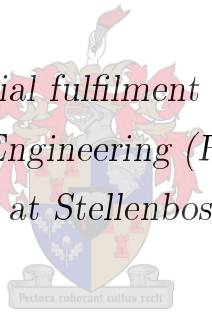
Development of *CubeStar*
A CubeSat-Compatible Star Tracker

Alexander Olaf Erlank



Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Research) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. W.H. Steyn

Department of Electrical and Electronic Engineering

December 2013

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: September 2013 .....

Abstract

Development of *CubeStar*
A CubeSat-Compatible Star Tracker

A. Erlank

Supervisor: Prof. W.H. Steyn

Thesis: MEng (E&E)

December 2013

The next generation of CubeSats will require accurate attitude knowledge throughout orbit for advanced science payloads and high gain antennas. A star tracker can provide the required performance, but star trackers have traditionally been too large, expensive and power hungry to be included on a CubeSat. The aim of this project is to develop and demonstrate a CubeSat compatible star tracker. Subsystems from two other CubeSat components, CubeSense and CubeComputer, were combined with a sensitive, commercial image sensor and low-light lens to produce one of the smallest star trackers in existence. Algorithms for star detection, matching and attitude determination were investigated and implemented on the embedded system. The resultant star tracker, named CubeStar, can operate fully autonomously, outputting attitude estimates at a rate of 1 Hz. An engineering model was completed and demonstrated an accuracy of better than 0.01 degrees during night sky tests.

Uittreksel

A. Erlank

Supervisor: Prof. W.H. Steyn

Tesis: MIng (E&E)

Desember 2013

Die volgende generasie van CubeSats sal akkurate orientasie kennis vereis gedurende 'n volle omwentelling van die aarde. 'n Sterkamera kan die vereiste prestasie verskaf, maar sterkameras is tradisioneel te groot, duur en krag intensief om ingesluit te word aanboord 'n CubeSat. Die doel van hierdie projek is om 'n CubeSat sterkamera te ontwikkel en te demonstreer. Substelsels van twee ander CubeSat komponente, CubeSense en CubeComputer, was gekombineer met 'n sensitiewe kommersiële beeldsensor en 'n lae-lig lens om een van die kleinste sterkameras op die mark te produseer. Algoritmes vir die ster opsporing, identifikasie en orientasie bepaling is ondersoek en geïmplementeer op die ingebedde stelsel. Die gevolglike sterkamera, genaamd CubeStar, kan ten volle outonoom orientasie afskattings lewer teen 'n tempo van 1 Hz. 'n Ingenieursmodel is voltooi en 'n akkuraatheid van beter as 0.01 grade is gedemonstreer.

Acknowledgements

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations and Acronyms

ADC	Analogue to digital converter
ADCS	Attitude determination and control system
APS	Active pixel sensor
BCT	Blue Canyon Technologies
CCD	Charge-coupled device
CMOS	Complementary metal-oxide-semiconductor
COTS	Commercial off-the-shelf
DCM	Direction cosine matrix
EBI	External Bus Interface
ECI	Earth-centred inertial
EDAC	Error detection and correction
EEPROM	Electrically erasable programmable read-only memory
ESL	Electronic Systems Laboratory
FOV	Field of view
FPGA	Field programmable gate array
GPIO	General purpose input/output
I^2C	Inter-integrated circuit
IC	Integrated circuit

JPL	Jet Propulsion Laboratory
LED	Light-emmitting diode
LEO	Low Earth orbit
LIS	Lost in space
MCU	Microcontroller
MIT	Massachusetts Institute of Technology
MOSFET	Metal-oxide-semiconductor field-effect transistor
MSB	Most significant byte
MST	Miniature star tracker
OBC	Onboard computer
PC	Personal computer
PCB	Printed circuit board
RMS	Root mean square
SEL	Single event latchup
SEU	Single event upset
SNR	Signal-to-noise ratio
SRAM	Static random access memory
SSTL	Surrey Satellite Technology Ltd
UART	Universal asynchronous receiver/transmitter
VHDL	VHSIC (Very high speed integrated circuits) hardware description language

Commonly Used Symbols

σ	Standard deviation
f	Focal length

Chapter 1

Introduction

The majority of modern satellites require knowledge of their orientation, or attitude, with respect to the earth at all times. This information is required to enable the satellite to point its solar arrays at the sun, its antennas at the earth and its scientific instruments at specific targets. Earthbound autonomous vehicles can make use of the gravity vector, which always points towards the centre of the earth, to determine their attitude. However, this vector is unavailable to satellites orbiting the earth. Instead, satellites make use of the earth's magnetic field and horizon, the sun and stars to determine their attitude.

A new class of nano-satellites, called CubeSats, is rapidly maturing. Traditionally, these small satellites have been primarily a teaching aid for universities training engineers. Their attitude determination and control systems (ADCSs) have often been crude, only partially stabilising the satellite. However, CubeSats are reaching the point of maturity where they could potentially take over the work of much larger satellites. If this is to be the future of CubeSats, their attitude determination and control systems will have to receive a major upgrade.

Sun and horizon sensors, magnetometers, magnetorquers and reaction wheels have all been miniaturised to the point where they can be used on board CubeSats. The majority of CubeSats currently in development aim to make use of these components to achieve full three-axis stabilisation. Using fine sun and horizon sensors, CubeSats can hope to achieve an attitude knowledge accuracy in the order of 0.1 degrees. However, this will only be possible in the sunlit part of the orbit, as the sun and horizon vectors are unavailable during eclipse. Once this performance has been proven in orbit, the next generation of CubeSats will require even higher performance, which will only be possible with a star tracker.

A star tracker makes use of the stars to determine the satellite's attitude. Star trackers can achieve accuracies two orders of magnitude better than other absolute attitude sensors and work throughout the orbit. Star trackers have traditionally been large, power hungry and expensive, prohibiting their use on board CubeSats. However, several key technologies have advanced to the point where a CubeSat compatible star tracker is possible.

This thesis describes the development of a CubeSat compatible, nano star tracker named *CubeStar*. It begins by discussing several important background topics, including coordinate systems, the CubeSat standard, current CubeSat ADCS capabilities and an introduction to star trackers. The broad aims of the CubeStar project are also discussed. In Chapter 3, several fundamental design decisions are made. These include choosing an image sensor and an appropriate field of view (FOV). Chapter 4 describes the star matching and attitude determination algorithms which turn a camera into a star tracker. The construction and testing of a complete engineering model of CubeStar are discussed in Chapters 5 and 6. Finally, the project is concluded and recommendations for the future are made in Chapter 7.

Chapter 2

Background

2.1 Coordinate Systems

Five coordinate systems are used throughout this thesis: image plane coordinates, sensor body coordinates, satellite body coordinates, orbit referenced coordinates and Earth-centred inertial coordinates. A brief introduction to these coordinates is given in this section. More details on each coordinate system and transformations between coordinate systems are given in the sections where those coordinates are used.

Image plane coordinates are two dimensional coordinates describing a location on the image sensor's surface. The coordinates can be expressed in either pixels or mm. The origin of the coordinate system is typically the point where the boresight axis intersects the image plane.

Sensor body coordinates originate at the optical centre of the star tracker lens and the z-axis is aligned with the lens boresight. Sensor body coordinates are typically expressed in Cartesian form.

Closely related to the sensor body coordinate system is the satellite body coordinate system. This coordinate system has its origin at the centre of mass of the satellite and is typically defined during the design of the satellite. The three principal axes of the satellite body coordinate system are typically defined to pass through three perpendicular faces of the satellite. The transformation matrix from sensor body coordinates to satellite body coordinates is fixed and is dependent of the mounting of the sensor on the satellite.

The orbit referenced coordinate system has its origin at the centre of mass of the satellite and it moves with the satellite as the satellite moves through its orbit. It rotates about its y-axis through an angle equal to the orbit true anomaly to keep its

z-axis nadir pointing. The y-axis points perpendicular to the orbit plane and the x-axis points in the direction of the velocity vector in a circular orbit. A satellite's attitude is often expressed as three angles, roll, pitch and yaw in the orbit referenced coordinate system.

The Earth-centred inertial (ECI) coordinate system is oriented using the earth's orbit plane and spin axis. Unlike the orbit referenced coordinate system, the ECI coordinate system is assumed to be fixed in inertial space. The z-axis points towards the north celestial pole and the x-axis is in the direction of the point where the earth's equator intersected the earth's orbit plane (Vernal Equinox) at 12:00 on 1 January 2000. This is referred to as the J2000 ECI coordinate system. This date is required as the earth's spin axis is not fixed in inertial space but precesses over time. This precession is slow enough that it is usually ignored for practical purposes. A new ECI coordinate system specification will be defined in a few decades to account for the new orientation of the earth's spin axis.

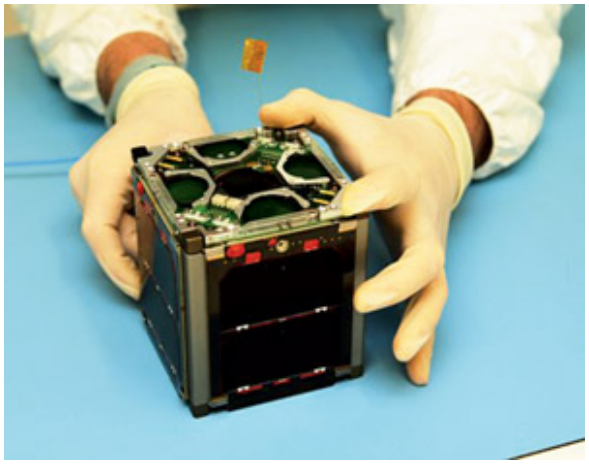
ECI coordinates are often given in terms of right ascension and declination, which are the celestial equivalents to longitude and latitude. These spherical coordinates are best understood if all stars are imagined to be on a sphere of unit radius, centred on the observer. Since the sphere has unit radius, any location on the sphere can be described using only two coordinates. Right ascension is measured from the vernal equinox (0 to 360 degrees) and declination is measured from the celestial equator (-90 to +90 degrees).

2.2 CubeSats

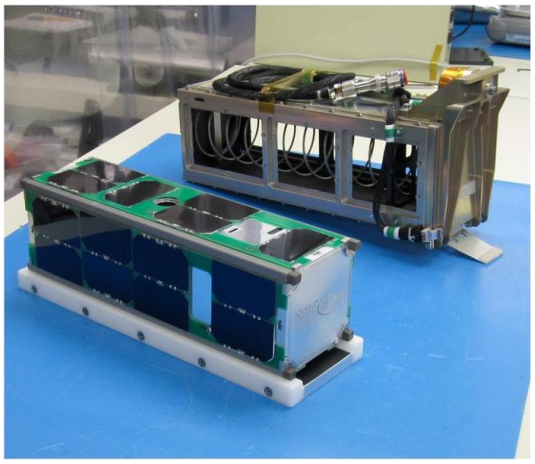
The CubeSat Project was started in 1999 as a collaborative project between California Polytechnic State University and Stanford University. The project's goal is to: "provide a standard for design of picosatellites to reduce cost and development time, increase accessibility to space, and sustain frequent launches" [4].

As specified by the CubeSat Design document, which is the primary output of the CubeSat Project, a CubeSat shall be a 10 cm cube which weighs no more than 1.33 kg. This is called a 1U (for unit) CubeSat. Other specifications describe the electrical bus, restrictions on materials, energy storage, radio frequency interference and location of centre of gravity. CubeSats can be scaled along one axis to become 2U (10 x 10 x 20 cm) or 3U (10 x 10 x 30 cm) CubeSats.

By adhering to these specifications, CubeSat builders have several advantages, includ-



(a) ZACube1



(b) 3U CubeSat and P-POD Deployer

Figure 2.1: South Africa's first 1U CubeSat, ZACube1 [1], and the Colorado Student Space Weather Experiment 3U CubeSat [2]

ing:

- Most CubeSat subsystems have a common interface and form factor. Therefore, by using a combination of custom built, reused, and purchased subsystems, the development time can be reduced.
- A large number of CubeSat subsystems are already available for purchase online.
- CubeSats are deployed using a common deployment mechanism called a P-POD, which negates the need to design a custom satellite-rocket interface.
- Thanks to the P-POD, CubeSats can be launched as secondary payloads, making launch less expensive.
- Access to the global CubeSat community, which believes in sharing information and experiences.

The first CubeSats were launched in 2003 and since then hundreds of CubeSats have been launched [5]. Traditionally, CubeSats have been built by universities as educational tools and have been simple by satellite standards. However, the complexity and ambition of CubeSat missions is increasing rapidly. CubeSats performing the job of much larger weather and communication satellites is foreseeable in the future.

Sensor	Typical Accuracy (deg)	Works in Eclipse	Successfully used on a CubeSat
Magnetometer	5	yes	yes
Coarse Sun	5	no	yes
Fine Sun	0.2	no	yes
Horizon	0.3	no	no
Star Tracker	better than 0.01	yes	no

Table 2.1: The typical accuracies of various satellite attitude sensors.

Accuracy (deg)	Minimum Sensors	Required Actuators	Achieved on a CubeSat
>5	none	permanent magnets or gravity gradient	yes
5-1	magnetometer coarse sun sensor	3 x magnetorquers opt. momentum wheel	yes
1-0.1	fine sun sensor horizon sensor	3 x reaction wheels	no
<0.1	star tracker	3 x reaction wheels	no

Table 2.2: The attitude sensors and actuators typically required to achieve a certain ADCS performance. Adapted from [3].

2.3 Current CubeSat ADCS Capabilities

An attitude determination and control system is composed of two subsystems, an attitude estimator and an attitude controller. The attitude estimator uses various sensors, such as sun and horizon sensors, to determine the satellite's current attitude. This information is used by the attitude controller to change the satellite's attitude to a desired attitude to point antennas or science instruments. The satellite's pointing accuracy is at best equal to the attitude estimation accuracy, but is usually an order of magnitude less accurate.

This thesis is concerned with developing a high accuracy attitude sensor for CubeSats. To understand the effect that this sensor will have on future CubeSat ADCS performance, a brief summary of CubeSat ADCS achievements to date is given below.

Table 2.1 displays the typical accuracies of various attitude sensors. Only absolute sensors are included in the table. All these sensors have been miniaturised enough to be used on CubeSats, however, no CubeSat to date has included a horizon sensor or star tracker. All the sensors with the exception of the star tracker provide only two-axis attitude information and only the magnetometer and star tracker work in eclipse.

The set of sensors and actuators included on a satellite is dependent on the required attitude accuracy, as shown in Table 2.2. In this table, attitude accuracy refers to attitude knowledge and stability. Full three-axis control is only possible using reaction wheels.

The most elementary form of attitude control, called passive control, cannot achieve accuracies better than approximately 5 degrees. Most CubeSats to date have been university projects with the aim of training new engineers. These CubeSats have no major scientific goal or experiment to conduct in orbit and as such, can use passive control. The most popular form of passive control is to include a permanent magnet and some hysteresis material on the CubeSat. The permanent magnet will cause the CubeSat to align itself with the earth's magnetic field. The hysteresis material is required to damp attitude oscillations. CubeSats with passive attitude control require no attitude sensors or actuators and will remain stable throughout the orbit. However, they have no ability to reorient themselves in space. The Colorado Student Space Weather Experiment CubeSat uses a passive magnetic stabilisation system to align itself within approximately 10 degrees of the Earth magnetic field [6].

Better performance can be achieved using active magnetic control, which makes use of three orthogonal electromagnets, called magnetorquers. The magnetorquers interact with the earth's magnetic field to create control torques. Magnetic control allows a CubeSat to be stabilised into any attitude and is often used by more complex satellites as an initial detumbling controller. Active magnetic control requires at least a magnetometer for detumbling and an additional attitude sensor for three-axis attitude estimation. Coarse sun sensors, which can be as simple as reading the currents from each of the solar panels, are popular in this category of ADCSs. The disadvantage of magnetic control is that only two axes of the satellite can be controlled at any one time, depending on the satellite's position in orbit. Many CubeSats, such as CanX-1, DTU-Sat and COMPASS have been designed to achieve 10 degree pointing accuracy using active magnetic control [7].

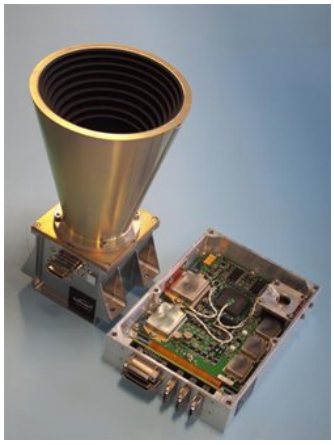
Active magnetic control can be supplemented with a momentum wheel to achieve better disturbance rejection and attitude accuracies of better than 5 degrees. The 3U, CanX-2 CubeSat has demonstrated attitude control with an accuracy of 2 degrees using this method [8].

An ADCS that can achieve an attitude accuracy in the 1-0.1 degree range has not been successfully demonstrated on a CubeSat in orbit. However, fine sun and horizon sensors have accuracies approaching 0.1 degrees and small reaction wheels have been

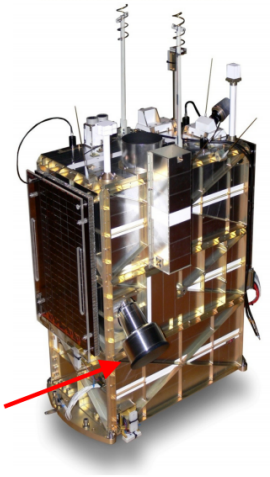
demonstrated in orbit. Therefore, a fully three-axis stabilised satellite with at least 1 degree pointing accuracy should be demonstrated in the near future. This system will only work in the sunlit part of the orbit as both the sun and horizon sensors do not work in eclipse.

Attitude accuracies better than 0.1 degrees can only be achieved using a star tracker and 3-axis reaction wheels. These kinds of accuracies are required by high gain antennas and optical instruments with long focal lengths. Even if the mission does not require such accurate pointing capabilities, accurate attitude knowledge is still important to many science instruments. No CubeSat to date has included a star tracker. However, as discussed in Section 2.5, several nano star trackers have recently come out of development. The majority of commercially and scientifically useful satellites require accurate attitude control throughout the orbit, so a star tracker is an essential component of future CubeSat ADCSs.

2.4 Star Tracker Overview



(a) SSTL Procyon star tracker [9]



(b) The position of Sumbandilasat's star tracker [10]

Figure 2.2: Modern star trackers like those from SSTL and Sunspace are too large to be used on CubeSats.

A star tracker is composed of three main components: a sensitive camera, an embedded processor and a list of known bright stars, known as a star catalogue.

The camera is used to take an image of a patch of stars. This requires a very sensitive image sensor and typical exposure times in the order of 100 ms. The satellite must

remain relatively still during the exposure time to prevent the images from smearing. Thus, star trackers are usually only employed for fine attitude adjustments after coarse sensors have been used to stabilise the satellite first.

The processor first determines whether any stars have been imaged and identifies the centroids of the imaged stars. Bright stars are easy to detect, but dimmer stars may be hard to separate from background noise. The next step is to determine which stars from the star catalogue have been imaged. Each detected centroid must be matched to a star from the star catalogue. It is very difficult or impossible to match individual stars using their brightness or colour alone, so star matching algorithms typically look at the relationships between neighbouring stars.

Once all the stars in the FOV have been matched, the star tracker has enough information to determine its attitude. Each identified star has a known, fixed position on the celestial sphere, which is also stored in the star catalogue. An algorithm compares these known locations of the identified stars to the observed locations of the stars to determine the satellite's attitude.

A star tracker has several advantages over sun and horizon sensors:

- Up to two orders of magnitude more accurate.
- Works throughout the orbit, including in eclipse.
- Outputs an attitude estimate, not a vector. No further processing of a star tracker's output is required to determine attitude.

Unfortunately, existing star trackers such as those shown in Figure 2.2 are too large, expensive and power hungry to be used on CubeSats.

2.5 Existing Nano Star Trackers

In order to determine CubeStar's desired specifications, a survey of existing nano star trackers was conducted. For the purposes of this survey, a nano star tracker was defined as a star tracker that can fit within a 1U volume and weighs less than 1 kg. Only flight qualified, commercially available star trackers were included. At the time of the survey, four suitable star trackers could be found, of which only one had flown in space.

The most important specifications of any CubeSat sensor are cost, accuracy, power consumption, volume and weight. The accuracy is normally expressed as the 3 σ error

of the sensor, therefore a lower value is better. Volume is expressed as a fraction of a U. A summary of the surveyed star trackers can be seen in Table 2.3.

Name	Acc 3 σ (°)	Pow (W)	Vol (U)	Wt (g)
Comtech MST	0.02	2.5	0.35	375
Sinclair ST-16	0.002	0.5	0.107	90
BCT Nano	0.0015[a]	0.5	0.25	500
BST ST-200	0.008	0.22	0.034	50

Table 2.3: A Comparison of Existing Nano Star Trackers

[a] 1 σ accuracy quoted. The BCT Nano datasheet is still undergoing major changes.

2.5.1 Comtech AeroAstro Miniature Star Tracker

The Comtech AeroAstro Miniature Star Tracker (MST) was developed jointly by the Massachusetts Institute of Technology (MIT) and private company AeroAstro around 2003 [11]. Comtech has since acquired AeroAstro. MST was designed to fill the gap between relatively cheap but coarse attitude sensors such as sun sensors and much more expensive and precise commercial star trackers. Unlike most commercial star trackers, MST has a relatively low sensitivity. Using its 1 megapixel CMOS Active Pixel Sensor, MST can detect stars only up to fourth magnitude, but makes up for this with a wide, 30 degree FOV. It has a claimed accuracy of 70 arc seconds (0.02 degrees) and a 1 Hz update rate. MST's mass of 425 g and size of 5.1 x 7.6 x 7.6 cm make it a feasible CubeSat sensor. However, MST requires more than 2 W of power, which is an impractically large percentage of an average CubeSat's total available power.

MST appears to be the only nano star tracker that has been flown in space. It flew as a technology demonstrator aboard NASA's FASTSAT microsatellite which was launched in 2010 [12]. Unfortunately, no information could be found regarding MST's performance on FASTSAT. The BRITE constellation consists of six 7 kg satellites, two of which contain an AeroAstro MST [13]. Two of the satellites in the constellation, UniBrite and TUGSAT, were launched in February 2013.

MST is commercially available for roughly $250 000 from Space Micro, which acquired the AeroAstro product line from Comtech in 2012 [14].

2.5.2 Sinclair Interplanetary ST-16

The Sinclair Interplanetary ST-16 (also known as S3S) star tracker was jointly developed by Ryerson University's Space Avionics and Instrumentation Lab (SAIL), the

Space Flight Laboratory of the University of Toronto Institute for Aerospace Studies and private spacecraft hardware supplier Sinclair Interplanetary [15]. It came out of development in the last two years. The ST-16 star tracker has impressive specifications, matching those of much larger star trackers. The 5 megapixel CMOS active-pixel sensor and 20 x 15 degree FOV allow for an advertised accuracy of 7 arc seconds (0.002 degrees) and an update rate of 2 Hz. Each ST-16 measures 5.9 x 5.6 x 3.25 cm and weighs only 90 g (without a baffle). On average, the ST-16 consumes only 0.5 W.

The ST-16 star tracker is interesting for several reasons. Firstly, it performs a full lost-in-space calculation on each frame, instead of transitioning into a tracking mode. Secondly, in order to mitigate radiation induced memory upsets, the ST-16 performs a full reboot after every frame. Finally, the ST-16 uses an image sensor with a rolling shutter, instead of a snapshot shutter. The significance of these design decisions are discussed in more detail in Chapters 3 and 4.

The ST-16 has no flight heritage. However, according to Sinclair Interplanetary's website 14 flight units have been delivered. Four of the six BRITE constellation satellites include ST-16 star trackers and are expected to launch before the end of 2013 [16]. It is commercially available from Sinclair Interplanetary at an order of magnitude quote of $100 000.

2.5.3 Blue Canyon Technologies Nano Star Tracker

Blue Canyon Technologies (BCT) is a relatively young American company that was founded in 2010. According to their website, its founders have a combined experience of more than 100 years in designing and building spacecraft [17].

The BCT website advertises a Nano Star Tracker, however, as of the beginning of 2013 the star tracker's datasheet is still undergoing major changes. It advertises a 7 arc second accuracy (1 σ) and 5 Hz update rate [18]. It achieves this impressive accuracy thanks to a narrow FOV of 9 x 12 degrees and a large star catalogue containing stars down to 7th magnitude. As it is a commercially developed product, very little technical information is available on this sensor. However, it does feature a novel design which integrates a baffle into the main sensor body. At 5 x 5 x 10 cm in volume, $<$0.5 kg in weight and with an average power consumption of $<$0.5 W, the BCT Nano Star Tracker is compatible with CubeSats.

The BCT Nano Star Tracker has not flown in space. However, BCT has been awarded a contract to supply star trackers for the NASA INSPIRE interplanetary CubeSat which is currently scheduled to launch in 2014 [18].

2.5.4 Berlin Space Technologies ST-200

Berlin Space Technologies have been responsible for the development of key components and the operation of several missions, including TUBSAT, Orbcomm 2nd Generation, LAPAN-A2 and LAPAN-ORARI [19]. They have recently introduced the ST-200 star tracker, which is based on the fifth generation of star tracker developed at the Technische Universität Berlin [20]. The ST-200 appears to be the smallest commercially available star tracker as of May 2013. The ST-200 advertises 30 arc second (0.008) accuracy in bore site pointing and 200 arc second (0.06 degrees) accuracy in roll, with a 1 Hz update rate. The ST-200 star tracker has a volume of 3 x 3 x 3.8 cm and a weight of 50 g without a baffle or enclosure. It has an average power consumption of 220 mW.

A unique feature of the ST-200 star tracker is its internal gyro, allowing it to report slew rates of up to 200 degrees per second even when the star tracker can no longer lock onto any stars[20].

The ST-200 star tracker (as part of the iADCS-100 package) is scheduled to fly in space for the first time onboard the Aalto-1 Finnish 3U cubesat scheduled to be launched in 2013 [21]. If this mission is successful the ST-200 will become the first commercial nano star tracker to fly in space onboard a CubeSat. The ST-200 is available for approximately $30,000.

2.6 CubeStar Project Aims

The design philosophy of CubeStar was to investigate what performance could be achieved by using only off-the-shelf components and by reusing as many subsystems as possible. This is in contrast to the usual design philosophy of setting specifications and making all design decisions in an effort to achieve those specifications. However, by comparing the specifications of the nano star trackers surveyed in Section 2.5, the abilities of other attitude sensors and the likely requirements of near-future CubeSat missions, the following broad aims were determined for the CubeStar project:

- Develop a complete engineering model of a CubeSat compatible star tracker within 2 years.
- Aim for a volume of less than 0.5 U.
- Aim for a power consumption of under 0.5 W, preferably as low as possible.

- Aim for an accuracy of at least 0.1 degrees (better than CubeSense), but preferably closer to 0.01 degrees

CubeStar will extend the capabilities of existing CubeSat attitude determination and control systems by adding the following functionality:

- More accurate attitude knowledge throughout the orbit, allowing more accurate pointing. This will enable the use of longer focal length imaging systems.
- Accurate attitude knowledge throughout eclipse.
- By tracking planets instead of stars, CubeStar could enable interplanetary navigation.

Chapter 3

Design

A star tracker is a complex instrument with many interdependent subsystems. However, there are several fundamental design decisions that need to be made early in the development process which ultimately determine the performance of the instrument. This chapter describes how these fundamental design decisions were made.

3.1 Component Interdependency

A star tracker consists of several components, or subsystems, including the optics, image sensor, electronics, mechanical structure and algorithms. The largest difficulty in designing a star tracker is the fundamental interdependency between these components. During the early stages of development, any design decision made regarding one of the components has large implications for the other components. This interdependency is depicted in Figure 3.1.

For example, choosing an image sensor has implications for the optical system, the memory requirements and the processor. The image sensor will have a certain sensitivity. The f-stop and focal length of the optics will have to be chosen to ensure that the image sensor can detect enough stars. The required memory of the star tracker will be directly proportional to the bit-depth and resolution of the image sensor. The bus-width of the whole electronic subsystem will likely be designed to match the bit-depth of the image sensor, and the load on the processor is directly proportional to the sensor resolution. Similarly, choosing any other component first will dictate the requirements of other components in the system.

To complicate matters further, every subsystem has an impact on the performance and characteristics of the star tracker. Ideally, accuracy should be maximised, while power

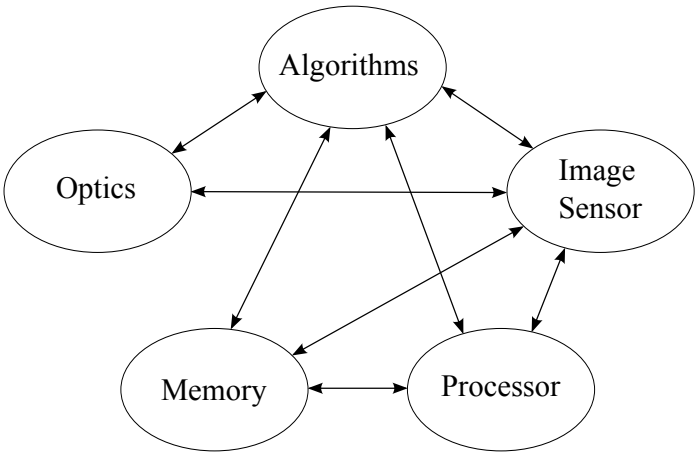


Figure 3.1: Simple depiction of the interdependency which exists between star tracker components

consumption, volume and mass are minimised. However, many of these specifications have inverse relationships due to the characteristics of the star tracker components. For example, the accuracy of the star tracker can be improved by choosing better optics or a higher resolution image sensor. Better optics will usually be larger and heavier, having an adverse effect on the star tracker's mass and volume. A higher resolution image sensor requires more memory and a faster processor, both of which will have adverse effects on the star tracker's power requirements.

Component	**Specifications**			
	Accuracy	**Power**	**Volume**	**Mass**
Algorithms				
Optics				
Image Sensor				
Processor				

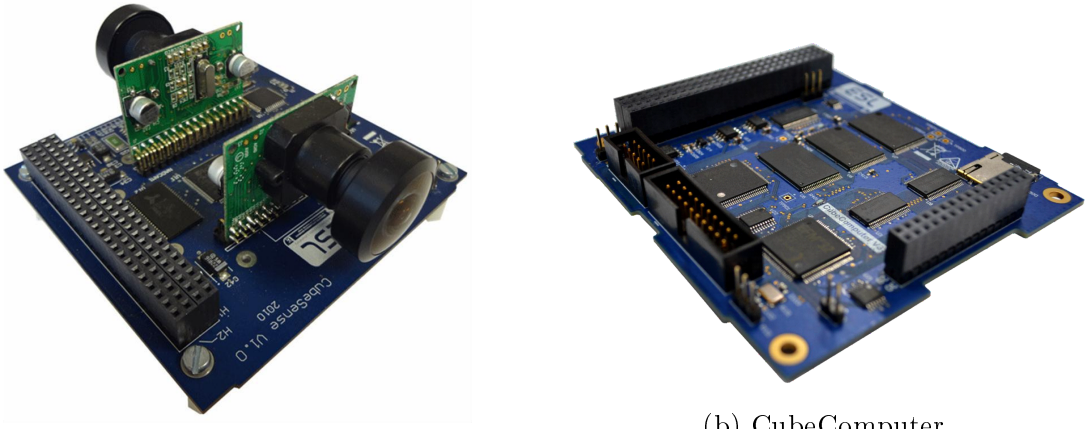
Table 3.1: The relationship between components and specifications. Green specifies a proportional relationship, while red specifies an inverse relationship.

The relationships between components and specifications are summarized in Table 3.1. The important components of the star tracker are listed along the left side of the table, and the star tracker specifications are listed along the top. The table shows what effect choosing a better component will have on the specifications, or, inversely, what quality of component will need to be chosen when a higher specification is required. A green block specifies a proportional relationship and a red block specifies an inversely proportional relationship. For example, if better quality optics are chosen, the accuracy of the star tracker will improve (proportional relationship), but the volume and mass

of the star tracker will likely increase, which is undesirable (inversely proportional). From the table it is also evident that better accuracy can only be achieved by relaxing one of the other specifications.

3.2 Hardware Heritage

The early design decisions are very important, due to the component interdependency described in Section 3.1. However, due to the time constraints of this project, it was essential to try to build on existing hardware and reuse as many subsystems as possible. Two existing ESL CubeSat components, called CubeSense and CubeComputer, were used as a starting point in the design of CubeStar.



(a) CubeSense

(b) CubeComputer

Figure 3.2: CubeStar makes use of subsystems developed for other ESL CubeSat modules, such as CubeSense and CubeComputer.

3.2.1 CubeSense

CubeSense is a combination sun and horizon sensor in PC104 form [22]. It makes use of two small CMOS APS image sensors and fish-eye lenses to track the sun and earth's horizon. The image sensors are cheap, off-the-shelf components with a resolution of 640 x 480 pixels. The cameras are interfaced to an 8-bit PIC Microchip microcontroller via a small FPGA. The microcontroller cannot handle the output data rate of the image sensor, so the FPGA, in combination with external SRAM, acts as a buffer.

The microcontroller does not contain enough RAM to store a whole image, so the FPGA saves the raw images to external SRAM. Raw images that have been saved in the external SRAM are only accessible by the FPGA. The FPGA performs some simple

image processing before sending small parts of the images to the microcontroller. In the case of the sun sensor camera, the FPGA looks for the first bright pixel above a set threshold and sends the pixels in a small window around this pixel to the microcontroller. The microcontroller then performs a centroiding algorithm on these pixels to determine the centroid of the sun. For the horizon sensor, the FPGA also performs a basic search function before sending only certain rows and columns of pixels to the microcontroller. It is possible to poll CubeSense for a complete image. In this case, the image is read out sequentially from the SRAM by the FPGA. The microcontroller transmits the bytes of the image over the UART or I2C bus as it receives them from the FPGA. The rate of data transfer between the FPGA and microcontroller is controlled by the microcontroller, and is dependent on the UART/I2C transmit data rate.

The first components of CubeSense to be investigated for possible reuse on CubeStar were the camera modules. These cameras are attractive due to their low price, good availability and ease of use (the image sensor comes pre-soldered to an interface board). The code to control and interface to these cameras has already been developed, which would speed up integration time. To test the suitability of these cameras for a star tracker, a working CubeSense board was taken outside in an attempt to image stars. The fish-eye lens was replaced with a stock lens that came with the camera module. Unfortunately, it was quickly discovered that these image sensors did not have the sensitivity required to image stars. Usually a lack of sensitivity can be compensated for by longer exposure times, but the image sensor is not capable of long enough exposures. Only one or two of the very brightest stars could be detected. Therefore, it was concluded that the CubeSense cameras were unsuitable for reuse on CubeStar.

The algorithms required by star trackers are processor intensive and require a large amount of floating point computations. The PIC18F45K20 Microcontroller on CubeSense is an 8-bit device. Performing floating point operations on an 8-bit device without floating point hardware is very inefficient and slow. It takes CubeSense approximately 200 ms to process and find the single centroid of a horizon image. CubeStar will have to find the centroids of tens of stars even before starting the processor intensive matching algorithms. The PIC microcontroller contains only 1.5 kB of RAM, which is predicted to be insufficient for the matching algorithms. It was concluded that CubeSense's microcontroller would be unsuitable for reuse on CubeStar.

While neither the cameras nor the microcontroller are suitable for CubeStar, the FPGA and external SRAM subsystems are generic enough that they can be reused with few modifications. The IGLOO NANO AGLN030 FPGA has enough pins that it can be interfaced to cameras with an output data width of 8-12 bits and has enough pins

that it can be connected to SRAM memory ICs with 512 to 2048 kB of memory. Developing VHDL code for FPGAs is time consuming and difficult to troubleshoot, so being able to reuse the FPGA and its VHDL will significantly speed up CubeStar's development process. In order to further reduce the required VHDL development, it was decided that the FPGA on CubeStar will act only as an interface between the camera, external SRAM and microcontroller. Unlike on CubeSense, on CubeStar the FPGA will not perform any image processing and will instead transfer the whole image to the processor during every iteration. All image processing will be performed by the processor, allowing a large degree of flexibility. For example, the processor could be reprogrammed while in orbit to work as a horizon sensor instead of a star tracker.

3.2.2 CubeComputer

CubeComputer is a general purpose CubeSat onboard computer developed in the ESL [23]. It consists of an ARM Cortex M3 processor, external EDAC protected SRAM, external Flash memory, a micro-SD card slot and several other peripherals in a PC104 form factor. CubeComputer's processor was investigated for possible reuse on CubeStar.

The 32-bit Energy Micro EFM32GG was chosen as CubeComputer's processor after a detailed survey of low power, high performance embedded processors [23]. Many of the reasons for choosing this processor for CubeComputer also apply to CubeStar:

- Energy Micro's processors are specifically designed for low power consumption, which is especially important for CubeSat applications.
- The 32-bit Cortex M3 core enables high performance essential for handling complex algorithms.
- The EFM32GG has an external memory bus and memory controller allowing the addition of external SRAM.

Other reasons for choosing the same processor on CubeStar include:

- CubeComputer can serve as a development board for the processor.
- Many useful libraries, such as I2C, have been developed by members of the ESL for this processor.

- All development software and hardware, such as programmers, are readily available in the lab.
- Several ESL members have extensive experience with this processor.

Before committing to the EFM32GG processor for CubeStar, it was necessary to determine whether the processor would be powerful enough to run the star detection and matching algorithms described in Section 4. As described in more detail in Section 4.3.5, early, crude versions of the detection and matching algorithm were developed in MATLAB and then ported to C. By executing the C program on a desktop computer and noting the runtime of the algorithms, a rough calculation could be done for the expected runtime on a 48 MHz processor. This test showed that the runtime would be in the right order of magnitude. A more definitive test involved programming CubeComputer with the C code porting of the algorithms. A simulated star image (see Section 4.3.5) was loaded onto CubeComputer's SD card. CubeComputer performed a complete star matching iteration on the simulated image within 1 second, proving that the EFM32GG processor was powerful enough for use on CubeStar.

Other subsystems reused from CubeComputer include the current monitoring and power switching circuits. These circuits monitor the current consumption of the external SRAM modules. If a large spike in current is detected, the power to the SRAM modules is cycled. Large increases in current consumption are a sign of radiation induced latch-ups, which can destroy the SRAM if the power is not cycled immediately.

3.3 Data Flow

The flow of data from the image sensor to the processor is described in this section. There are various different ways that the camera, FPGA, SRAM and processor can be connected together, as shown in Figures 3.3-3.5. Each arrangement has advantages and disadvantages.

Option 1 borrows the most from the existing hardware designs of CubeSense and CubeComputer. The FPGA and processor each have their own external SRAMs, which only they can access. The FPGA and the processor are connected using only eight parallel data lines and a few control lines. Thus, the processor can only access the data stored in the FPGA's SRAM sequentially. The processor uses its External Bus Interface (EBI) to connect to its external SRAM. The data flow works as follows:

1. An image is captured and stored in the FPGA's external SRAM

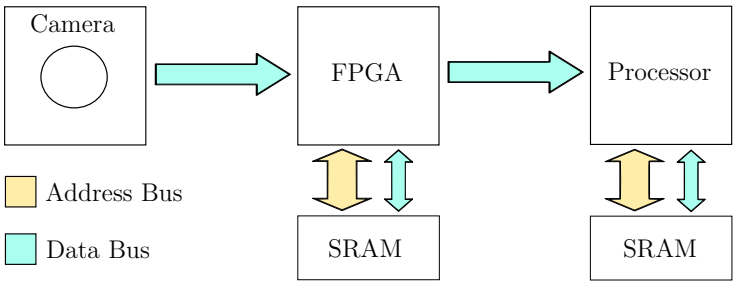


Figure 3.3: Option 1. Component interconnections and data flow

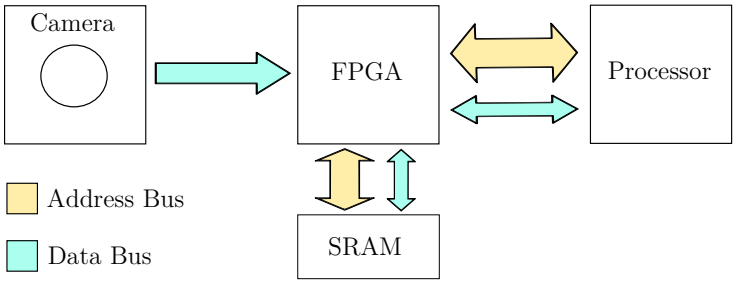


Figure 3.4: Option 2. Component interconnections and data flow

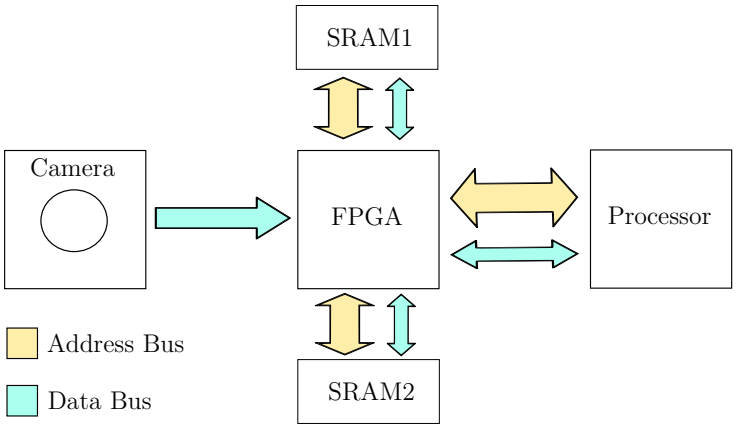


Figure 3.5: Option 3. Component interconnections and data flow

2. The image is transferred from the FPGA's external SRAM to the processor's external SRAM
3. The processor operates on the image now in its external SRAM, while step one repeats in parallel

Option 2 uses the least hardware. Instead of having two separate SRAMs, both the FPGA and processor have read and write access to a single shared SRAM. Both the processor's external SRAM and the FPGA are connected to the processor's EBI. When the FPGA has access to the SRAM, it will control the SRAM's address and data buses. When the processor wants to access the SRAM, the FPGA will effectively connect the processor's address and data buses to the SRAM's address and data buses. The FPGA will act as a data flow switch. The data flow works as follows:

1. An image is captured and stored in the shared external SRAM
2. The processor operates on the image in the shared external SRAM

Option 3 is the most efficient. It uses two, separate, external SRAMs connected to the FPGA. Both the FPGA and processor have read and write access to both SRAMs. As in Option 2, the FPGA acts as a data flow switch and determines who has control of the SRAM's data and address buses. This layout gives redundancy in the event that one of the SRAMs fails. The data flow works as follows:

1. An image is captured and stored in SRAM1
2. The processor operates on the image in SRAM1, while a new image is being captured to SRAM2
3. The processor operates on the image in SRAM2, while step one repeats in parallel.

Option 1 will be the easiest to implement, as it makes the most use of existing designs. It also requires the fewest connections between components, allowing the hardware to be developed in three separate modules. The interface between the FPGA and processor is simple, so upgrading or changing to a different processor would be easy. The image transfer process between the FPGA's external SRAM and the processor's external SRAM is inefficient and will occupy a large portion of the processor's time. Fortunately, the dual SRAM's allow the image capture and image processing to be done in parallel.

Option 2 requires only one external SRAM module, saving space and reducing power consumption. No image transfer is required, but the image capture and image processing will have to be performed sequentially. This will result in the same update rate as Option 1, but the timing will be simpler to implement. The main disadvantage of Option 2 is the interface between the processor and the FPGA, which consists of both an address bus and a data bus. This interface requires a large number of lines and will require significant modifications to the FPGA code. If used with the Gecko processor's EBI, this will be the most elegant solution.

Option 3 has redundancy against a failed SRAM module and can achieve twice the update rate of the other two options. It requires the same processor-FPGA interface as Option 2. The main disadvantage of Option 3 is the large number of FPGA I/O pins required. The IGLOO NANO AGLN030, which is the first choice for CubeStar as it is used on both CubeSense and CubeComputer, has too few pins to implement Option 3.

A short development time necessitates the mitigation of risks, which ultimately led to the adoption of Option 1. CubeStar is the first version of an ESL nano star tracker, so there are many unknowns in the development process. By choosing Option 1, the hardware design is based as closely as possible on existing designs. The aim of this first version of CubeStar is to get a complete working system. Thereafter, improvements can be made to various subsystems, as more is learnt about the effectiveness of each subsystem.

3.4 Image Sensor

The image sensor is arguably the most important component of a star tracker. The choice of image sensor has large implications for the rest of the system components and ultimately determines the star tracker's performance.

3.4.1 CCD vs CMOS APS

Charge coupled devices (CCD) and complementary metal-oxide-semiconductor (CMOS) active pixel sensors (APS) are two image sensor technologies. CCDs were invented in the early seventies. In essence, a CCD consists of a two dimensional (2D) array of capacitors which can be charged by incoming photons. After a set exposure time a control circuit causes each capacitor to transfer its charge to its neighbour in the row, causing each row to operate as a shift register. The final capacitor in each row trans-

fers its charge to a charge amplifier which converts the charge to an analogue output voltage. CCDs are very efficient at turning photons into light, thus making them very sensitive.

Unlike CCDs which have only one amplifier per row, CMOS Active Pixel Sensors contain an amplifier at each pixel. Image sensors with in-pixel amplifiers were actually invented before the CCD. However, it was not until the nineties, when the CMOS process had become well established, that Active Pixel Sensors became practical. CMOS Active Pixel Sensors are easier to manufacture and allow the control and image processing circuitry to be fabricated on the same IC as the imaging array. Various pixel architectures exist. The most common one, called the Noble 3T Pixel, consists of three transistors, one of which acts as the light sensitive element. The unfortunate consequence of adding an amplifier to each pixel is that the light sensitive area of each pixel is reduced. Early CMOS Active pixel Sensors were noisy and insensitive compared to CCDs. However, modern CMOS Active Pixel Sensors have reached the level where they are replacing CCDs in almost all applications, except in very high end medical and scientific equipment.

Advantages of CCDs

- Higher sensitivity
- Global Shutter prevents image smear when imaging moving objects

Advantages of CMOS APS

- Much lower power consumption
- Typically cheaper
- Control and image processing circuitry usually contained on the IC, meaning less external circuitry is required
- Immune to blooming, where overloaded pixels bleed into neighbouring pixels

CMOS Active Pixel Sensors clearly offer many important advantages over CCDs. Power consumption is vitally important for a CubeSat sensor such as CubeStar. The simpler external circuitry required by a CMOS APS compared to a CCD will speed up development time and will save space. CubeSat sensors are expected to be significantly cheaper than regular satellite sensors, so the cost of the image sensor is important.

Immunity to blooming may be advantageous when the moon enters the star tracker FOV.

CMOS Active Pixel Sensors with a global shutter exist, but they are rare. Most CMOS APSs have a rolling shutter. A rolling shutter means that the image sensor rows are exposed sequentially, instead of all at once. This causes image distortion when capturing moving objects as the top and bottom of the sensor are imaging different moments in time. Fortunately, there are several reasons why a rolling shutter should not be a problem for CubeStar. Firstly, CubeStar is not expected to work at angular velocities greater than a few tenths of a degree per second. Secondly, CubeStar will have a relatively wide FOV, causing objects to move slowly across its FOV. Finally, other star trackers, such as the Sinclair Interplanetary S3S, have been designed around rolling shutter sensors, proving that a global shutter is not required [15].

On average, Active Pixel Sensors are still less sensitive than CCDs. However, as the calculations in Section 3.4.4 show, modern CMOS APSs are sensitive enough for star tracker applications.

In conclusion, a CMOS Active Pixel Sensor, instead of a CCD, will be used on CubeStar.

3.4.2 Desired Specifications

In order to achieve the best possible performance while being compatible with CubeSats, the following image sensor specifications need to be maximised:

- Sensitivity
- Availability
- Ease of interface

While the following specifications are minimised:

- Physical Size
- Cost
- Power Consumption

High sensitivity is vital as stars are very faint sources of light. Being able to detect fainter stars allows more stars to be used during each attitude estimate, leading to more

accurate estimates. An image sensor's sensitivity is a function of its quantum efficiency, fill factor and noise sources. Every manufacturer quotes different specifications and uses different units, making sensitivity calculations difficult. However, pixel size is a good approximation for sensitivity. Larger pixels can catch more photons, resulting in more sensitivity. Based on a survey of image sensors used in commercial star trackers, a minimum pixel size of 5 x 5 μm is desirable. However, pixel size is not the only factor influencing sensitivity, so a more comprehensive calculation is performed in Section 3.4.4.

The availability of the chosen image sensor should be high. Old, obsolete or image sensors which are only available in bulk should be avoided.

Since the FPGA and external SRAM subsystems from CubeSense will be reused for CubeStar, it is desirable that the chosen image sensor's interface is similar to the interface of the CubeStar cameras. Therefore, the chosen image sensor will preferably have an I2C control interface and parallel data output. Analogue data outputs are undesirable as they require additional analogue to digital converters (ADCs), which increase complexity.

The physical size and weight of the image sensor should be minimised to be compatible with CubeSats. Image sensors with larger pixels tend to be more sensitive, but are also larger and require larger lenses. Common image sensor sizes include 1/4", 1/3", 2/3" and 1". 1/3" seems to be the smallest sensor size that can accommodate 5 μm pixels at a reasonable resolution. A large number of small, commercial-off-the-shelf (COTS) lenses are available for 1/3" format sensors.

Image sensors vary widely in price, costing from tens of dollars to thousands of dollars per sensor. The most expensive sensors are radiation hardened especially for space applications. These include sensors manufactured by FillFactory (now acquired by Cypress Semiconductor) and the STAR range of sensors from ON Semiconductor. However, these image sensors are prohibitively expensive for a CubeSat mission. The cheapest commercial image sensors are used in low-end cellphones and gadgets. These image sensors have very small pixels ($<$2 μm) resulting in noisy images in low light conditions. Fortunately, a large selection of mid-range commercial and industrial image sensors are available for under \$300.

The image sensor is expected to contribute a significant amount to the star tracker's power consumption. The power consumption of an image sensor increases significantly while imaging. Unfortunately, in a star tracker application, the image sensor will likely be imaging most of the time due to the long exposure times required to detect stars.

Since CubeStar is required to use less than 500 mW, the image sensor should consume significantly less than this.

The resolution of the image sensor is not mentioned in either the maximise or minimise list. Maximising the resolution will maximise the star tracker's accuracy, but it will also maximise its power consumption due to the large amounts of memory and computation required to process the images. Therefore, a compromise for the resolution must be found. Very few modern star trackers have a resolution of less than 512 x 512 pixels. A CubeSat compatible star tracker will have limited computational power, so the lowest resolution sensor that can give acceptable results is desirable. Therefore, a resolution of 512 x 512 pixels is preferred.

Make	**Model**	**Resolution**	**Pixel Size (μm)**	**Power (mW)**	**Price ($)**
ZMD	ZMD33220	840 x 640	10.6	<300	-
Melexis	MLX75412	1024 x 512	5.6	250	41
NIT	NSC1104	768 x 576	15	230	255; 1550*
ST	VL5510	1024 x 512	5.6	<150	20
E^2V	EV76C560	1280 x 1024	5.3	200	-
E^2V	EV76C454	860 x 640	5.8	80	2000*

Table 3.2: A summary of the image sensors considered for CubeStar. * denotes the price for an evaluation kit.

Table 3.2 lists the image sensors that were initially considered for CubeStar.

The ZMD33220 image sensor has the second largest pixel size in the list and a global shutter, making it very desirable. It was designed for automotive applications, allowing it to operate over a large range of temperatures. Its power consumption is the highest in the group, but still within budget. Its 2/3" form factor is larger than desired, but not too large. Unfortunately, during attempts to get a quote, it was discovered that the ZMD33220 has been discontinued and is no longer available.

The MLX75412 and VL5510 appear to be very similar, although they are manufactured by different companies. Both are designed for low light automotive applications, such as night vision assist cameras. Their rectangular resolution is not ideal for a star tracker, as a lot of the incoming light from the lens will fall outside the image sensor. However, both sensors have I2C command interfaces and parallel data outputs, allowing them to be interfaced easily. Both sensors are the ideal 1/3" format and have several useful on-chip image processing functions. The VL5510 is cheaper but is only available in bulk, while individual MLX75412 sensors are available.

New Imaging Technologies (NIT) makes a variety of high quality CMOS APS image sensors for industrial applications. The NSC1104 has very large pixels, giving it good sensitivity. Unfortunately, while the NSC1104 would likely perform the best out of all the sensors in the list, it has major drawbacks. These drawbacks include its large size (1" format), analogue outputs (requiring external ADCs) and high price.

E^2V also makes high quality image sensors. E^2V has extensive experience developing imaging sensors for space applications. Several current satellites and space probes, including Pleiades and New Horizons, contain E^2V image sensors [24]. The EV76C454 (JADE) and EV76C560 (SAPPHIRE) image sensors are very similar and differ mainly in resolution. Both have a 1/3" format and have global shutters. The JADE sensor has the lowest power consumption in the list. These would have been the first choice for CubeStar. Unfortunately, the sensors need to be bought directly from E^2V. E^2V would prefer to sell very expensive development kits than individual image sensors. However, the JADE and SAPPHIRE sensors should be reconsidered if a second generation CubeStar is ever developed.

After comparing each sensor to the required specifications, the MLX75412 image sensor was chosen for CubeStar.

3.4.3 Melexis MLX75412

The Melexis MLX75412 is the newest sensor in Melexis's long range of automotive image sensors. The automotive industry may seem like an unlikely place to look for components that will fly in space. However, it is important to remember that CubeSats try to make use of commercial, off-the-shelf components, instead of expensive space rated parts. Since these commercial components are not designed to operate in the harsh environment of space, it is always a risk to include them. However, this risk can be minimised by choosing components with a wide operating range. Since the Melexis MLX75412 was designed for the automotive industry, it can operate in a wide temperature range (-40 to +115 degrees Celsius) and should be more robust than other off-the-shelf components.

The MLX75412 is available in both a monochrome and a colour RGB version. Star matching algorithms do not require colour information, so the monochrome version, which is also slightly more sensitive, is chosen.

Table 3.3 summarises the specifications of the MLX75412.

Unfortunately the MLX75412's power consumption is quite high. Hopefully the power consumption can be brought down by running the image sensor at a slow framerate.

Specification	Value	Units
Optical Specifications		
Resolution	1024 x 512	pixels
Pixel Size	5.6 x 5.6	μm^2
Optical Format	1/3	inches
Sensitivity (SNR10)	25	nW/cm^2 @ 535nm
Dark Current Leakage	1008	DN12/s @ 65°C
Electrical Specifications		
Supply Voltages	1.8 and 3.3	V
Power Consumption	250	mW @ 35fps
Interface Specifications		
Control	I2C slave	
Data	8-12 bit parallel	

Table 3.3: Specifications of the Melexis MLX75412 image sensor.

The Melexis MLX75412 is available for approximately $41 in single quantities from online distributor Future Electronics.

3.4.4 Sensitivity Analysis

The Melexis datasheet mentions two useful optical characteristics of the sensor: the sensitivity and the dark current leakage. The sensitivity figure states that 25 nW/cm^2 of radiant energy is required for a signal to noise ratio (SNR) of 10. A signal to noise ratio of 10 means that the output signal is ten times larger than the noise. The quoted sensitivity incorporates all the sensor noise sources, making it a single figure of merit. SNR10 seems to be a common benchmark for comparing image sensors. A paper on the design of *DayStar* [25], another star tracker, notes that a signal to noise ratio of 6 is sufficient for detecting stars, so SNR10 is a conservative benchmark.

In order to calculate whether the Melexis sensor will be sensitive enough to detect stars, the quoted sensitivity value must be converted from Watts to photons per second. This is achieved with Equation 3.4.1, which relates the rate of photons at a particular wavelength to power.

$$N_{ph} = \frac{P\lambda}{hc} \tag{3.4.1}$$

where

N_{ph} = equivalent number of photons per second
P = power of the electromagnetic radiation in Watts
λ = the wavelength of the light in m
h = Planck's constant: $6.63 \cdot 10^{-34}$
c = speed of light: $3 \cdot 10^{8}$ m/s

Stars do not emit light of a single wavelength, but emit a whole range of wavelengths. However, to simplify the calculation it is assumed that all the starlight is confined to a single wavelength of 535 nm, which is the middle of the visible spectrum. Using this approximation, Equation 3.4.1 shows that 25 nW of electromagnetic radiation at 535 nm is equivalent to $6.733 \cdot 10^{10}$ photons per second.

Therefore, $6.733 \cdot 10^{10}$ photons per second per cm^2 are required to get a signal to noise ratio of 10. That equates to 673.3 photons per second per μm^2 and $21.115 \cdot 10^3$ photons per second per pixel, since each pixel has a size of 5.6 x 5.6 μm^2.

Next, the number of starlight photons captured by the optics is calculated. This is achieved using equation 3.4.2, which is reproduced from a paper titled: *Study on the Detection Sensitivity of APS Star Tracker* [26]. By setting T equal to 1 second, Equation 3.4.2 gives the number of photons per second which hit the image plane.

$$N_{php} = A_l \cdot T_l \cdot \Delta B \cdot \phi_m \cdot T \tag{3.4.2}$$

where

N_{php} = number of photons per second hitting the image plane
A_l = the lens aperture in cm^2
T_l = the permeance rate of the optics. Usually between 0.6 and 0.8
ΔB = the bandwidth of the lens. Usually between 3000 and 6000 Angstrom
$\phi_m = 10^{(15-2M)/5}$ luminous flux of M visual magnitude
T = exposure time

For the order of magnitude equation the lens radius was chosen as 1 cm, the permeance was chosen as 0.6 and the bandwidth was chosen as 3000 Angstrom. If the optics were perfectly focussed, the resultant photons per second calculated in equation 3.4.2

would all fall on a single pixel. However, as explained in Section 4.1.1, the optics are purposely defocussed and the starlight spans several pixels, often over a 3 x 3 or 5 x 5 grid. Therefore the resultant photons per second per pixel can be calculated by dividing the total photons per second by the number of pixels that are covered.

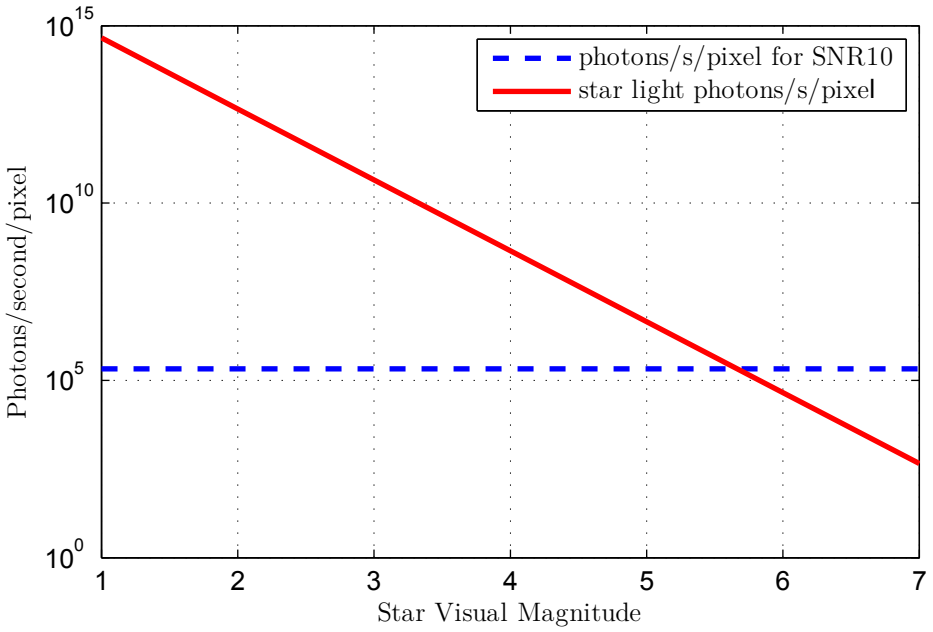


Figure 3.6: Starlight photons per pixel per second vs required photons per pixel per second to achieve SNR10

Figure 3.6 plots the required number of photons per second per pixel to achieve SNR10, against the number of starlight photons per second per pixel for stars of varying magnitudes. The plot shows that stars down to 5.5 magnitude should be detectable against background sensor noise.

To calculate the maximum possible exposure time, it is necessary to look at the second optical characteristic given in the datasheet: dark current leakage. The datasheet states a dark current leakage of 1008 ADC units per second at 65 degrees Celsius. This means that the image sensor will output a value of 1008 (out of 2^{12} for 12-bit mode) for each pixel if a 1 second exposure is taken in a perfectly dark environment. An ideal image sensor would output zero, but dark current leakage causes a non zero value to be output. Therefore, exposures longer than $2^{12}/1008 = 4.06$ seconds will saturate the sensor.

The exposures used on CubeStar will never exceed 500ms to ensure that the algorithms have enough time to execute. Therefore, a maximum exposure time of 4 seconds is

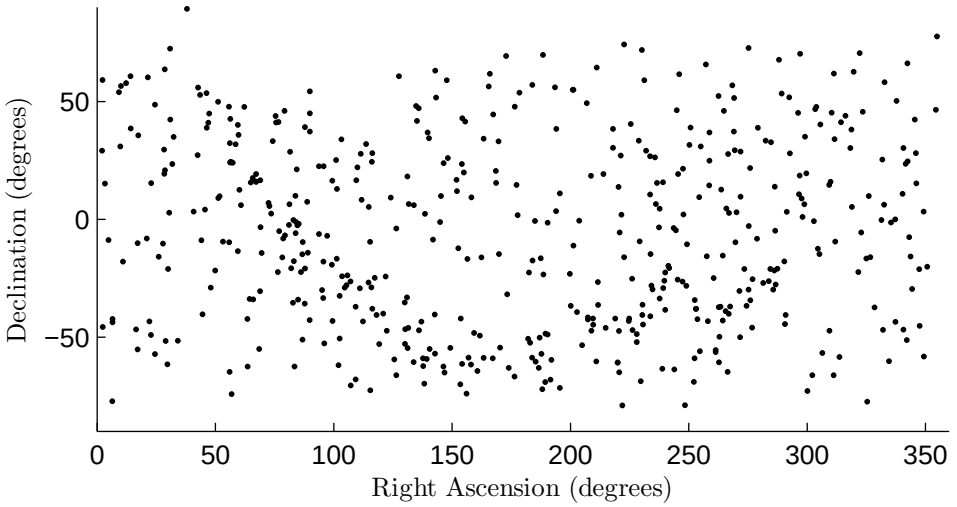


Figure 3.7: The distribution of all stars up to 4th magnitude over the celestial sphere

acceptable. Most modern APS CMOS sensors, including the Melexis, have automatic dark current correction. At least one row of the image sensor is not exposed to light during exposures. The average value of these pixels can then be subtracted from all the other pixels to remove the effects of dark current leakage. In this way the sensor can still output zero when imaging a perfectly dark environment.

3.5 Field of View

The FOV of a star tracker needs to be chosen such that at least three detectable stars are always in the FOV. Three is the minimum amount of stars required to determine three-axis attitude. This section describes the process of choosing a lens which fulfils this requirement.

3.5.1 Star Distribution

Stars are not evenly distributed over the celestial sphere. Certain areas of the night sky contain many stars, while other areas are relatively spars. This is a result of the shape of our galaxy. Our galaxy is a spiral galaxy, which is a flattened, spinning disk of stars. If we look up and out of our galaxy, we see few stars, but if we look through our galaxy, along one of its long axes, we see many stars. The stripe of densely populated night sky commonly referred to as the Milky Way is our galaxy seen side on. Figure 3.7 shows the distribution of all the stars down to 4th magnitude. This is approximately equal to all the stars that are visible to the naked eye in the suburbs. The figure clearly shows the uneven distribution of the stars.

3.5.2 Field of View Simulations

CubeStar always needs at least three detectable stars in its FOV, even while pointed at the spars parts of the celestial sphere. To determine the minimum FOV required for different limiting magnitudes, a simulation was run in MATLAB. Figure 3.8 was created using Monte Carlo simulations. For each data point, 20 000 simulated star images from various, random parts of the sky were generated. Each set of 20 000 images was generated using a specific circular FOV and limiting magnitude. Then the images were investigated to determine what fraction contained at least three stars, giving the sky coverage. A sky coverage of 1, or 100%, is desirable, as this means that the star tracker will work over the whole celestial sphere. The majority of commercial star trackers have sky coverages of around 99.9%.

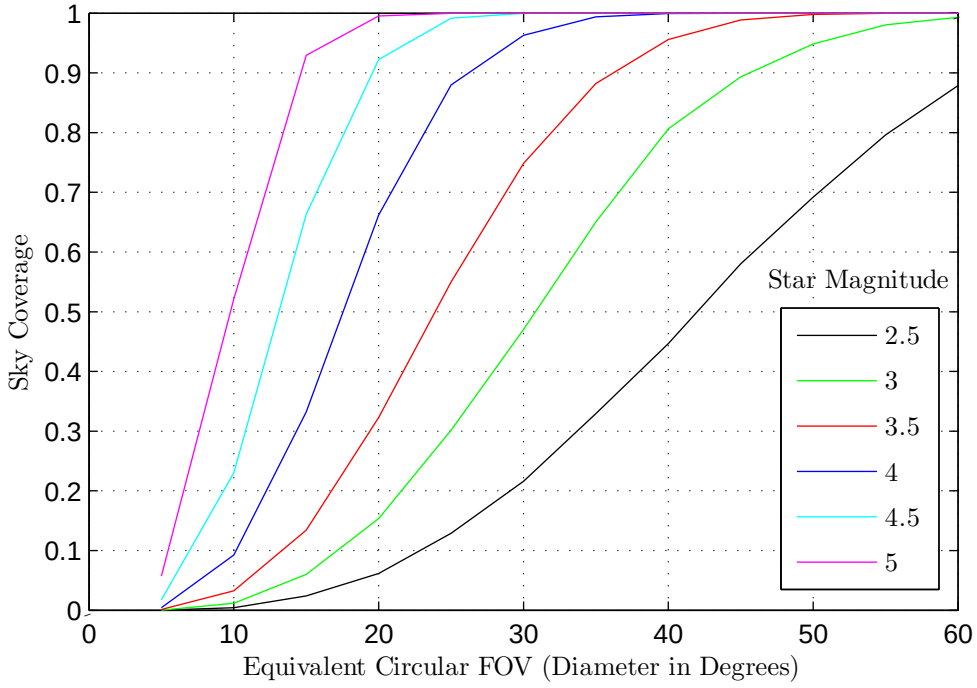


Figure 3.8: Sky coverage for various combinations of FOV and limiting magnitude.

The most common star tracker design has an equivalent circular FOV of approximately 20 degrees and a limiting magnitude of about five. From Figure 3.8 it can be seen that this combination results in a sky coverage approaching 1, or 100%. Before CubeStar's required FOV could be determined from Figure 3.8, more information about the Melexis image sensor's sensitivity was required.

3.5.3 Initial Sensitivity Test

The required FOV is largely dependent on the sensitivity of the image sensor. While the calculations in Section 3.4.4 prove that the image sensor is sensitive enough for star tracker applications, they do not indicate what magnitude of stars will be visible with a specific exposure time. In order to get more data on the sensitivity of the Melexis image sensor, a breakout board was developed to interface the Melexis image sensor to CubeSense. The breakout board, described in detail in Section 5.2.1, gives the Melexis sensor the same interface as the CubeSense cameras. By replacing one of the CubeSense cameras, images can be acquired with the Melexis image sensor.

The Melexis image sensor, interfaced to CubeSense, was fitted with a cheap s-mount lens and taken outside for a night sky test. The aim of the test was to determine the magnitude of the dimmest stars that would be visible with a 500ms exposure. After taking images of various parts of the night sky, it was concluded that stars down to magnitude 3.5 were detectable. This is called the limiting magnitude. Dimmer stars should be detectable with better optics. With a limiting magnitude of 3.5, Figure 3.8 shows that an equivalent circular FOV of approximately 50 degrees is required for 100% sky coverage.

Fifty degrees is a very wide FOV for a star tracker. A wide FOV allows an image sensor with a lower sensitivity to be used, but it also has a downside. The primary concern with a wide FOV is objects other than stars entering the FOV. If the sun, moon or the earth comes into the FOV, the image sensor will get saturated and no stars will be detectable. The larger the FOV, the more often these objects will enter the FOV and disable the star tracker. Therefore, it is desirable to keep the FOV as small as possible. Since the sensitivity of the image sensor cannot be changed and the exposure cannot be lengthened past 500 ms, the only option for reducing the FOV is to use a lens with a lower f-number.

3.5.4 f-Number

The f-number is a measure of the light gathering ability of a lens. It is related to the lens's focal length and aperture, as given in Equation 3.5.1. The amount of light that reaches the image sensor through the lens is related to the square of the lens's f-number. Larger f-numbers result in darker images, therefore a lens with the smallest available f-number is desired for CubeStar. The f-number can be decreased by increasing the aperture, or by decreasing the focal length. A standard scale of f-numbers exists where each increasing number in the sequence represents a halving of the light reaching the

image sensor. This scale is given in Table 3.4 It is common for lenses to quote their f-number proceeded by $f/$, which forms a mathematical expression for the lens's aperture diameter.

$$f_{number} = \frac{focal\ length}{aperture} \tag{3.5.1}$$

Standard f-number scale					
f-number	0.7	1.0	1.4	2.0	2.8

Table 3.4: Standard scale of f-numbers where each increasing element represents a halving of the light reaching the image sensor.

There is a convenient relationship between star magnitude and lens f-numbers. Stars which differ by one in stellar magnitude differ by approximately 2.5 times in brightness. Similarly, lenses with consecutive f-numbers (from the standard scale) produce images that differ by two times in brightness. Therefore, it is a good initial estimate to say that a lens with an f-number one lower on the scale will be able to image stars of one stellar magnitude dimmer.

3.5.5 Lens Selection

A suitable lens for CubeStar must be found commercially, as designing a custom lens is beyond the capabilities and budget of the ESL. The lens must have an equivalent circular FOV of 50 degrees and an f-number as low as possible. For weight and size reasons, only s-mount, 1/3" lenses were considered.

The FOV is determined by the focal length of the lens and the size of the image sensor. Since the Melexis image sensor has a rectangular resolution, its horizontal and vertical FOVS will differ. However, an equivalent circular FOV can be determined by multiplying the horizontal and vertical FOVs together to get an area and calculating what diameter circle will give an equivalent area. The size of the light sensitive area of the Melexis image sensor is 5.8016 x 2.9120 mm. The FOV of a lens with a focal length f is given by Equations 3.5.2-3.5.4.

$$FOV_h = atan(\frac{I_x}{f}) \times 2 \tag{3.5.2}$$

$$FOV_v = atan(\frac{I_y}{f}) \times 2 \tag{3.5.3}$$

(a) Marshall 5.7 mm lens

(b) Lensation 6 mm lens

Figure 3.9: Two commercial, s-mount lenses that were considered for CubeStar

$$FOV_{circular} = \sqrt{\frac{FOV_h FOV_v}{\pi}} \times 2 \tag{3.5.4}$$

where

FOV_h, FOV_v = the horizontal and vertical FOVs

$FOV_{circular}$ = the diameter of an equivalent circular FOV

I_x, I_y = the two dimensions of the light sensitive area of the image sensor in mm

f = lens focal length in mm

Two suitable commercial lenses were found: one from Marshall Electronics [27] and the other from Lensation [28]. The specifications of these lenses are shown in Table 3.9. Both lenses have FOVs which are slightly smaller than 50 degrees. However, this was deemed acceptable as both lenses are better quality than the cheap lens used during the initial sensitivity test (see Section 3.5.3). The Lensagon lens is specifically designed with a very low f-number for low light applications.

The lower f-number of the Lensation lens allows it to image stars of almost half a magnitude fainter than the Marshall lens. This performance is worth its slightly larger weight, size and cost. Unfortunately, the Lensation lens is not available locally, but it can be imported from Germany for a reasonable price. The Lensation lens will be used for the CubeStar project.

Table 3.6 gives detail on the expected sky coverage of CubeStar when using the Lensation lens. Detecting magnitude 3.6 stars will ensure that at least three stars are

Specification	Marshall Lens	Lensation lens
Model	V-4305.7-1.3-A	BL6012
Image Format	1/3"	1/3"
Mount Type	s-mount (M12x0.5)	s-mount (M12 x 0.5)
Focal Length	5.7 mm	6 mm
FOV with Melexis sensor	54 x 28.6°	51.6 x 27.28°
Equivalent circular FOV	44.34°	42.33°
Aperture	$f/1.4$	$f/1.2$
Weight	5 g	32.5 g
Price	ZAR 310	ZAR 487 (38 Euros)

Table 3.5: COTS Lenses considered for CubeStar

within the FOV 99.15% of the time. The initial sensitivity test proves that this will be achievable. However, the increased performance of the Lensation lens is expected to allow the detection of even fainter stars, perhaps even down to 4th magnitude. The star matching and attitude determination algorithms (Chapter 4) will be more robust and more accurate if more stars are in the FOV.

Limiting Mag	% images with ≥ 2 stars	% images with ≥ 5 stars	Average Stars
3.0	84.04	52.61	5.71
3.2	90.51	68.81	7.04
3.4	93.71	82.52	8.55
3.6	99.15	91.92	10.61
3.8	99.92	98.92	13.52
4.0	100	99.61	17.10

Table 3.6: Star count vs. cutoff magnitude determined by generating 20000 simulated star images for each cutoff magnitude (42° radius circular FOV)

Chapter 4

Algorithms

This chapter describes the algorithms which process each star image in order to determine and output the estimated bore-sight pointing vector of the sensor. These algorithms distinguish a star tracker from a simple camera.

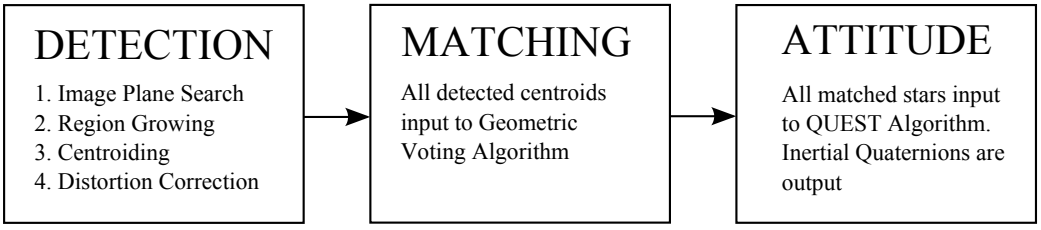


Figure 4.1: The three stages required to go from star image to attitude estimate

Figure 4.1 shows an overview of the process required to go from raw star image to attitude estimate. Most star trackers, including CubeStar, split the process into three steps, as shown in the figure. Detecting stars is described in Section 4.1. Star matching is described in Sections 4.2 - 4.5 and attitude determination is described in Section 4.4.

4.1 Detecting Stars

Before any star matching can occur, all stars present in the image must be detected and extracted from the background noise. This is a computationally expensive process as each image contains thousands of pixels that need to be examined. It is thus crucial that the algorithms used during this process be as efficient as possible. The star detection and extraction process is split into three steps: image plane search, region growing and centroiding.

4.1.1 Image Plane Search

A star is, for all practical purposes, a point source of light. If a star is imaged with perfectly focussed optics, the star will appear as a single pixel on the image plane (actually up to four pixels if the star falls exactly between pixels). Since all the light from the star will be focussed on a single pixel, the chance of detecting the star above the background noise will be maximised. However, there are two problems with this approach.

Firstly, it will be impossible to distinguish the star from a dead pixel. A dead pixel is a pixel which no longer responds to light (or has a very reduced response to light). The pixel may get stuck in the *off* position, or worse, in the *on* position. Dead pixels are a real concern for imaging hardware in space as radiation damage often manifests itself as dead pixels.

The second problem when using perfectly focussed optics is more counter intuitive. The centroid of a defocussed star image can be more accurately determined than that of a perfectly focussed star. The worst case scenario for perfectly focussed optics is if the star falls perfectly on a single pixel. In this case the centroid of the star can only be located to pixel accuracy. Therefore, the accuracy of the star tracker will be determined solely by the resolution of the image sensor. However, if the optics are slightly defocussed, the starlight will be spread over several pixels, allowing centroiding algorithms to determine the centre of the star to sub-pixel accuracy. Typically, the starlight is defocussed such that the dimmest detectable star is spread over an area of 3x3 pixels.

For these two reasons most star trackers use slightly defocussed optics. The theory behind how the starlight spreads due to the defocussed optics is complex. A detailed discussion can be found in B. C. Greyling's Master's Thesis [29]. For the purposes of this thesis it is enough to know that the distribution of the starlight can be approximated as a 2D Gaussian function.

Each raw image contains 1024 x 512 = 524288, or more than half a million pixels. To find stars, these pixels need to be checked against a threshold. Ideally, the background of a star image should be perfectly black, allowing all pixels above the black value to be detected as stars. Unfortunately, the background of real star images is never perfectly black. This is due to noise sources in the electronics, and light pollution from city lights while testing on the ground, or light sources such as the moon or the earth's albedo while in space. Thus, a threshold needs to be chosen which will be used to differentiate between the background and stars. Choosing a threshold is highly dependent on the

performance of the hardware, so it is discussed in Section 5.

The fact that a star does not appear as a single pixel, but rather as an area of pixels, is used to speed up the search for stars in the raw image. Starting from the first pixel in the top left corner of the raw image, only every third pixel in the row is checked against the threshold. Since every detectable star will have a diameter of at least 3 pixels, no stars will be missed. When the end of the row is reached, the algorithm skips two rows and starts again on the fourth row of the image. In this way only 1/9th of all the pixels need to be checked. This results in almost an order of magnitude increase in speed of the search procedure.

Whenever a pixel above the threshold is detected, the Region Growing Algorithm is called to find all pixels belonging to the detected star. When the Region Growing Algorithm completes, the search over the image plane continues.

4.1.2 Region Growing Algorithm

The Region Growing Algorithm is the same as used onboard SUNSAT's star tracker [30]. It is responsible for finding all pixels belonging to a single star by growing out from a single pixel of that star. It is a recursive algorithm which works as follows:

1. The algorithm is given the location of a single pixel, called the seed, belonging to a star.
2. The seed is added to a list of pixels belonging to the star.
3. The seed's value is set to zero to prevent it from being detected again.
4. The seed's four neighbouring pixels are checked against the detection threshold.
5. If a neighbouring pixel is above the threshold, the algorithm is called recursively with that pixel as seed.
6. The algorithm completes when no more neighbouring pixels above the threshold can be found.

The image plane search procedure and Region Growing Algorithm are described graphically in Figure 4.2

The Region Growing Algorithm returns a list of pixels belonging to the same star as the seed. Two checks are performed on this list to determine whether it represents a

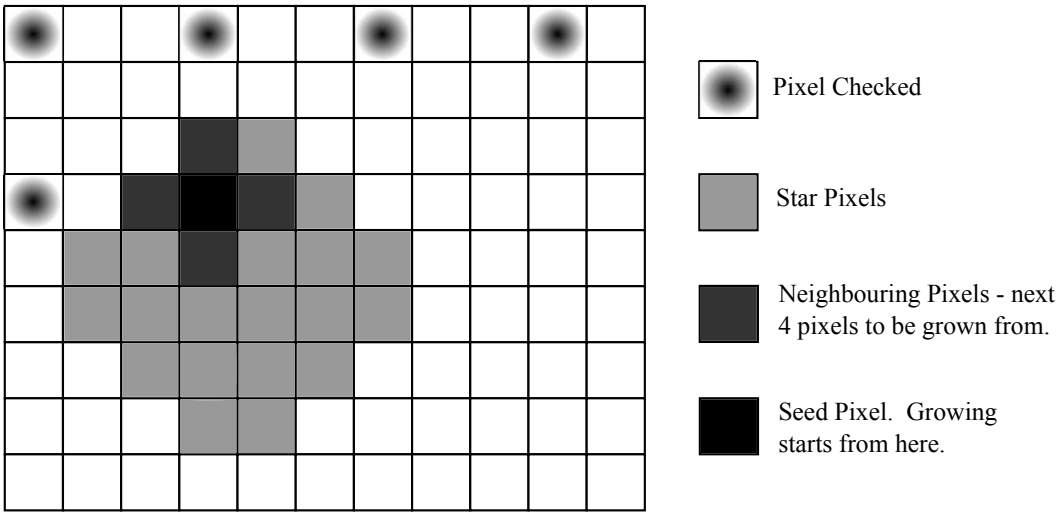


Figure 4.2: A graphical representation of the Image Plane Search and Region Growing algorithms used to detect and extract stars in raw star images

star and not a dead pixel or other image artefact. The list must contain a minimum number of pixels. Too few pixels indicate a dead pixel or star which is too faint to detect reliably. The list may also not contain too many pixels, as this indicates that the moon, earth, or reflections from the sun are entering the FOV. Lists with too few or too many pixels are discarded. Choosing values for the minimum and maximum number of pixels can only be done once the performance of the hardware is established. See Section 5 for more details.

4.1.3 Centroiding

The centroiding algorithm takes a list of pixels belonging to a star (as found by the Region Growing Algorithm) and finds the star's centroid. As explained in Section 4.1.1, the accuracy of the star tracker is directly related to the accuracy of the centroiding algorithm.

The are two centroiding algorithms which are commonly employed by star trackers: centre of gravity and Gaussian curve fitting. As mentioned in Section 4.1.1, the pattern of light caused by a single star on the image plane can be approximated as a 2D Gaussian function. Gaussian curve fitting attempts to fit a Gaussian function to the pattern of light caused by each star. Once this is achieved, the centroid of the star can be found by calculating the location of the fitted Gaussian function's peak. This is the more accurate of the two methods [31]. However, Guassian fit is a very processor intensive operation.

Compared to Gaussian curve fitting, a centre of gravity equation is less accurate, but

it is also simpler and far less processor intensive. SUNSAT's star tracker can find the centroid of a star to within 0.2 pixels using a centre of gravity equation [32]. A centroiding accuracy of 0.2 pixels will allow CubeStar to calculate its bore-sight pointing to within 0.01 degrees. Therefore, there is no need to implement the more complex Gaussian curve fitting algorithms.

Equations 4.1.1 and 4.1.2 are the basic centre of gravity equations used by CubeStar. The weight of each pixel is its intensity (or brightness). These equations are applied to each list of pixels, representing a detected star, generated by the Region Growing algorithm.

$$centroid_x = \frac{\sum_{i=0}^{totalpixels}[(pixel[i]_x)(pixel[i]_{intensity})]}{\sum_{i=0}^{totalpixels}[pixel[i]_{intensity}]} \tag{4.1.1}$$

$$centroid_y = \frac{\sum_{i=0}^{totalpixels}[(pixel[i]_y)(pixel[i]_{intensity})]}{\sum_{i=0}^{totalpixels}[pixel[i]_{intensity}]} \tag{4.1.2}$$

where

$centroid_x$ = horizontal centroid position in pixels

$centroid_y$ = vertical centroid position in pixels

$totalpixels$ = the number of star pixels to be used for centroiding

Instead of using all the pixels, only the pixels in a 5 x 5 grid around the brightest pixel of each star are used to calculate the centroid. This is done for two reasons. Firstly, it increases the speed of the centroiding process. Secondly, it ensures that only the brightest pixels are used for the centroiding process. The brighter the pixels are the less the centroid will be affected by noise. Experimenting with the size of this grid can be done once the star tracker is operational.

A discussion on the accuracy of this algorithm is not included in this thesis, as SUNSAT's star tracker has proven that the algorithm is accurate enough for CubeStar's purposes. Tests using simulated star images were conducted and are explained in Section 4.1.6.

4.1.4 Distortion Correction

Once the centroids of all the stars have been found, the centroid locations need to be undistorted. Unfortunately, an ideal pin-hole lens does not exist. Every physical lens causes some distortion which needs to be characterised and corrected for.

The most common form of lens distortion, called radial distortion, is radially symmetric about the optical axis (or bore sight). Radial distortion appears in two forms: Barrel distortion and Pincushion distortion. Barrel distortion is common on lenses with a wide FOV. It causes objects to appear further from the optical axis than expected. Fisheye lenses make use of this form of distortion to achieve very wide FOVs. Pincushion distortion causes objects to appear closer to the optical axis than expected. Both forms of distortion are quadradic, meaning their effect increases as the square of the distance from the optical axis.

While radial distortion is usually dominant, a small amount of tangential distortion is also often present. Tangential distortion is caused by misaligned lens elements, or an image sensor which is inclined slightly.

It is essential that as much distortion as possible is removed for machine vision applications. In the case of CubeStar, it is essential that the true, undistorted centroid locations be used during the matching process, otherwise the matching algorithm will fail. Fortunately, both radial and tangential distortion can be corrected using Brown's distortion model [33]. Equations 4.1.3 and 4.1.4 undistort image points using a version of Brown's distortion model simplified to include only the first two distortion coefficients. Higher order distortion coefficients are unnecessary for lenses with only a small amount of distortion.

$$x_u = x(1 + K_1 r^2 + K_2 r^4) + P_2(r^2 + 2x^2) + 2P_1 xy \tag{4.1.3}$$

$$y_u = y(1 + K_1 r^2 + K_2 r^4) + P_1(r^2 + 2y^2) + 2P_2 xy \tag{4.1.4}$$

with

$$x = \frac{x_d - x_c}{f_{pix}} \quad y = \frac{y_d - y_c}{f_{pix}} \tag{4.1.5}$$

where

x_u, y_u = undistorted image point in pixels

x_d, y_d = distorted image point in pixels

x_c, y_c = distortion centre in pixels (assumed to be the principal point)

f_{pix} = focal length in pixels

$K_n = n^{th}$ radial distortion coefficient

$P_n = n^{th}$ tangential distortion coefficient

$r = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$

The difficult part of the distortion correction procedure is determining the distortion coefficients. On high end lenses, these coefficients may be included in the lens's datasheet. However, since CubeStar's lens does not come with a detailed datasheet, the coefficients will have to be determined experimentally. This procedure is described in detail in Section 5.4.

4.1.5 Image Plane to Unit Vector

The final step before the matching process is to convert the distortion-corrected centroids from 2D coordinates on the image plane to Cartesian unit vectors in sensor body coordinates. This is the inverse of the more common problem of projecting points of a 3D scene onto a plane (the solution of that problem can be found in any source explaining how a pinhole camera works).

Equation 4.1.6 gives the conversion from a centroid location in pixels to a unit vector. This form of the equation is adapted from [34]. The full derivation can be found in Appendix B.

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} (x_u - x_c)\frac{pp_x}{f_{mm}}[1 + ((x_u - x_c)\frac{pp_x}{f_{mm}})^2 + ((y_u - y_c)\frac{pp_y}{f_{mm}})^2]^{-\frac{1}{2}} \\ (y_u - y_c)\frac{pp_y}{f_{mm}}[1 + ((x_u - x_c)\frac{pp_x}{f_{mm}})^2 + ((y_u - y_c)\frac{pp_y}{f_{mm}})^2]^{-\frac{1}{2}} \\ [1 + ((x_u - x_c)\frac{pp_x}{f_{mm}})^2 + ((y_u - y_c)\frac{pp_y}{f_{mm}})^2]^{-\frac{1}{2}} \end{bmatrix} \tag{4.1.6}$$

where

u_x, u_y, u_z = components of a unit vector

x_u, y_u = undistorted centroid coordinates in pixels

x_c, y_c = pixel coordinates of the principal point

pp_x, pp_y = pixel pitches of the imager

f_{mm} = focal length of the lens in mm

4.1.6 Simulation and Testing

In order to test the accuracy and robustness of the star detection and centroiding algorithms discussed in Sections 4.1.1-4.1.3, both artificial and real star images were used.

The Image Plane Search and Region Growing Algorithms were tested by running them on real star images from SUNSAT's star tracker. The algorithms proved to be robust even on noisy images. Even stars that were only a few levels above the background noise could be detected, and dead pixels were ignored. These algorithms have already been flight proven onboard SUNSAT's star tracker [32].

Testing the centroiding algorithm was more difficult. In order to determine the accuracy of the centroiding algorithm, it was necessary to know the exact location of each star in the image beforehand. This could only be achieved by generating artificial star images with stars at known locations. These artificial star images had to include the effect of a defocussed lens, which meant simulating the point spread function of the star light. Fortunately, as mentioned in Section 4.1.3, the point spread function of the starlight can be approximated as a 2D Gaussian distribution.

To generate a simulated star, Equation 4.1.7 is solved for each pixel in a 5 x 5 grid around the desired star's location. This method is reproduced from M. Knutson's thesis titled: *Fast Star Tracker Centroid Algorithm for High Performance CubeSat with Air Bearing Validation* [35]. The solution of the equation is a value between zero and one, which represents the brightness of the star at the given location in the 5x5 grid. This value, G(X), is multiplied by 255 (representing a pure white pixel) and plotted on the image. The standard deviation is related to the amount that the lens is defocussed. A larger value, representing a more defocussed lens, spreads the starlight out over more pixels. A value of one was found to produce a good spread over the 5 x 5 pixel grid.

$$G(X) = e^{-J} \tag{4.1.7}$$

$$J = \frac{1}{2}(X - \bar{X})^T \Sigma^{-1} (X - \bar{X}) \tag{4.1.8}$$

where

$$G(X) = \text{the Gaussian function solved at discrete points X}$$

$$X = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \text{ coordinates of discrete points on the image plane}$$

$$\bar{X} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} \text{ coordinates of the centre of the star}$$

$$\Sigma = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix} \text{ where } \sigma \text{ is the standard deviation}$$

Some example simulated star images that were generated using this method are shown in Figure 4.3. The first three images show simulated stars and the forth image is of a real star for comparison.

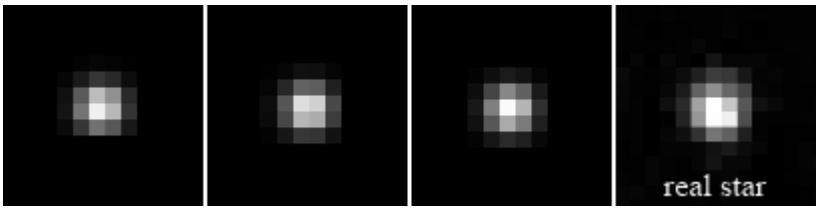


Figure 4.3: Three simulated star images generated using a 2D Guassian distribution, and a real star for comparison. The images are zoomed to reveal individual pixels.

A simulated star image was generated with 18 randomly placed stars at known locations. The image Plane Search, Region Growing and centroiding algorithms were run on the image to test the accuracy of the star detection procedure. The 18 stars were found with an average centroiding accuracy of 0.0427 pixels in the Y direction and 0.0244 pixels in the X direction. This corresponds to an accuracy of about 0.002 degrees, which is almost an order of magnitude better than required by CubeStar. However, it is important to note that the simulated star image was ideal. It contained no lens distortion or noise, so the the real life accuracy of CubeStar will be less accurate. However, the test did prove that the star detection, extraction and centroiding algorithms perform well.

4.2 Star Matching Overview

The matching algorithm is responsible for determining which catalogue star produced each detected centroid. It must make use of the available information present in the distribution of the centroids to match each centroid to a star in the catalogue, or reject it as a false star. It takes a list of centroids in the form of cartesian unit vectors in sensor coordinates and outputs a list of corresponding inertial unit vectors from the star catalogue. These lists of vectors can then be used to determine the sensor's bore sight pointing using the algorithm described in Section 4.4

4.2.1 Brief Description of Important Matching Algorithms

Many suitable star matching algorithms have been developed since the 1970's when the first primitive CCD based star tracker was developed [36]. The algorithms differ in how they select features to match, how the star catalogue is structured and searched and whether there are any validation steps. While the algorithms use different strategies for matching, they all compare star catalogue information to stars detected in the image. The majority of matching algorithms can be classified as implementations of either subgraph matching or pattern matching. Three of the most fundamental and important algorithms are:

- Triangle Algorithm
- Match Group Algorithm
- Grid Algorithm

4.2.1.1 Triangle Algorithm

The Triangle Algorithm is a variant of the earliest star matching algorithms and is likely the most implemented technique to date [37]. It is a member of the subgraph matching family. It attempts to match an observed triangle feature to an isomorphic triangle from an onboard catalogue. A triangle feature is created by taking any three observed stars as the vertices of the triangle. Isomorphic catalogue triangles can then be found in one of two methods: side-angle-side (SAS) or side-side-side (SSS). Using SAS, the angular distances of two of the triangle's sides are measured, together with the angle between them, and compared to the catalogue entries. SSS uses just the three angular distances. The onboard catalogue needs to contain an entry for every observable triangular feature over the whole night sky. The number of catalogue entries

is proportional to the sensitivity of the camera and the camera's FOV. The catalogue can become very large for sensitive sensors, requiring long search times. For this reason, several methods to minimise and optimise the catalogue have been developed.

One such modification is to always pick the brightest three stars for matching, instead of randomly picking any three stars. However, to increase accuracy, it is often desirable to try to match all the triangular features in an image, instead of just one. Brightness information can also be stored in the star catalogue to speed up the catalogue search. Noise and measurement error require a tolerance to be selected when searching for matching triangles. However, the tolerance may cause several triangles to be matched to each observed triangle. To prevent this, some method of match verification is required. Verification can be achieved by comparing star brightness to catalogue brightness, or by attempting to match a fourth star in the vicinity of each matched triangle.

4.2.1.2 Match Group Algorithm

The Match Group Algorithm is also a member of the subgraph matching family [37]. It attempts to identify a star by checking the distances to each of its neighbouring stars. The star catalogue consists of entries containing a pair of stars and the angular distance between them. Every pair of stars which can be viewed simultaneously within the sensor's FOV is included in the catalogue.

The algorithm begins by randomly selecting a single star in the image. This star is called the "pole star". The distance between the pole star and each of the other stars in the FOV is measured. For each of these distances a lookup is performed in the catalogue. Brightness information can be used to minimize the number of catalogue "hits". Hopefully only one pair of stars will be returned for each distance lookup. However, it is possible that more than one match is returned due to noise and measurement error. After all the distances have been looked-up the resulting matched star pairs are examined. Since the pole star is one of the stars in each pair whose distance was looked up, the catalogue star identity (ID) that appears most often in the matched pairs is most likely the pole star. The pole star and its neighbours are called a match group. This process is repeated by giving each star in the FOV a chance to be the pole star.

During the previous step only the distances between the pole star and its neighbours are calculated. To verify each match group, the inter-neighbour distances can now be checked. The match group with the most verified matches is considered the most reliable and is used to calculate the sensor's pointing.

4.2.1.3 Grid Algorithm

The Grid Algorithm was published by Padgett in 1997 and was the first algorithm that relied on pattern matching [38]. Instead of measuring inter-star angles or distances, the Grid algorithm tries to associate a unique pattern or "fingerprint" with each star. This fingerprint is generated by placing a loose grid over the star and checking which of the grid's cells contain neighbouring stars. The fingerprint is stored as a bit string. In order to perform star matching using this algorithm, a fingerprint of each star must first be generated and stored in an onboard catalogue. This catalogue then contains one bit string per star, making it much smaller than the catalogues required by the other matching algorithms. When a new star image is taken the fingerprint of each star in the image is generated and compared to the fingerprint of each star in the catalogue. A linear search through the catalogue must be performed as there is no way to order the fingerprints. Despite this, the largest advantage of the Grid Algorithm is that it does not require any multiplications or trigonometric operations, significantly reducing the load on the processor.

4.2.2 Comparison of Matching Algorithms

Over the years, many derivations and modifications of these algorithms have been developed, each with marginal improvements over the previous ones. Doing a complete comparison between the different matching algorithms would be a very time consuming undertaking and is therefore beyond the scope of this thesis. Instead, the choice of matching algorithm was based on the results of several other comparison studies and the specifications and limitations of the CubeStar project specifically.

The CubeStar hardware puts limitations on the requirements of the matching algorithm. The processor's 1 MB of internal flash memory is the only flash memory present on CubeStar. Thus, the entire star catalogue, star list and program code need to fit within 1 MB. This restricts the algorithm choices to those with small catalogue requirements. Star trackers with small fields of view and high sensitivity can produce dense stellar images. However, due to CubeStar's wide FOV and low stellar magnitude cut-off, many of its stellar images include only a few widely spaced stars. Therefore algorithms which can use all the stars in the FOV, as opposed to those which focus on a small constellation of stars somewhere in the FOV, are preferred.

Table 4.1 shows a summary of a comparison study done by the Jet Propulsion Laboratory (JPL) in 1996 [37]. Although the study is old, three of the most important and commonly used algorithms had already been developed and are compared. These in-

clude a version of the Triangle Algorithm, van Bezooijen's version of the Match Group Algorithm, and the Grid Algorithm. The algorithms were compared by looking at the required catalogue size, RAM usage, execution speed and robustness to inaccurate measurements and false stars. The comparison concluded that the Grid Algorithm was the most efficient in all categories, and that the Triangle Algorithm was the worst[1]

Property	**Algorithm**		
	Triangle	**Match Group**	**Grid**
Catalogue Size	Large	Medium	Small
RAM Usage	Low	High	Low
Speed	Slow	Faster	Fastest
Robustness	Lowest	Medium	Highest

Table 4.1: Summary of a 1996 JPL Comparison of Matching Algorithms

A newer comparison paper by Spratling and Mortari was published in 2009 [36]. It examines the evolution of the matching algorithms. The paper notes the advantages of the Grid Algorithm. However, it goes on to explain that advances in database search techniques have resulted in even faster algorithms. These include the Search-Less-Algorithm (SLA), the Pyramid Algorithm and the Geometric Voting Algorithm, all of which make use of Mortari's *k-vector* database search technique. The paper concludes that the two fastest and most robust algorithm families to date are versions of the Grid Algorithm, and versions of the Search-Less Algorithm.

The Flash storage limitations of CubeStar rule out the Triangle Algorithm, as it requires the largest star catalogue. The Triangle Algorithm is also the slowest algorithm of those compared, making it the least desirable. The Grid algorithm requires dense stellar images as it uses each star's close neighbours to generate a unique fingerprint. Sparse or spread out stellar images, like those produced by CubeStar, will not produce good fingerprints. Unfortunately, while versions of the Grid Algorithm are very fast and memory efficient, they are not suitable for the CubeStar project. This leaves versions of the Search-Less-Algorithm.

The Geometric Voting Algorithm, a modification of the Search-Less-Algorithm, was chosen as CubeStar's matching algorithm. The algorithm is explained in detail in Section 4.3.

[1]It should be noted that this comparison was written by two of the authors of the Grid Algorithm.

4.3 Geometric Voting Algorithm

4.3.1 Algorithm Overview

The Geometric Voting Algorithm, published by Kolomenkin in 2008, is a modification of the Search-Less Algorithm [31]. It was chosen as CubeStar's matching algorithm. It was chosen above other versions of the Search-Less-Algorithm for the following reasons:

- It claims to be as fast as the fastest published methods to date, while being more robust.
- Kolomenkin's paper is well written and contains psuedocode, minimising implementation time.
- The University of Texas has also chosen the Geometric Voting Algorithm for their CubeSat star tracker[2]. It is currently in development but scheduled to fly onboard two CubeSats in the next two years [39].

The Geometric Voting Algorithm consists of a voting scheme based on pairs of stars. For every imaged pair of stars the inter-star distance is measured. Then the catalogue is searched for star pairs with similar inter-star distances. Each of the imaged stars in the pair then gets a vote from each of the stars of the matching catalogue pair. These votes are possible identities for the imaged stars. Once all the star pairs in the image have been considered, each imaged star will have many votes from different catalogue stars. Usually, the correct identity for the imaged star is the one which got the most votes. This voting stage is followed by another verification voting stage before the matching process is considered complete.

4.3.2 Star Catalogue

The star catalogue required by the Geometric Voting Algorithm is divided into two parts: a star list and a list of distances between star pairs. Both lists are generated by a MATLAB script before launch and stored in CubeStar's flash memory. The star catalogue has the same form as that required by the Van Bezooijen Algorithm used on SUNSAT's star tracker [32].

[2]This star tracker is not an independent unit. Instead, it is composed of an off-the-shelf camera connected to the satellite's main computer.

Star List

The star list contains all stars which are brighter than a set magnitude. On CubeStar this is set to magnitude 3.8 as discussed in Section 3.4. Each entry in the list contains a star identification number and the star's position in celestial coordinates.

The European Space Agency's (ESA) Hipparcos satellite was launched in 1989 with the aim of mapping the stars [40]. Between 1989 and 1993 it mapped the positions of 118200 stars to an unprecedented accuracy of better than 0.001 arc seconds. It also mapped the positions of more than one million stars to a lower but still impressive accuracy of 0.03 arc seconds using a secondary payload. The main results of the mission were the Hipparcos and Tycho star catalogues, completed in 1996.

The sheer size of the Hipparcos and Tycho catalogues make them difficult to use. Fortunately, several online search tools such as the one on ESA's site [41] allow a reduced catalogue to be downloaded. A version of the catalogue listing stars only down to magnitude 7 was downloaded. This catalogue was filtered using a MATLAB script to remove all stars dimmer than magnitude 3.8, leaving 405 entries. The right ascension and declination in decimal degrees of each entry are converted to a Cartesian ECI unit vector in radians before being added to CubeStar's star list. The conversion process is simply a conversion from spherical to Cartesian coordinates. Table 4.2 displays a small part of the final star list.

Star ID	**ECI Unit Vector (x, y, z)**		
31	0.6919	0.4388	0.5734
32	0.5152	0.3492	-0.7827

Table 4.2: Example extract from CubeStar's onboard star list

Inter-Star Distance List

The second part of the catalogue is a list of distances between star pairs. This list is formed by calculating the distance between every star and every other star from the list of 405 stars. Equation 4.3.1 gives the angular distance between two stars, where x,y, and z represent the ECI unit vector components of stars S_i and S_j:

$$D_{ij} = acos(x_i x_j + y_i y_j + z_i z_j) \tag{4.3.1}$$

Each distance list entry consists of two star IDs and the angular distance between them in radians. The list is sorted by increasing angular distance. Psuedocode for the

distance list generation is given in Algorithm 1. Several steps are taken to minimise the size of the distance list:

- Only one entry per pair of stars is added. The distance between stars S_i and S_j is the same as the distance between stars S_j and S_i.
- Only star pairs whose inter-star distance is less than CubeStar's diagonal FOV are added to the list.
- Only star pairs whose inter-star distance is large enough that they can be discerned as two individual stars are added to the list.

By initially generating the distance list without checking for a minimum inter-star distance, it was discovered that there are only two pairs whose inter-star distances may be too small. One pair will appear less than two pixels apart (0.0017 rad), and the other will appear approximately 7 pixels apart (0.0067 rad). The pair that will appear less than two pixels apart will definitely be detected as a single star. Two options exist to deal with this problem. One option is to combine the two stars into a single star with an averaged position in space. The other option is to leave both stars in the catalogue. The combined imaged star may be matched to either of the catalogue stars. This will give a maximum error of half the inter star distance, which is less than one pixel. Combined with the fact that other correctly matched stars in the FOV will increase the estimated attitude accuracy, an error of less than 1 pixel (or 0.05 degrees) is deemed acceptable.

The pair of stars which are approximately 7 pixels apart should be detected as two separate stars, as real sky images have shown that only the very brightest stars in the sky are more than 7 pixels in diameter. At 2.8 and 3.6 magnitude, the two stars in this pair should not be bright enough to merge into one star. However, if they do merge, the average centroid should be sufficiently different from either of the catalogue stars that no match will be made. Alternatively, the combined star will consist of too many pixels and will be ignored by the star detection algorithms (See Section 4.1.2). No match is better than a false match, as the other stars in the FOV should match correctly. Therefore, the decision was made to not exclude any star pairs from the distance list for not meeting minimum separation.

The final distance list contains 11486 entries, of which a small part is shown in Table 4.3.

Star 1 ID	Star 2 ID	Angular Separation (rad)
31	401	0.695071
32	92	0.700600

Table 4.3: Example extract from CubeStar's onboard inter-star distance list

Algorithm 1 Catalogue Distance List Generation

```
for i=1 to i= length, N, of star list - 1 do
  for j=i+1 to j=N do
    Compute the inter-star distance D_ij =| P_i − Pj |
    if D_ij ≤ FOV and D_ij ≥ minimum separation then
      Append entry (i, j, D_ij) to table T(ID1, ID2, d)
    end if
  end for
end for
Sort T according to distance d
```

4.3.3 Detailed Description

In this description of the Geometric Voting Algorithm the terms *imaged stars* or *possible stars* refer to the centroids identified by the region growing and centroiding algorithms. Each of these centroids is probably a star and must be matched to a catalogue star by the matching algorithm. Whenever reference is made to the distance between two stars, the angular distance in radians is implied.

The Geometric Voting Algorithm is split into five steps:

1. Calculate inter-star angular distances of all imaged stars
2. First round of voting
3. Preliminary matching based on first round of votes
4. Verification round of voting
5. Final list of verified matches determined based on verification votes

These steps are explained in detail below and psuedocode is provided in Algorithm 2.

A list of centroids of imaged stars in the form of camera-centered Cartesian unit vectors is supplied by the star detecting algorithms described in Section 4.1. Just as with the generation of the star catalogue, the first step is to calculate the inter-star distance of each pair of imaged stars using Equation 4.3.1.

Every imaged star S_i gets its own voting list V_i. This list is simply an array of possible identities in the form of catalogue star IDs. The purpose of the first voting round is to fill these lists. For every pair of imaged stars S_i and S_j the inter-star distance D_{ij} has been calculated during the first step. Unfortunately, there will always be some error e in this distance due to inaccuracies in the detecting hardware (see Section 4.1). Therefore, the actual distance between the two stars lies somewhere in the range $R_{ij} = [D_{ij} - e_{ij}, D_{ij} + e_{ij}]$. The distance list T of the star catalogue is searched and all entries with distances that fall within R_{ij} are returned. Each of the returned entries contains two catalogue star IDs. Each of these IDs is added to the voting list V_i and V_j of imaged stars S_i and S_j respectively. What this process represents is the fact that either of the imaged stars could be either of the catalogue stars as both pairs have similar inter-star distances. Unfortunately, due to the error e, every search of the catalogue can return multiple entries. Therefore the identity of the imaged pair can not be determined definitively. After every pair of imaged stars has been considered the first round of voting is complete. Each imaged star's voting list will now contain many votes.

In most cases, the catalogue star ID which appears most often in an imaged star's voting list will be its identity. Therefore, every imaged star's identity is preliminarily set to the ID which got the most votes.

All of the imaged stars S_i now have matching catalogue star identities St_i, however, they still need to be verified. This is achieved by comparing the distance $\mid St_i - St_j \mid$ to R_{ij} for all pairs. If the difference is small, both stars receive a verification vote. This verification step is simply comparing the distance between a pair of imaged stars to the distance between their assumed identities. Once all the imaged stars pairs have been through the verification process, each will contain many, or close to no votes.

It is easy to tell which stars are correctly matched as they will each have close to the maximum number of verification votes among all stars. For example, eight correctly identified stars may each have six or seven votes while two false stars may have only a single vote each. The original algorithm described in Kolomenkin's paper suggests using only those stars which received at most one less than the maximum number of verification votes as correctly matched stars [31]. However, this threshold can be experimented with.

4.3.4 Assisted Matching

The Geometric Voting Algorithm is able to match the stars in an image to stars from an onboard database without any additional attitude information. This qualifies the star tracker as fully autonomous. However, considering a satellite always has several other attitude sensors, its ADCS system should always have a rough idea of the satellite's attitude. It would be wasteful to not make use of this rough attitude estimate when trying to match stars. Therefore, the Geometric Voting Algorithm is extended to include an Assisted Match Mode.

In Assisted Match Mode the satellite OBC (or the previous attitude estimate) supplies CubeStar with a rough boresight-pointing vector. This vector is used to generate a reduced star catalogue to be used subsequently by the matching algorithm. No modification to the matching algorithm is required to use the reduced catalogue as opposed to the full catalogue. Therefore, the Assisted Match Mode requires the extra step of generating the reduced catalogue, but the reduced time required to search through the reduced catalogue should compensate for this.

Reduced Catalogue Generation

The reduced star catalogue should contain only those stars which should be visible to CubeStar, based on the given rough boresight vector. Therefore, to generate the reduced catalogue, only those stars that are within a certain angular distance D_{max} of the boresight vector are selected from the full star catalogue. CubeStar has a diagonal FOV of approximately 60 degrees (radius of 30 degrees). The rough boresight vector from the satellite OBC is expected to be accurate to within ± 5 degrees, therefore, D_{max} is chosen as 35 degrees.

Equation 4.3.1 is used to calculate the distance between the rough boresight pointing vector (in the form of an inertial Cartesian unit vector) and every star in the full star catalogue. If a star is calculated to be within 35 degrees of the boresight vector, it is added to the reduced catalogue. The reduced catalogue is generated live during every iteration of the star tracker, if operating in Assisted Match Mode.

Advantages over Lost-In-Space and Tracking Modes

There are several advantages of the Assisted Match Mode over the Lost-In-Space (LIS) Mode and over more traditional tracking modes. These include:

- More robust than LIS mode as there are fewer stars to choose from when matching.
- Faster than LIS mode.
- Does not suffer from problems relating to stars entering and leaving the frame between iterations, which is often a problem for tracking modes.
- Could potentially be used for auto-exposure, as the star tracker knows how many stars it should be seeing.

Algorithm 2 Geometric Voting Algorithm - Executed live on the star tracker

Detect n possible stars S_i in the image
for all possible stars **do**
  Calculate centroid P and convert to camera-centred unit vector
end for
Determine average centroid uncertainty e
for i=1 to i=n-1 **do**
  for j=i+1 to j=n **do**
    Compute the inter-star distance $D_{ij} =\mid P_i - Pj \mid$
    if $D_{ij} \neq 0$ **then**
      Compute the distance uncertainty region $R_{ij} = [D_{ij} - e_{ij}, D_{ij} + e_{ij}]$
      Locate all entries k in T that fall within the region R_{ij}
      for all entries T_k **do**
        Append to the voting lists V_i and V_j of possible stars S_i and S_j respectively the two catalogue stars $T_k.ID1$ and $T_k.ID2$
      end for
    end if
  end for
end for
for all possible stars S_i **do**
  Assign St_i to the catalogue star which got the maximum number of votes
end for
for i=1 to i=n-1 **do**
  for j=i+1 to j=n **do**
    if $\mid St_i - St_j \mid$ is within the uncertainty region R_{ij} **then**
      Add a vote for the match (S_i, St_i) and for the match (S_j, St_j)
    end if
  end for
end for
All possible stars whose votes are clustered together are assumed correct

4.3.5 Simulation and Testing

Testing of the matching algorithm was performed in the MATLAB environment. The matching algorithm was implemented as an unoptimised MATLAB script. A separate MATLAB script described in Appendix A was used to generate artificial star images. These images simulate images captured by CubeStar's camera with a focal length of 6 mm, a resolution of 1024 x 512 pixels, a FOV of 54 x 28 degrees and variable magnitude cutoff. Lens distortion is not included in these images.

Magnitude Cutoff Test

The first test aimed to determine the effect of magnitude cutoff on the performance of the matching algorithm. Three sets of one hundred random star images were generated with magnitude cutoffs of 3.6, 3.7 and 3.8 respectively. A magnitude cutoff of 3.6 was determined to be the minimum sensitivity required to always (3 σ) have at least three stars within the FOV (See Section 3.5). These sets of images were input into the matching algorithm to determine the percentage of correctly matched stars in each image. Simultaneously, the matching algorithm was tested in three different modes: Lost-In-Space, Assisted and Assisted Perfect. For the Lost-In-Space tests no additional attitude information was given to the matching algorithm. For the Assisted Match tests an attitude with a random error of up to 5 degrees was given to the Assisted Matching Algorithm (Section 4.3.4), and for the Assisted Perfect tests the exact attitude was given to the Assisted Matching algorithm. The results of the tests are shown in Table 4.4.

	Lost-In-Space	**Assisted $\pm 5°$**	**Assisted Perfect**
Cutoff Mag 3.6			
% Stars Matched	75.45	84.7	88.21
% >2 Matched	80	89	93
Cutoff Mag 3.7			
% Stars Matched	85.2	86.62	90.76
% >2 Matched	92	94	96
Cutoff Mag 3.8			
% Stars Matched	84.01	89.62	91.84
% >2 Matched	93	98.5	98.5

Table 4.4: The effect of magnitude cutoff on the performance of the matching algorithm

The *Percentage Stars Matched* field shows how many of the stars in each image were correctly matched, averaged over the hundred images. It is interesting that even though the input images contain no distortion, not all the stars are matched correctly. This is

due to the chosen search tolerance when searching the star catalogue. For the simulated images this search tolerance could, in theory, be made very small resulting in far better matching. However, using a more realistic search tolerance of 0.001 radians (about 1 pixel) gives a better prediction of real life performance. It is important to note that the algorithm did not output any incorrect matches, it simply output no match for certain stars. This is important as incorrect matches will degrade the accuracy of the calculated attitude output, whereas stars with no match will have no effect on the calculated attitude. Those stars which were not matched were probably incorrectly matched in the first part of the matching algorithm, but were subsequently disqualified by the verification stages of the algorithm. While the *Percentage Stars Matched* results reveal something about the inner workings of the matching algorithm, they are not the most important results.

The *Percentage Greater Than 2 Matched* field is the most important result. It shows how many of the hundred images contained at least three correctly matched stars. Three correctly matched stars is the minimum needed to calculate three-axis attitude. Since the hundred images were randomly generated over all parts of the celestial sphere, these results can be interpreted as predicted sky coverage. Therefore, if CubeStar can detect stars as faint as magnitude 3.8 and is lost in space, then it is expected to output a valid attitude over 93 percent of the celestial sphere. If the satellite's OBC can give CubeStar a rough attitude estimate, then CubeStar should be able to output a valid attitude over 98.5 percent of the celestial sphere. Table 3.6 in Section 3.5 shows that a cutoff magnitude of 3.8 results in at least five visible stars over 98.2 percent of the celestial sphere. This suggests that the matching algorithm seems to require at least five stars in order to reliably match at least three stars.

The *Assisted Perfect* results give an indication of CubeStar's expected sky coverage if it uses its own previous attitude match during the next assisted match. This is a possible alternative to the tracking mode (Section 4.5).

The most important results of this test are:

- CubeStar must detect stars down to magnitude 3.8
- CubeStar can be operated in full autonomous or assisted modes
- In Assisted Match Mode CubeStar has an expected celestial sphere coverage of 98.5%
- CubeStar should be able to recover from Lost-In-Space over 93% of the celestial sphere

False Stars Test

The second test aimed to determine the matching algorithm's robustness to false stars. Three tests consisting of 100 randomly generated star images each, were conducted. Each image in the first set of star images contained one randomly placed false star. The images from the second and third sets each contained 5 and 10 false stars respectively. The performance of the matching algorithm during these tests is shown in Table 4.5

	% of images with at least 3 stars matched	
No. of False Stars	**Lost-In-Space**	**Assisted** $\pm 5^\circ$
1	79	96
5	72	96
10	58	94

Table 4.5: The effect of false stars on the performance of the matching algorithm

The Lost-In-Space algorithm's performance degrades as more false stars are included in the images. When 10 false stars were included in the image, the algorithm's success rate dropped by almost 30 percent. Once again, no stars were incorrectly matched, but many stars failed to be matched. Fortunately, CubeStar's image sensor has dead pixel correction and its centroiding algorithms have several checks which should prevent the majority of false star detections (Section 4.1). The assisted match algorithm did not suffer the same magnitude of degradation in performance. Even with 10 false stars the algorithm still successfully matched at least three stars 94% of the time. This is an important result as assisted match mode (Section 4.3.4) is likely to be used far more often than lost-in-space mode.

Run Time Test

The final test investigated the runtime of the matching algorithm. It was important to have an order of magnitude estimate of the time required to process a star image in order to determine whether the algorithm would be suitable for implementing on a low-clock-speed embedded processor. MATLAB is an interpreted language, making it very slow compared to compiled languages such as C. Therefore it was unsuitable for examining the runtime of the matching algorithm. Instead, a rough and unoptimised version of the Geometric Voting Algorithm was written in C and executed on a desktop computer. The specifications of the desktop computer were:

- Intel Core i5-2500 CPU @ 3.30 GHz

- 4.0 GB RAM
- 32-bit Windows 7 OS

The C algorithm was given the same simulated star images as used during the MATLAB tests. Its runtime was measured using a system timer with 1 ms resolution. The algorithm completed the image processing and matching operation in 34 ms. Almost all of the time, 33 ms, was spent during the first voting stage of the algorithm. The cause of the bottleneck was determined to be the search through the star catalogue which had to be repeated for every pair of stars in the image. For this initial test the search was being done linearly. By replacing the linear search with binary search, the total runtime of this voting stage could be lowered significantly. For a catalogue of 405 entries, linear search requires 405 comparisons, while binary search would require nine at most. This would result in a reduction in runtime of the voting stage by more than an order of magnitude. The total predicted runtime of the matching algorithm would then be about 2 ms.

As described in Section 3.2, CubeStar will make use of an ARM Cortex M3 processor clocked at 48 MHz. A rough calculation of the expected runtime of the matching algorithm on this processor is given in Equation 4.3.2.

$$\text{Expected Runtime} = 2 \text{ ms } \times \frac{3.3 \cdot 10^9}{48 \cdot 10^6} = 137.5 \text{ ms} \tag{4.3.2}$$

During CubeStar's operation, approximately 500 ms will be available for image processing after each 500 ms image exposure. The matching algorithm is expected to take up the largest portion of this time. 137.5 ms is well within 500 ms, therefore, this rough test shows the feasibility of implementing the Geometric Voting algorithm on CubeStar's embedded processor.

More detailed runtime tests, including the effect of number of stars in the image on the runtime, were conducted on the embedded system and are described in Section 5.3.

4.4 Attitude Determination

The aim of the attitude determination algorithm is to use a pair of vector lists, one in inertial coordinates and the other in sensor body coordinates, to determine the rotation matrix, $\mathbf{R}_{bi}$, between the inertial frame and the sensor body frame. These vector lists, known as the reference and measured vectors, are the result of the star

matching process. Several algorithms for determining $\mathbf{R}_{bi}$ are discussed in this section. The following notation is used throughout this section:

$\mathbf{v}_{ki}$ - Vector number k in inertial coordinates
$\mathbf{v}_{kb}$ - Vector number k in sensor body coordinates

Ideally, $\mathbf{v}_{kb}$=$\mathbf{R}_{bi}\mathbf{v}_{ki}$ for each vector pair k. However, due to the overdetermined nature of the problem, it is generally impossible to find $\mathbf{R}_{bi}$ by solving the problem in this form. There are two branches of solutions to the problem. The first branch, which can give an analytic solution to the problem, includes the TRIAD algorithm. The second branch involves constructing and minimising a loss function.

4.4.1 TRIAD

One of the earliest solutions to the spacecraft attitude determination problem was the TRIAD algorithm. Unlike most of the other attitude determination algorithms which involve an iterative search for $\mathbf{R}_{bi}$, the TRIAD algorithm provides an analytic solution. The TRIAD algorithm makes use of two pairs of measured and reference vectors, $(\mathbf{v}_{1b}, \mathbf{v}_{1i})$ and $(\mathbf{v}_{2b}, \mathbf{v}_{2i})$, and makes use of an intermediate reference frame. The three components of this intermediate reference frame, $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$, are expressed in both sensor body coordinates and in inertial coordinates.

The first component of the intermediate reference frame is constructed by assuming that one of the two measured vectors is error free. This direction is used as the first component of the intermediate reference frame. Expressed separately in inertial and sensor body coordinates:

$$\mathbf{t}_{1b} = \mathbf{v}_{1b} \tag{4.4.1}$$

$$\mathbf{t}_{1i} = \mathbf{v}_{1i} \tag{4.4.2}$$

The second component of the intermediate reference frame is chosen as the direction perpendicular to the two measured vectors:

$$\mathbf{t}_{2b} = \frac{\mathbf{v}_{1b} \times \mathbf{v}_{2b}}{|\mathbf{v}_{1b} \times \mathbf{v}_{2b}|} \tag{4.4.3}$$

$$\mathbf{t}_{2i} = \frac{\mathbf{v}_{1i} \times \mathbf{v}_{2i}}{|\mathbf{v}_{1i} \times \mathbf{v}_{2i}|} \tag{4.4.4}$$

Finally, the third component of the intermediate reference frame is chosen perpendic-

ular to the other two components:

$$\mathbf{t}_{3b} = \mathbf{t}_{1b} \times \mathbf{t}_{2b} \tag{4.4.5}$$

$$\mathbf{t}_{3i} = \mathbf{t}_{1i} \times \mathbf{t}_{2i} \tag{4.4.6}$$

By placing the intermediate reference vector components into the columns of two 3x3 matrices, two rotation matrices, $\mathbf{R}_{bt}$ and $\mathbf{R}_{it}$, can be constructed. Finally, to obtain the desired rotation matrix, $\mathbf{R}_{bi}$, it is only necessary to multiply $\mathbf{R}_{bt}$ and $\mathbf{R}_{ti}$ together, as shown in Equation 4.4.7.

$$\mathbf{R}_{bi} = \mathbf{R}_{bt}\mathbf{R}_{ti} = \begin{bmatrix} \mathbf{t}_{1b} & \mathbf{t}_{2b} & \mathbf{t}_{3b} \end{bmatrix} \begin{bmatrix} \mathbf{t}_{1i} & \mathbf{t}_{2i} & \mathbf{t}_{3i} \end{bmatrix}^T \tag{4.4.7}$$

While the TRIAD algorithm is still in use today onboard satellites, it has a major downside which makes it poorly suited for use on a star tracker. The TRIAD algorithm uses only two pairs of vectors to determine its attitude estimate, while a star tracker provides one pair of matched vectors for each matched star. In CubeStar's case, this means that up to 20 matched pairs are available for attitude determination, but only two can be used by the TRIAD algorithm. Therefore, a large amount of useful information is discarded. The TRIAD algorithm is still useful for determining attitude from other sensor measurements, such as a sun and horizon vector, but it is not well suited for use on CubeStar.

The second branch of attitude determination algorithms, including Davenport's q-method, QUEST and SVD [42], provide statistical solutions. They are most suited to the case where more than two vector pairs are available. Since each set of measurements will include some error due to to noise or other sources, it is impossible to find a single $\mathbf{R}_{bi}$ that satisfies $\mathbf{v}_{kb}$=$\mathbf{R}_{bi}\mathbf{v}_{ki}$ for each vector pair k. Instead, it is desirable to find an $\mathbf{R}_{bi}$ which is the "best-fit" over all the vector pairs. The problem can be stated as: find a matrix $\mathbf{R}_{bi}$ that minimises the loss function:

$$J(\mathbf{R}_{bi}) = \frac{1}{2}\sum_{k=1}^{N} \omega_k |\mathbf{v}_{kb} - \mathbf{R}_{bi}\mathbf{v}_{ki}|^2 \tag{4.4.8}$$

where J is the loss function to be minimised and ω_k is a set of weights assigned to each vector pair ($\mathbf{v}_{kb}$, $\mathbf{v}_{ki}$). This approach was first proposed by Wahba in 1965 [43]. If the measurements are all perfect, then Equation 4.4.8 will be satisfied for all N vector pairs and $J = 0$. However, in practice there will always be some error in the measurements, resulting in $J > 0$. The smaller J can be made, the better the approximation of $\mathbf{R}_{bi}$.

The q-method provides an optimal least-squares estimate of the attitude. The derivation of the q-method is not reproduced here, but can be found in [44]. The method works by rearranging and restating the loss function using quaternions in such a way that it becomes an eigenvalue problem. The method requires finding the largest eigenvalue and corresponding eigenvector of a matrix. While this is easily done on a modern desktop computer using software such as MATLAB, it is very computationally intensive for an embedded system.

The QUEST algorithm approximates the q-method and was developed to bypass the expensive eigenvalue problem [45]. The QUEST algorithm uses an approximation to evaluate the largest eigenvalue. Again, the full derivation can be found in [44] or [45], but the final form of the problem involves solving for $\mathbf{p}$ in Equation 4.4.9. This is a problem of simultaneously solving multiple linear equations, which can be achieved using Gaussian elimination.

$$[(\lambda_{opt} + \sigma)\mathbf{1} - \mathbf{S}]\mathbf{p} = \mathbf{Z} \tag{4.4.9}$$

where

$$\begin{aligned}
\mathbf{B} &= \sum_{k=1}^{N} \omega_k (\mathbf{v}_{kb}\mathbf{v}_{ki}^T) \\
\lambda_{opt} &= \sum \omega_k \\
\sigma &= tr(\mathbf{B}) \text{ sum of the elements on the diagonal} \\
\mathbf{S} &= \mathbf{B} + \mathbf{B}^T \\
\mathbf{p} &= \text{the Rodriguez parameters, which must be solved for} \\
\mathbf{Z} &= \begin{bmatrix} B_{23} - B_{32} & B_{31} - B_{13} & B_{12} - B_{21} \end{bmatrix}^T
\end{aligned}$$

Once the Rodriguez parameters ($\mathbf{p}$) have been found, the attitude quaternion can be calculated using Equation 4.4.10

$$\overline{\mathbf{q}} = \frac{1}{\sqrt{1 + \mathbf{p}^T\mathbf{p}}} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{4.4.10}$$

where

$$\overline{\mathbf{q}} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4$$

The inventor of the QUEST algorithm, Malcom Shuster, claims that the QUEST algorithm is up to 1000 times faster than Davenport's the q-method [46]. Thanks to this computational efficiency, the QUEST algorithm has become the most frequently implemented batch three-axis-attitude estimation algorithm [47][48].

Since the 1970s, many other solutions to the attitude determination problem have appeared, including Markley's Singular Value Decomposition algorithm and FOAM algorithm, Mortari's ESOQ2 algorithm and several other adaptations [49]. Despite all of these new algorithms, no significant improvement in robustness or speed over the QUEST algorithm has been achieved. In fact, as the available computing power on spacecraft increases, the choice of attitude determination algorithm is becoming less significant.

The QUEST algorithm was chosen for the CubeStar project because of its long flight heritage, good efficiency and robustness. The QUEST algorithm should be simple to implement as there is a large amount of literature available on the topic.

A more complete comparison of modern attitude determination algorithms is available in Markley and Mortari's review [49].

The complete process from star image to attitude estimate was verified using MATLAB implementations of the algorithms. One hundred simulated star images from random parts of the sky were generated and input into the star tracker algorithms. The final attitude output of the QUEST algorithm was compared to the known attitude which produced each star image. The resultant attitude errors are shown in Figures 4.4 and 4.5. As is the case with all star trackers, the cross-boresight accuracy is higher than the rotation accuracy.

Figure 4.4 shows that the maximum cross-boresight error is approximately 0.004 degrees and Figure 4.5 shows that the maximum rotation error is approximately 0.02 degrees. This simulation does not include the effects of image sensor noise or lens distortion. However, it verifies the attitude determination algorithms and gives an estimate of CubeStar's "best-case" accuracy.

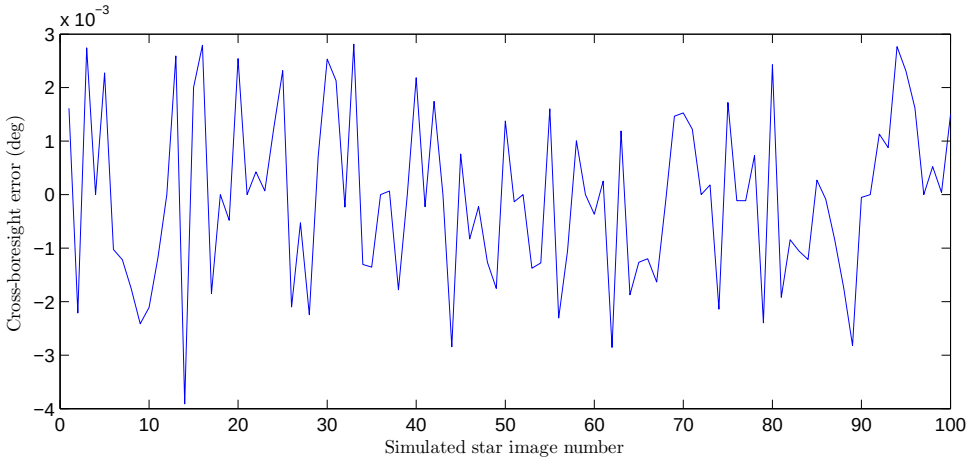


Figure 4.4: The cross-boresight accuracy of CubeStar's attitude determination algorithms over 100 simulated star images

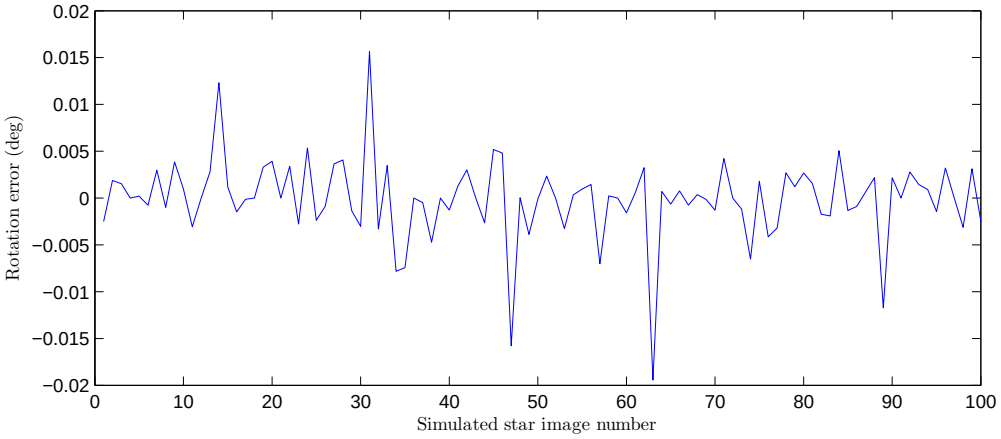


Figure 4.5: The rotation about the boresight accuracy of CubeStar's attitude determination algorithms over 100 simulated star images

4.5 Tracking Algorithm

A star tracking algorithm is an alternative to a star matching algorithm, under certain conditions. The essence of a star tracking algorithm is keeping track of individual stars between frames. If individual stars have been identified in the previous frame using a matching algorithm, it is often possible to find the same stars in the next frame without running the matching algorithm again. CubeStar has two properties which simplify the process of keeping track of stars between frames. Firstly, CubeStar has a wide FOV compared to most star trackers. This causes stars to move relatively slowly across its FOV. Secondly, CubeStar has a relatively small star catalogue, which reduces the probability of confusing neighbouring stars.

Most star tracking algorithms involve estimating how the stars will move between frames. With this knowledge, the tracking algorithm predicts where each previously identified star will appear in the next frame. However, on CubeStar, this is not necessary. CubeStar's tracking algorithm simply looks for the stars in the exact location (plus a surrounding region of 0.3 degree radius) where they were during the previous frame.

The tracking algorithm begins by generating a reduced star list based on the previous attitude estimate (the Assisted Match Algorithm begins the same way). Each entry in the reduced star list is then transformed to a sensor-centered unit vector using a transform based on the previous attitude estimate. All detected centroids are also converted to sensor-centered unit vectors as described in Section 4.1.5. Each detected centroid vector is compared to all the reduced star list vectors. If the angle between a reduced star list vector and a centroid vector is less than 0.3 degrees, the pair is matched. In this way, all the detected centroids should be matched to entries in the reduced star list. The output of the tracking algorithm has the same form as the output of the matching algorithm: a pair of matched vector lists.

The threshold of 0.3 degrees for matching vectors was decided on by examining the full star list. There is only one pair of stars which is less than 0.3 degrees apart and this pair will always be detected as a single star, as explained in Section 4.3.2. Therefore, as long as a centroid appears within 0.3 degrees of where a star is expected to be, the match is unambiguous. If the satellite's roll rates are greater than 0.3 deg/s, the star tracking algorithm will not function correctly.

Most star tracking algorithms store a list of matched stars from the previous iteration and search for the same stars during the next iteration. However, this approach leads to the problem of how to deal with stars that enter or leave the field of view between iterations. CubeStar's tracking algorithm does not require a list of matched stars from the previous iteration. Instead, all it requires is an attitude estimate from the previous iteration. CubeStar's tracking algorithm is less efficient as it has to generate a new reduced star list during every iteration, but it allows stars to enter and leave the FOV without problems.

Initially, CubeStar was not supposed to have a tracking algorithm, as the lost-in space and assisted matching algorithms can run at 1 Hz. However, the reliability of the matching algorithms is proportional to the number of stars in the FOV. The matching algorithms struggle when there are fewer than five stars in the FOV. A tracking algorithm requires only two stars to be present in the FOV. Therefore, by adding a

tracking algorithm to CubeStar, its reliability and sky coverage is increased.

The functionality of the tracking algorithm was tested by inputting a sequence of simulated star images generated from the attitudes of a simulated, slowly spinning satellite. Night sky tests, described in Section 6.2, further verified the functionality of the tracking algorithm.

Chapter 5

Implementation

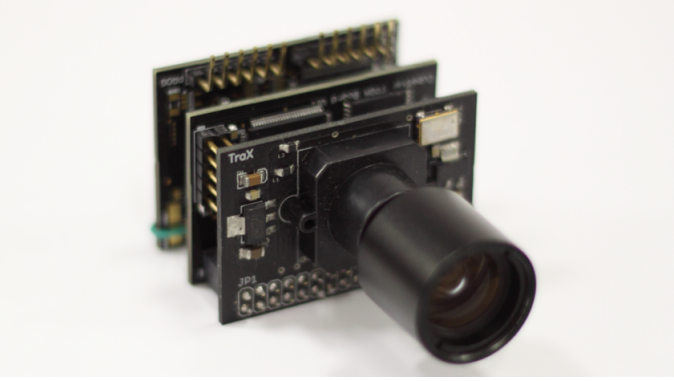


Figure 5.1: A complete engineering model of CubeStar, shown without a casing

5.1 Physical Layout

Most CubeSat components are implemented in a PC104 format. However, CubeStar was implemented as three separate printed circuit boards (PCBs) which stack on top of one another. Each PCB is approximately 3 x 4.5 cm and was designed to be as small as possible to minimise CubeStar's volume. This stacked layout was chosen over the traditional PC104 layout for several reasons:

- It makes the hardware modular and easier to troubleshoot.
- It allows CubeStar to be used on satellites other than CubeSats.
- The small dimensions allow two CubeStars to fit within 0.5 U volume. This is important as a single CubeStar can get blinded by the sun during parts of the orbit.

- Star trackers need to be angled to point away from the earth and sun. CubeStar's form allows it to be mounted at any angle relative to the satellite body.

Designing an enclosure for the stack was beyond the scope of this project. However, a temporary 3D printed enclosure was designed to protect CubeStar during outdoor testing. No mounting holes were included in the original CubeStar PCBs to minimise volume. It was envisioned that the PCBs would be held securely by slots milled into the casing walls. Unfortunately, this strategy did not work well for the 3D printed case as the case could not be printed accurately enough. Mounting holes will be added to the next design iteration of the CubeStar PCBs.

5.2 Electronics

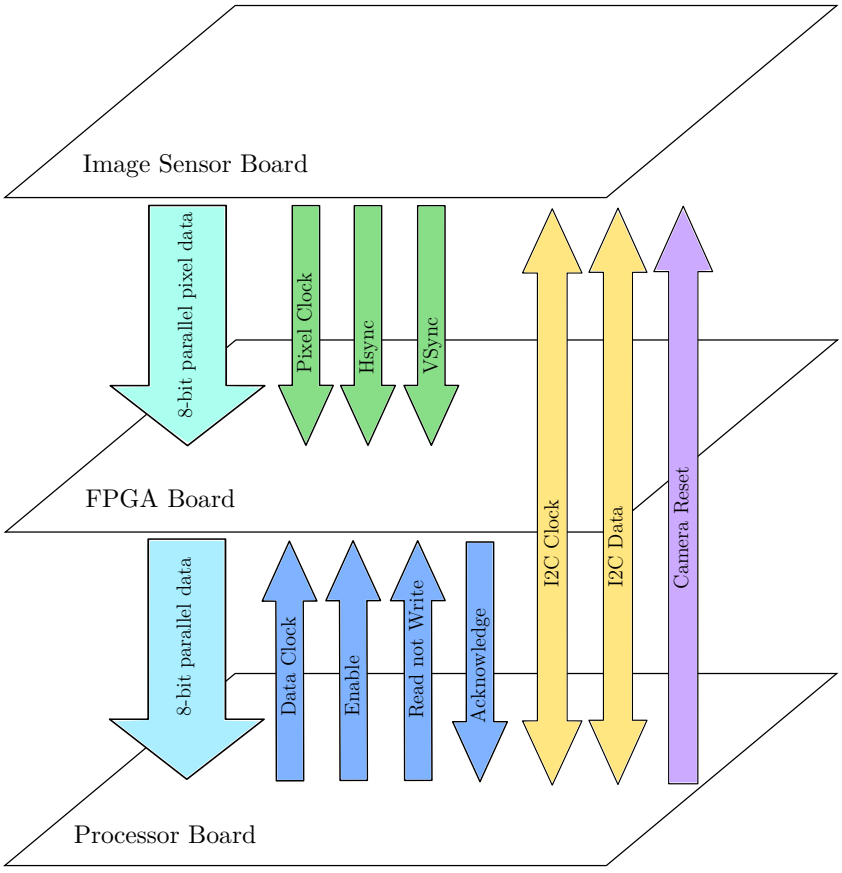


Figure 5.2: A diagrammatic representation of the three CubeStar PCBs and their data interconnections

The electronics stack consists of the image sensor board, the FPGA board and the processor board. The boards connect to each other using dual row PCB headers which

carry power and data signals. The flow of data between the boards is summarised in Figure 5.2 and is explained in more detail in the sections that follow. Additional information on the electronics, including design details and board pinouts, are available in the appendices.

5.2.1 Image Sensor Board

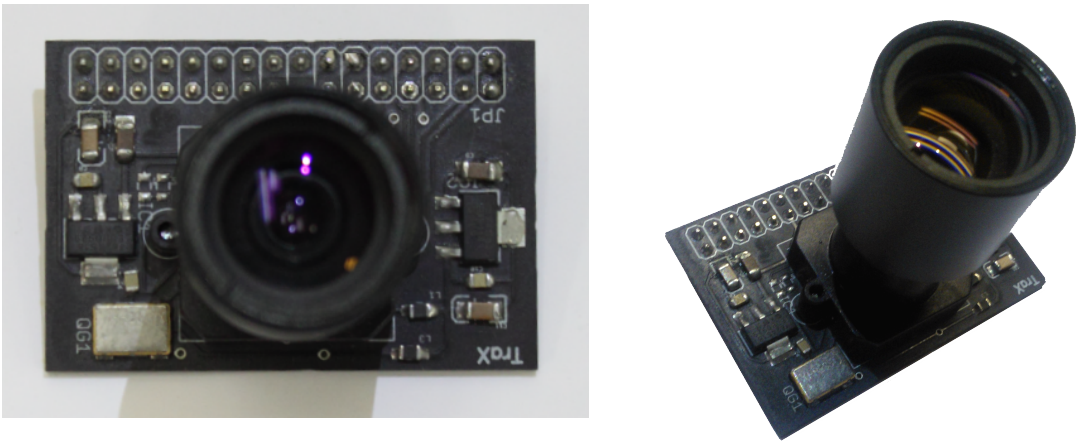


Figure 5.3: The image sensor board with Lensation lens

The image sensor board sits at the top of the stack and houses the image sensor and its supporting electronics and acts as a mounting point for the lens. As mentioned in Section 3.5.3, the design goal of the image sensor PCB was to make the Melexis image sensor compatible with CubeSense. Therefore, its pinout is compatible with the camera board sockets on CubeSense.

The image sensor board interfaces to both the processor board and the FPGA board as shown in Figure 5.2. The Gecko processor on the processor board can enable or disable the 3.3 V power supply to the image sensor board and can control the image sensor using I2C commands.

The Melexis image sensor requires both 1.8 V and 3.3 V sources. Two linear voltage regulators are included on the image sensor board, allowing the board to be powered off a single 5 V source for compatibility with CubeSense. However, when interfacing to the rest of the CubeStar stack, the 3.3 V regulator is replaced with an electrical short.

When enabled, the Melexis image sensor outputs a constant stream of images. The HSync pin goes low briefly at the end of every row of pixel data and the VSync pin goes low between images. Together with PIXCLK, these signals are used by the FPGA for data flow control.

The Melexis MLX75412 image sensor comes in a glass ball grid array (GBGA) package. A custom footprint had to be developed for the sensor in the PCB layout software Eagle. Eagle was initially chosen as the PCB layout software because the author had experience with this software from previous projects. The image sensor board was implemented as a two layer PCB and was manufactured by Trax.

The image sensor board was successfully tested by interfacing it to CubeSense. In order to capture images using the new Melexis image sensor, only minor modifications to the CubeSense firmware and ground support software were required. More information on firmware and software is available in Section 5.3.

5.2.2 FPGA Board

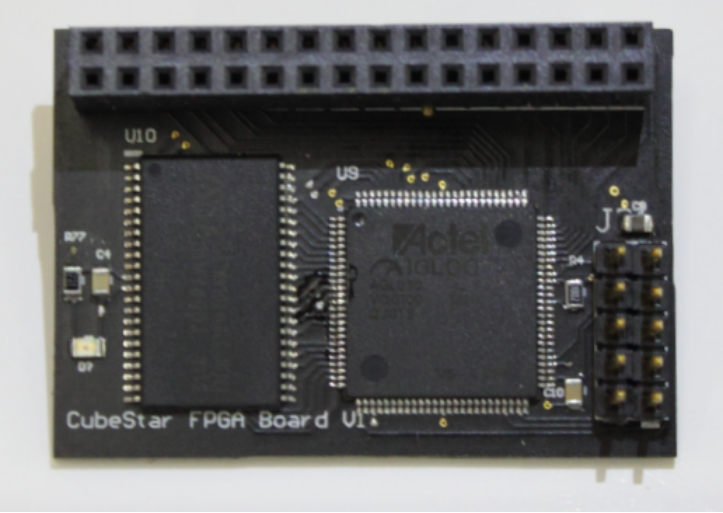


Figure 5.4: The FPGA board, which is the second board in the stack

The FPGA board serves as a buffer between the Melexis image sensor and the processor. It consists of an Actel IGLOO NANO AGLN030 FPGA, a 1024 kB Alliance Semiconductor SRAM module and supporting circuitry.

The FPGA and SRAM modules operate at 3.3 V. No regulators are included on the FPGA board, so it is directly connected to CubeStar's power supply via a power switch on the processor board. The processor board monitors the current consumption of the FPGA board and can cycle the power if it detects an abnormal current draw. A spike in current draw could be the result of a radiation induced latchup in the SRAM.

The Melexis image sensor outputs data at a rate of 11 million bytes per second. Since CubeStar's processor is clocked at 48 MHz it can perform a theoretical maximum of

48 million instructions per second. Four instructions are not enough to read and store each byte of data coming from the image sensor. Therefore, the FPGA and SRAM on the FPGA board act as a buffer, allowing the processor to read the pixel data at a reduced data rate.

Each image from the Melexis image sensor is 512 kB in size, but a 1024 kB SRAM was chosen to allow two images to be stored simultaneously. The current version of the FPGA firmware does not make use of the extra memory, but a firmware upgrade would allow the extra memory to be accessed. By storing an image taken with the lens cap on in the extra memory, dark frame subtraction could be implemented.

As shown in Figure 5.2, the communication between the FPGA board and the image sensor board is one way only. The FPGA receives pixel data over eight parallel pixel data lines from the image sensor board. Whenever the Pixel Clock goes high, it signals to the FPGA that it should read the eight parallel data lines, store the byte of data in SRAM and increment the SRAM address pointer. A counter in the FPGA counts the horizontal syncs. A complete image has been captured when the counter reaches 511, meaning all rows of the image have been stored. At this point the FPGA asserts the ACK line, telling the processor that an image has been captured successfully. Whenever the VSync line goes low, the HSync counter and SRAM address pointers are reset to zero in preparation for the start of a new image.

The FPGA board interfaces to the processor board through eight parallel data lines and four control lines. The flow of data from the FPGA board to the processor board is similar to the flow of data from the image sensor board to the FPGA board. Once an image has been captured and the ACK pin as been asserted by the FPGA, the processor can access the image data stored in SRAM by asserting the Enable and RNW lines and strobing the Data Clock line. Whenever the FPGA senses that the Data Clock line has gone high, it presents a byte of data from the current SRAM address on the eight parallel data lines and increments the SRAM address. Since the processor controls the Data Clock line, it controls the transfer data rate.

A header on the FPGA board allows the FPGA to be reprogrammed.

5.2.3 Processor Board

The processor board contains an Energy Micro Arm Cortex M3 processor, a 1024 kB Alliance Semiconductor SRAM module, current monitoring circuitry, power switches and supporting components. All image processing, star matching and attitude estimation calculations are performed by the processor on the processor board.

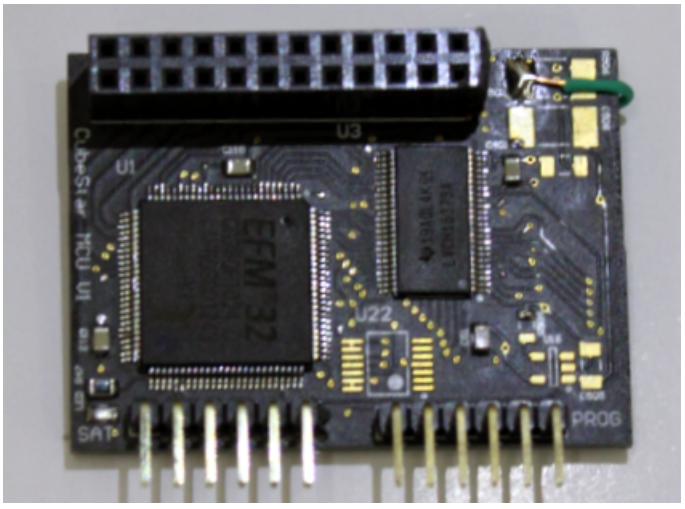


Figure 5.5: The processor board, which is the bottom board in the stack

The EFM32GG280F1024 processor contains 128 kB of RAM and 1024 kB of flash memory. 128 kB of RAM is not enough to store an image, therefore external SRAM was added. The Gecko processor contains an External Bus Interface (EBI) which allows external memory to be added to the Gecko processor. When configured correctly, the external memory can be accessed as if it were internal memory.

The Gecko processor and SRAM module operate at 3.3 V. No regulators are present on the processor board, so it is essential that the board is given a regulated 3.3 V supply. The processor board has two channels for measuring current consumption. The first channel measures the current consumption of the SRAM module and the second channel measures the combined current consumption of the other two boards in the stack. Load switches, controlled by the processor, can cut current to the SRAM module or other boards in the stack if abnormal current consumption is detected. The current monitoring and load switching circuits are similar to those used on CubeComputer [23].

SRAM is vulnerable to radiation induced latchups. A latchup is an unwanted and unexpected low-impedance path between two conductors in a MOSFET circuit. If a latchup occurs between the power rails of an integrated circuit, it will cause excessive current consumption and may permanently damage the device. Ionising radiation, as found in space, is known to cause latchups. Fortunately, latchups can be corrected by power cycling. It is essential that latchups be detected and corrected as early as possible to prevent damage to the device. For this reason the current monitoring circuits are connected to the processor's voltage comparators, instead of the analogue to digital converters. The voltage comparators can trigger an interrupt, shutting off the current

to the affected parts as quickly as possible.

The Gecko processor is clocked at its maximum speed using a 48 MHz crystal.

The processor board has a 24-pin header for connecting to the other boards in the stack and two 6-pin headers. One 6-pin header allows a programmer to be plugged in to reprogram the processor. The other 6-pin header is the external interface to CubeStar. This header will normally be connected to the satellite OBC.

5.3 Software

5.3.1 Image capture

The image capture process includes several steps, as depicted in Figure 5.6. An overview of the process in presented in this section and more details are available in Appendix D.

Before the image capture process can begin, the image sensor must be initialised by setting several of its registers using I2C commands. The output mode, exposure time, frame rate and other options must be set as described in Appendix D. Thereafter, the image sensor can be enabled, also by writing to a specific register.

After being enabled, the image sensor begins outputting image data. The first image output must be discarded as it has an undetermined exposure time. A 70 ms delay prevents the FPGA from storing this invalid image.

Next, the FPGA is set to *write* mode and enabled, by setting its Read-Not-Write (RNW) pin to zero and its Enable pin to one. The FPGA pin will begin monitoring the Hsync, Vsync and PixData lines from the image sensor board and storing the image data in its external SRAM. The processor monitors the ACK pin from the FPGA, which will be raised once a complete image has been captured. Once the ACK pin has gone high, the processor disables the FPGA and the image sensor.

A complete image is now stored in the FPGA's external SRAM. The processor needs to copy the image to its own external SRAM before the image is available for processing. This is achieved by setting the FPGA into *read* mode. Once the FPGA is enabled in read mode, it begins monitoring the processor's Data Clock line. Whenever this line toggles, the FPGA presents the next sequential byte of image data on the eight parallel data lines. The processor strobes the Data Clock line while reading in the image data from the FPGA and storing it in its external SRAM. Once all the image bytes have

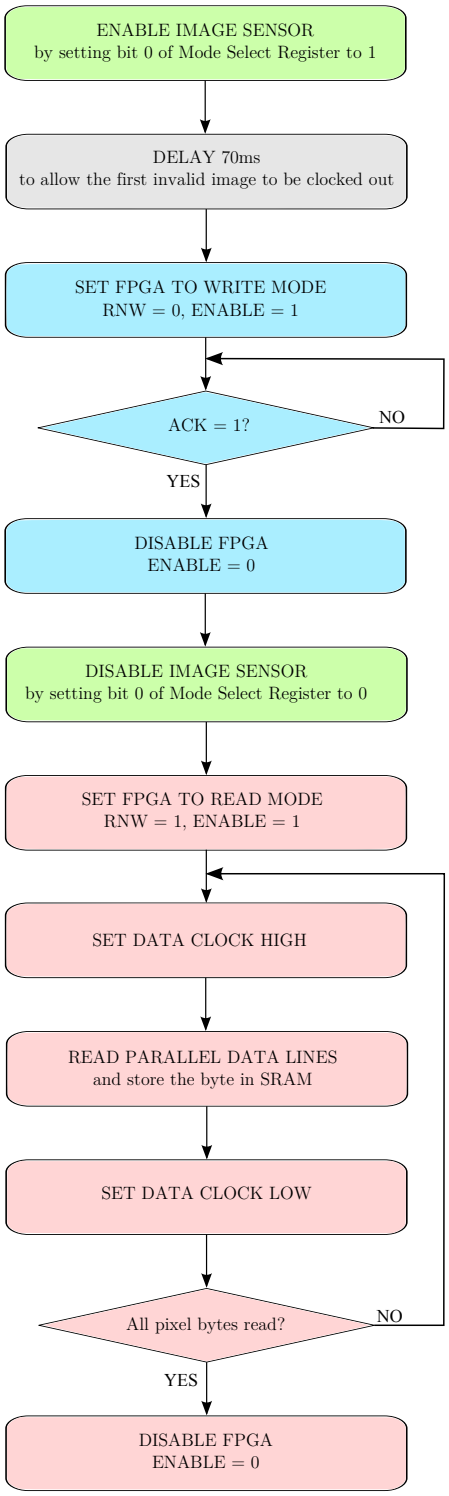


Figure 5.6: The process of capturing an image, as seen by the processor. Blocks with the same colour are executed in the same function.

been transferred, the processor disables the FPGA and the image capture process is complete.

The image is now stored in the processor's external SRAM. The EBI allows random read and write access to the image data, which is essential for the the image processing algorithms.

5.3.2 Star Catalogue Implementation

The star list is implemented as a set of four constant arrays, each consisting of 405 entries. These arrays contain the names and the x, y and z components of the inertial coordinates of each star. The *const* modifier is essential before the declaration of the arrays as it ensures that the arrays are stored in flash memory and not copied over to internal RAM. The complete star list is 5.54 kB in size.

The inter-star distance list is implemented as three constant arrays each consisting of 11484 entries. Two of the arrays contain star names and the third contains inter-star angular distances. The information can be understood by reading index n of each array, which will give two star names and their angular, inter-star distance. The complete inter-star distance list is 89.7 kB in size.

5.3.3 Star Detection

There are several functions involved in detecting possible stars in the raw images. The process is initiated by calling the function *detectStars*, which requires four arguments: a pointer to the raw image in memory, an empty list of centroids, a brightness threshold value and a minimum pixel count value.

detectStars searches systematically through the raw image by checking every third pixel against the threshold value. If a pixel is found that is above the threshold, the recursive function *grow* is called to find all pixels belonging to the detected blob. The recursive nature of the *grow* function can cause problems if left unchecked. The Gecko processor's stack will overflow if the recursion reaches too many levels deep. A stack overflow causes the processor to lock up, which can only be resolved with a reset. In order to prevent run-away recursion, which can occur when large bright objects enter the FOV, a maximum of 60 pixels are allowed per detected area. If an area contains more than 60 pixels which are above the threshold, the area is marked as an invalid centroid and is ignored. If an image contains more than 50 invalid centroids, the search is stopped.

Whenever the function *grow* is called, it returns with a list of pixels belonging to the detected blob. The function *centroid* calculates the centroid of the list of pixels and appends it to a list of centroids. The list of centroids is restricted to a maximum length of 20, for reasons described in Section 5.3.4. During the centroid calculation, a function called *correctDistortion* corrects for the lens distortion.

Experiments under the night sky with the engineering model of CubeStar suggest that a threshold of six and a minimum pixel count of six work well for semi-urban skies.

5.3.4 Star Matching

Star matching is performed by the function *starTrack*. *starTrack* requires several parameters, including a pointer to a list of detected centroids, a pointer to an empty list of vectors and a pointer to an empty list of weights. The function *starTrack* will attempt to match each detected centroid to a star from the onboard database. If the algorithms are successful, the list of vectors will be populated with the matched inertial vectors, and the list of weights will signify the confidence in each match.

The first thing starTrack does is check a flag in flash memory which keeps track of the current state of the star tracker. A "2" signifies that the star tracker is currently lost in space. A "1" signifies that the star tracker is operating in assisted tracking mode, and a "0" signifies that the star tracker is currently in tracking mode. This flag is set by the result of the attitude determination algorithms, as described in Section 5.3.5.

If the star tracker is in lost in space mode, the complete onboard star catalogue is used for star matching. However, if the star tracker is in tracking or assisted match mode a good attitude estimate is already available from the previous iteration. This attitude estimate is used to generate a reduced catalogue in real time to be used for the star matching. The reduced catalogue is generated by the function *generateReducedCatalogue*, which requires a FOV as its argument. The FOV argument influences which stars should be included in the reduced star catalogue. If the satellite has a good estimate of its current attitude, the FOV can be just larger than the star tracker's FOV. If the current estimate is poor, the FOV should be set proportionately larger.

Next, the inter-centroid distances are calculated and the voting rounds are performed. Searches through the star catalogue are performed using binary search. Several optimisations were made to speed up this process, including:

- The inter-centroid distances are calculated only once, initially, and the results are stored in a matrix. The matrix is symmetrical, so only half of it needs to be

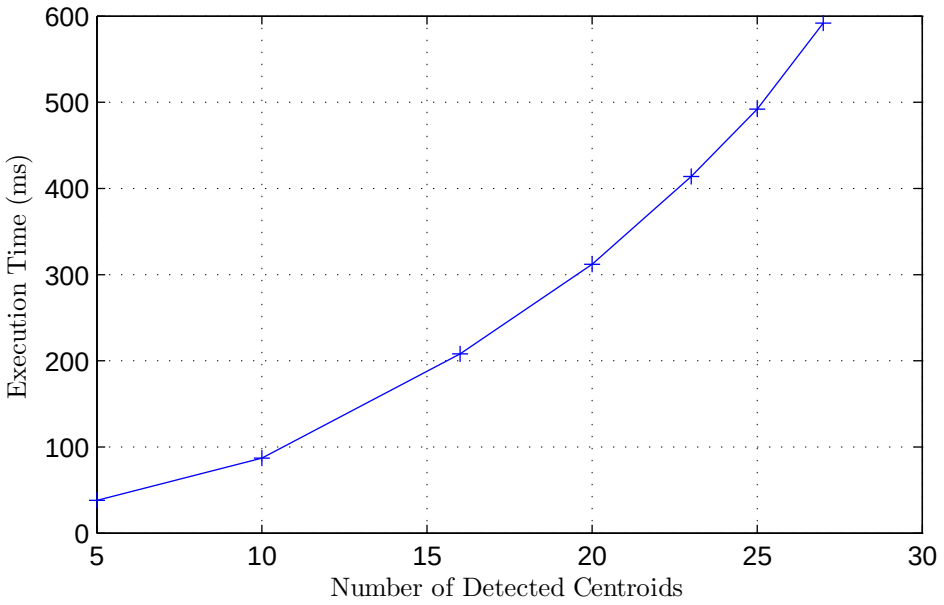


Figure 5.7: The execution time of the star detection and matching algorithms for increasing numbers of centroids

calculated.

- Dynamic memory structures were replaced with preassigned arrays. Preassigned arrays are less flexible and less memory efficient, but they were found to be approximately an order of magnitude faster to manipulate.
- The star catalogue entries are stored in Cartesian coordinates instead of spherical coordinates.

Finally, the lists of matched vectors and weights are populated. During the matching process, each matched star is checked against all its neighbouring matched stars to ensure that the inter-star distances are as expected (See Section 4.3). The weights list shows the result of this check for every star. A star with a high match confidence should have a weight which is very close to the total number of detected stars, meaning that all distances from that star to its neighbours are correct. The maximum weight that a star can achieve is one less than the total number of detected stars. Good star images will produce a list of weights which are all similar and all close to the maximum. False stars will increase the total number of detected centroids without increasing the weights.

The execution time of the star detection and star matching algorithms are dependent on the number of centroids in the image, as shown in Figure 5.7. The slope of the

figure suggests that the relationship is exponential, which is expected. As explained in Section 5.3.6, all the algorithms required to go from raw image to attitude estimate must complete within 500 ms. To allow enough time for the attitude determination to complete and to give some margin, a maximum of 20 centroids per image is set. With this maximum in place, the star detection and matching algorithms will take a maximum of 320 ms.

5.3.5 Attitude Determination

The QUEST algorithm was implemented by porting a MATLAB implementation to C. The Gaussian elimination stage of the algorithm was implemented using a function written by Paul Bourke [50]. The QUEST algorithm takes less than 10 ms to execute on the Gecko processor.

A function to convert the quaternion result to a DCM was also implemented. The function makes use of Equation 5.3.1.

$$\mathbf{C} = (q_0^2 - \mathbf{q}^T\mathbf{q}\mathbf{I}) + 2\mathbf{q}\mathbf{q}^T - 2q_0\mathbf{q}_x \tag{5.3.1}$$

where

$$\begin{aligned}
\mathbf{C} &= \text{is the Direction Cosine Matrix} \\
\mathbf{q} &= \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^T \\
\mathbf{q}_x &= \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix}
\end{aligned}$$

Both the quaternion and DCM results are stored in flash memory for use by the matching algorithms during the next iteration.

5.3.6 Timing

CubeStar has two important timing requirements. Firstly, it must produce a new attitude estimate every second. Secondly, the outputted attitude estimate should not be more than a second old. These tight timing requirements necessitate the parallelisation of tasks. Capturing a star image and transferring it to the processor typically takes about 900 ms. This does not leave enough time for the star matching and attitude

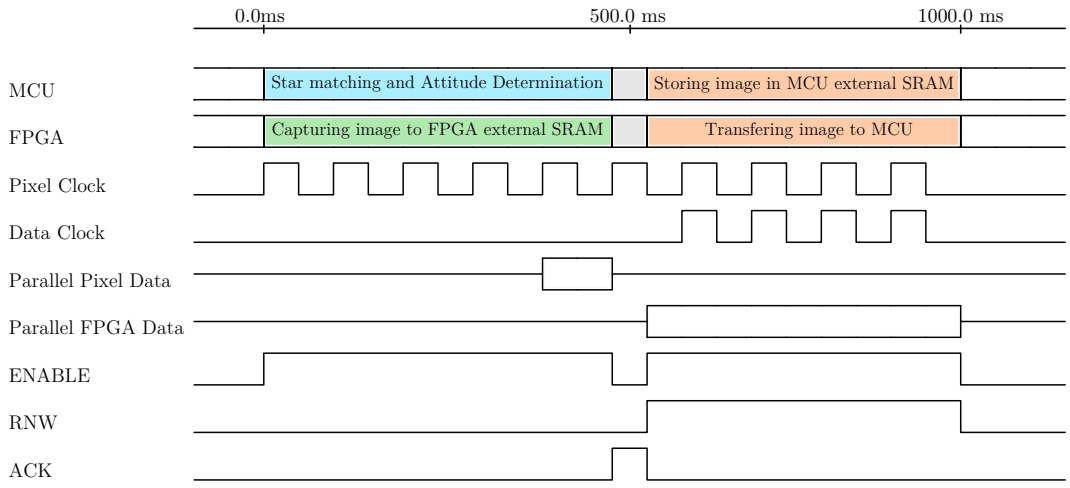


Figure 5.8: Timing diagram of the events that take place during each iteration.

determination algorithms to execute. To solve this problem, the star tracker's work is spread over two seconds, while maintaining an update rate of 1 Hz. This is achieved by capturing a new image while simultaneously processing the previous image.

The timing of tasks executed during each second, or iteration, is shown in Figure 5.8. Note that the clock signals are not shown to scale and that the timing of all signals is approximate. The first two lines, labelled *MCU* and *FPGA*, describe what tasks the MCU and FPGA are busy with at each time instant. The figure shows that the processor executes the star matching and attitude determination algorithms on the previous image, while the FPGA simultaneously reads and stores a new image. The time available for the image sensor to expose and readout is approximately equal to 500 ms. Shorter exposures are allowed, but longer ones are not. The time available for the algorithms to complete is also approximately 500 ms. It does not matter whether the algorithms or image capture completes first, as long as neither of them take longer than 500 ms.

The second 500 ms of each iteration is required for image transfer between the FPGA's SRAM and the processor's SRAM. During image transfer, both the FPGA and the processor are occupied. This is a bottle neck which wastes significant time, however, it is a result of the decision made in Section 3.3. The image transfer time is directly proportional to the processor's clock speed, so a faster processor would reduce the image transfer time.

The image sensor's Pixel Clock is always active, however, it is only used during the transfer of data from the image sensor to the FPGA's SRAM. The Pixel Clock runs at 10 MHz, resulting in a data transfer rate of 10 MB/s. The Data Clock is produced by

the processor during the transfer of data between the FPGA's external SRAM and the processor's external SRAM. The speed of the Data Clock is restricted to approximately 1 MHz for the Gecko processor, resulting in a data rate of approximately 1 MB/s.

At the end of every iteration, once the image capturing, algorithms and image transfer have completed, the processor performs a software reset. The software reset clears the processor's stack and internal RAM. The only information that is preserved between iterations is the most recently taken image and the most recent successfully calculated sensor attitude (in quaternions and DCM form). The reset procedure ensures that the processor is in the exact same state at the start of every iteration. In addition to aiding troubleshooting, performing a software reset at the end of every iteration prevents undetected memory leaks from eventually crashing the system. The processor reset takes only a few milliseconds to complete.

Unfortunately, while the parallelisation ensures an update rate of 1 Hz, the age of the outputted attitude estimates is 1.5 seconds. This may cause problems for the ADCS if it is running at a higher frequency. The SUNSAT star tracker suffers from the same problem. It may be possible to optimise the tracking algorithm and overall timing to achieve attitude estimates which are less than a second old. Alternatively, a hardware modification implementing one of the other options discussed in Section 3.3 could be considered.

5.4 Calibration

The calibration procedure is essential for obtaining good performance from a star tracker. The calibration procedure procedure consists of the following four steps:

1. Focus the optics to achieve the desired star point spread function
2. Determine the resultant focal length
3. Determine the principal point of the optics
4. Determine the distortion coefficients of the optics

The optics should be focussed at infinity and then slightly defocussed to spread out the star light over several pixels. Ideally, a collimated light source would be used to aid focussing of the optics. Unfortunately, no collimated light source is available in the ESL. Therefore, focus is achieved by taking images of real stars and adjusting the focus manually until the stars appear as blobs covering approximately 9-30 pixels.

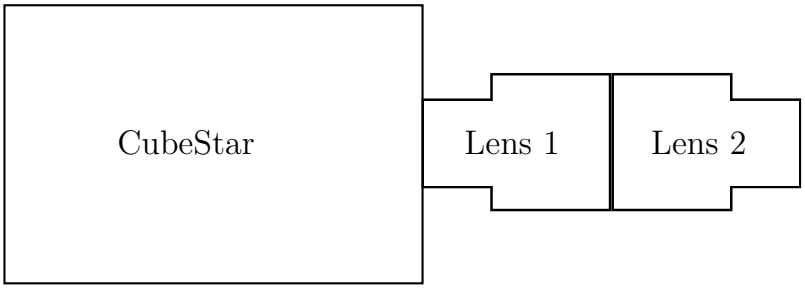


Figure 5.9: Lens setup to determine the principal point

To determine the resultant focal length of the optics, an image of a known constellation is taken. The distances between the stars in the image can then be compared to the known distances between the stars in the constellation to determine the optics' focal length. It is desirable to use stars which lie near the middle of the image where the lens distortion is at a minimum. Equation 5.4.1 calculates the focal length of the optics that produced an image of a pair of stars with true angular separation of $D_{radians}$ and measured separation on the image plane of D_{mm}. The true angular separation of a pair of stars can be found in the star catalogue. Equation 5.4.1 was applied to several star pairs and the results were averaged to determine that the CubeStar engineering model has a focal length of 6.28 mm.

$$f_{mm} = \frac{D_{mm}}{D_{radians}} \tag{5.4.1}$$

The principal point is the point where the optical axis intersects the image plane. It can be found by repeatedly imaging a dot on the wall while rotating the sensor around the optical axis. However, this process was found to be very tedious. An alternative method was devised. The alternative method involves fastening a second, identical lens to the front of CubeStar's lens. The lenses are connected facing one another, as shown in Figure 5.9.

This unusual lens setup produces an image of a circle of light which fades in intensity from the centre of the circle outwards, as shown in Figure 5.9. The centroid of this circle of light is the principal point. The principal point is required by the distortion correction (Section 4.1.4) and image plane to unit vector (Section 4.1.5) equations.

The final step of the calibration procedure is determining the distortion coefficients of the lens. The freely available MATLAB Camera Calibration Toolbox can calculate the focal length, principal point and distortion coefficients of a lens [51]. All the toolbox requires is a sequence of images taken by the unknown lens of a checker board pattern

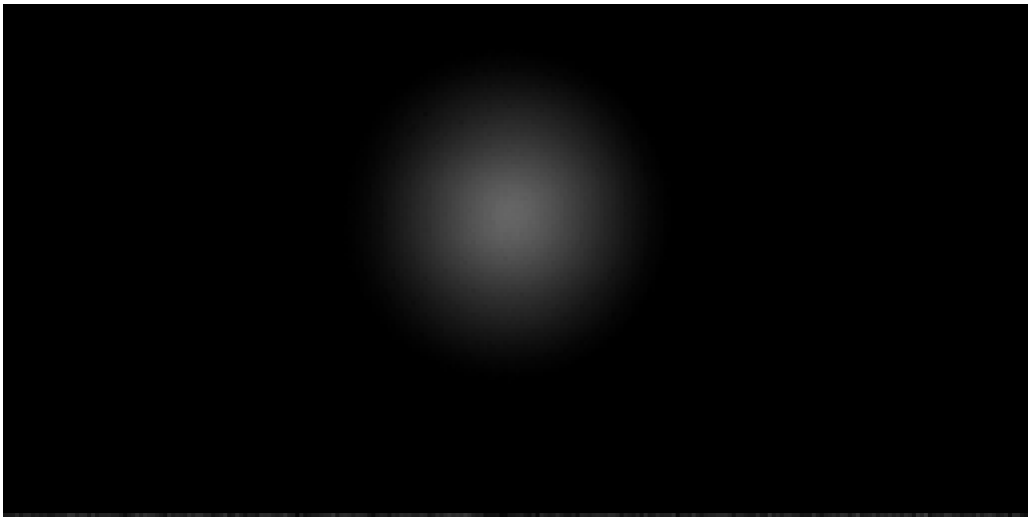


Figure 5.10: Image produced by the lens setup depicted in Figure 5.9

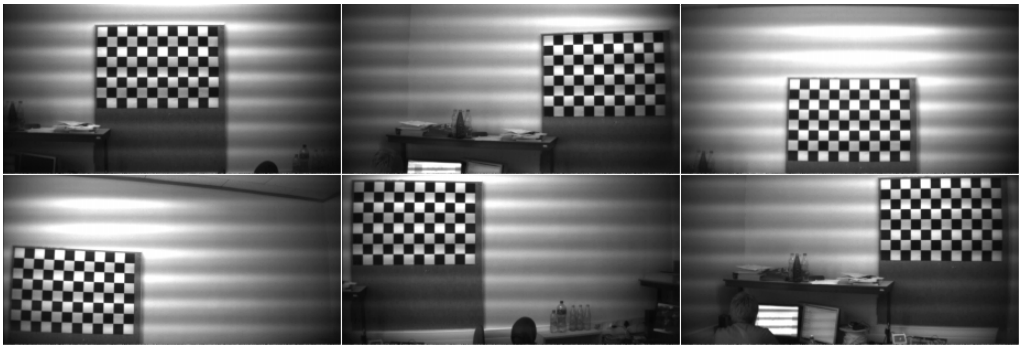


Figure 5.11: A subset of the calibration images input into the MATLAB Camera Calibration Toolbox. The horizontal streaking is a result of the rolling shutter and high frequency flickering of flourecent lighting.

with known dimensions. The toolbox requires very little user input and outputs the lens parameters with estimated error margins.

It is essential that the checker board fills as much of each image as possible. Unfortunately, this is difficult to achieve since CubeStar's lens has a wide FOV and is focussed close it infinity. The checker board must be moved several meters away from the camera before it comes into focus. At this distance, the checker board needs to be very large to fill the FOV. A compromise was made by building a 1 x 1.5 m checker board and moving it around within the FOV between images. The final set of images includes at least one image with the checker board in every part of the FOV. A subset of the images is shown in Figure 5.11.

In order for the toolbox to accurately calculate the focal length of the lens, the images of the checker board should be from various angles. However, since the focal length of

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 1147.50815   1147.40417 ] ± [ 39.98968   39.61536 ]
Principal point:       cc = [ 539.00635   175.53211 ] ± [ 13.25120   15.32062 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000 ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.32358   0.12456   0.00414   -0.00624   0.00000 ] ± [ 0.01004   0.03556   0.00163   0.00085   0.00000 ]
Pixel error:          err = [ 0.20336   0.14610 ]
```

Figure 5.12: The output from the MATLAB Camera Calibration Toolbox

the lens is already known from earlier experiments, the toolbox's focal length estimate is unimportant. Therefore, the checker board images were not taken from drastically different angles. The output of the toolbox is shown in Figure 5.12. The output shows that the focal length and principal point uncertainties are high, but the distortion coefficient uncertainties are very low.

The outputted distortion coefficients are in the form $[K_1\ K_2\ P_1\ P_2\ P_3]$. The calculated distortion coefficients (with inverted signs) are used in Equations 4.1.3-4.1.4 from Section 4.1.4 to undistort the image.

In conclusion, CubeStar's performance specifications can be met using the calibration procedures described in this section. Some performance increase may be possible with more complex calibration procedures, however, the described procedures are simple to implement using the facilities available in the ESL.

Chapter 6

Testing and Results

This chapter presents the results of tests performed on the complete engineering model of CubeStar. Subsystem tests are described in previous chapters.

6.1 Accuracy Test

There are many factors which influence the accuracy of a star tracker, the most significant of which are listed below:

- FOV of the optics
- Image sensor resolution
- Image sensor noise levels
- Accuracy of the optical distortion correction

Some factors, such as the FOV size, image sensor resolution and calibration procedure limit the accuracy in predictable ways. Other factors, such as the number and location of stars in the FOV, image sensor noise and atmospheric distortion have far less predictable impacts on the accuracy. Therefore, it is essential that the star tracker be tested in as realistic conditions as possible. This can best be achieved by testing the star tracker under the real night sky.

The more accurate a sensor is, the more difficult it becomes to reliably test its accuracy. Any output from the sensor includes errors caused by the test equipment, which can not be distinguished from errors intrinsic to the sensor. A star tracker is particularly difficult to test as there is no accurate way of pointing the sensor at a known location in

the sky. A more accurate star tracker could be used to determine CubeStar's accuracy by pointing both star trackers at the same location in the sky and comparing their output. However, a more accurate star tracker was not available at the time for testing purposes.

An alternative method of measuring a star tracker's accuracy is described in [52]. The method involves pointing the star tracker towards zenith and fixing it relative to the Earth. The attitude estimate output of the star tracker is logged over several minutes while the stars move through its FOV due to the rotation of the earth. It is convenient to convert the star tracker's attitude output into right ascension, declination and rotation. The declination and rotation output should stay constant, while the right ascension drifts at the sidereal rate. The star tracker's accuracy can be estimated by the statistical fluctuations of its attitude output.

Two versions of this experiment were performed using CubeStar mounted to a Mead tracking telescope on the roof of the engineering building in the town of Stellenbosch. The first iteration of the experiment involved pointing the star tracker at a location in the sky and turning off the telescope's tracking function so that the star tracker remained fixed relative to the Earth. The output of the star tracker was logged over several minutes and the results can be seen in Figures 6.1-6.3. CubeStar would start in lost-in-space mode and quickly transition to tracking mode for the duration of the test.

Figure 6.1 shows the number of detected centroids and the number of matched stars during the test. These two values follow one another exactly, meaning that every centroid was successfully matched to a star from the onboard database during every iteration. The number of detected centroids fluctuated between 8 and 16 stars, despite the fact that the same stars were likely in the FOV for the duration of the test. This fluctuation is due to atmospheric effects causing the stars to "twinkle" and due to light pollution from the town which causes more stars than expected to be near the detection threshold.

Figure 6.2 shows CubeStar's attitude output during the earth-fixed test. CubeStar outputs its attitude estimates as quaternions, which were converted to celestial coordinates for ease of interpretation. The right ascension was expected to increase linearly over time and declination and rotation were expected to remain constant. This behaviour can be seen in CubeStar's output, with superimposed noise. The random fluctuations around the expected outputs can be used as a measure of its accuracy.

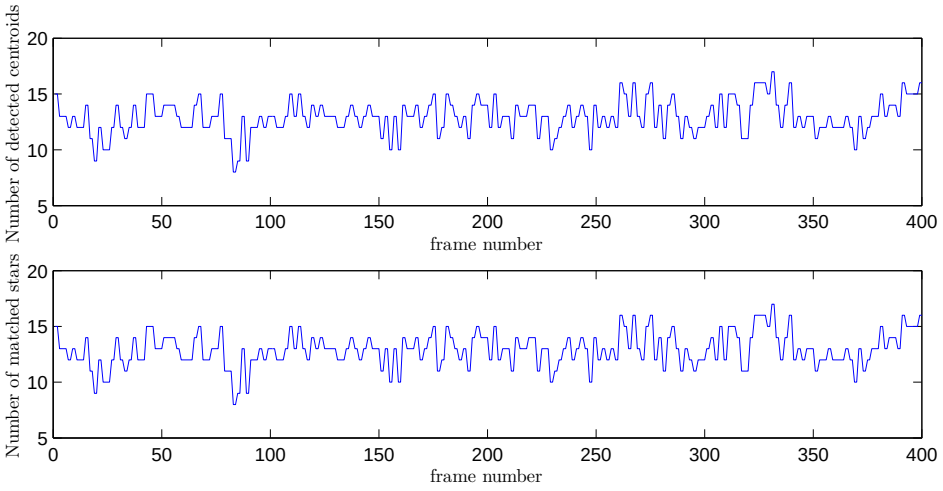


Figure 6.1: The number of detected centroids and matched stars during the earth-fixed, night sky test

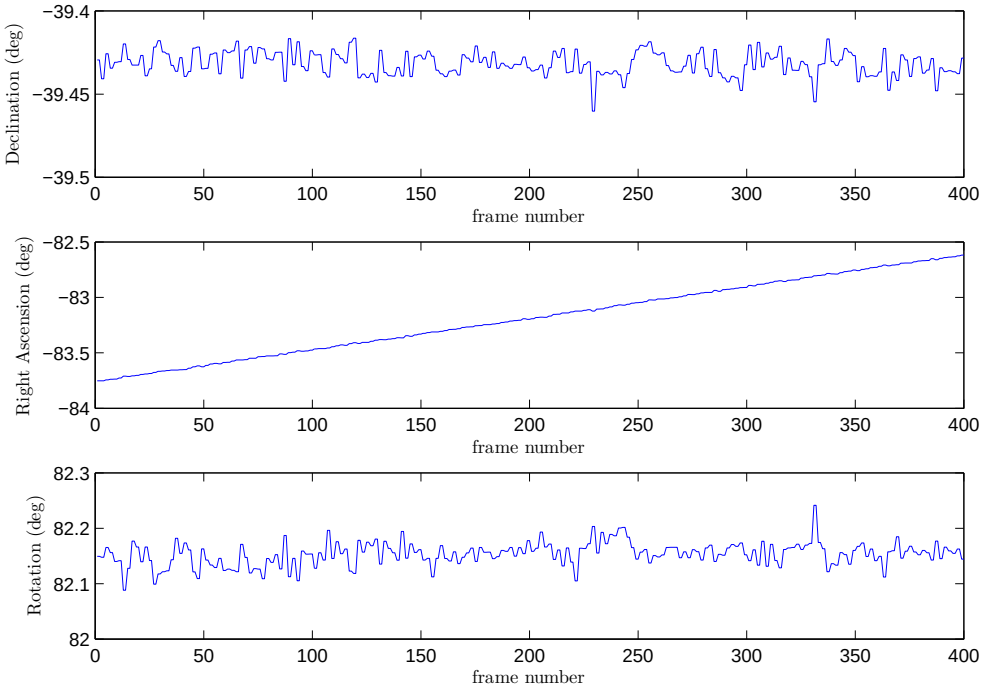


Figure 6.2: The attitude output of CubeStar during the earth-fixed test. The rotation and declination were expected to stay constant while the right ascension changed linearly.

Axis	3σ accuracy (deg)	1σ accuracy (deg)
Cross-Boresight (RA)	0.0154	0.0051
Cross-Boresight (DEC)	0.0215	0.0072
Around Boresight (ROT)	0.0608	0.0203

Table 6.1: CubeStar's accuracy as determined by the earth-fixed test

Figure 6.3 shows an error distribution for the attitude outputs during the earth-fixed test. To generate these error distributions straight lines were fitted to the attitude output plots. The straight lines were used as the true attitude with which to compare the raw CubeStar output. The error distributions have a roughly Gaussian shape, which proves that any errors in the attitude output are due to noise sources and not errors in the algorithms.

The error distributions show that, for this test, CubeStar achieved the accuracies presented in Table 6.1.

Similar accuracies were achieved when this test was repeated on different nights and on different parts of the sky.

The second version of the accuracy test involved activating the tracking function of the telescope so that CubeStar would remain pointing at a fixed location on the celestial sphere. The expected output was a constant right ascension and declination and a changing rotation. The results can be seen in Figure 6.4.

The results are close to the expected output, however, the right ascension and declination did not remain constant but drifted slightly during the test. This is likely due to the imperfect tracking of the telescope. The noise on the output is also higher because it is a combination of CubeStar's noise and the telescope drive's noise.

6.2 Slew Test

It is important for a star tracker to be able to track the stars, even when moving at certain angular rates. A satellite that is three axis stabilised and remains nadir pointing during its orbit at a height of 500km will be pitching continuously at a rate of 0.0635 deg/s. This is the minumum tracking speed required.

In order to test CubeStar's ability to track while slewing, CubeStar was attached to a Meade tracking telescope. The telescope has the ability to slew at four different rates: 8, 2, 0.1333 and 0.00833 deg/s. While slewing at 2 deg/s, CubeStar cannot successfully match any stars as all stars appear as streaks. However, tests performed

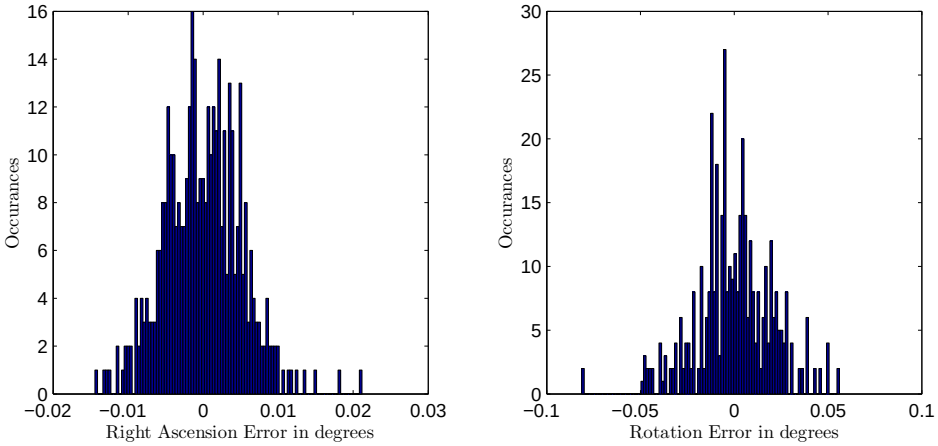


Figure 6.3: Error distributions of the CubeStar's attitude output during the earth-fixed test

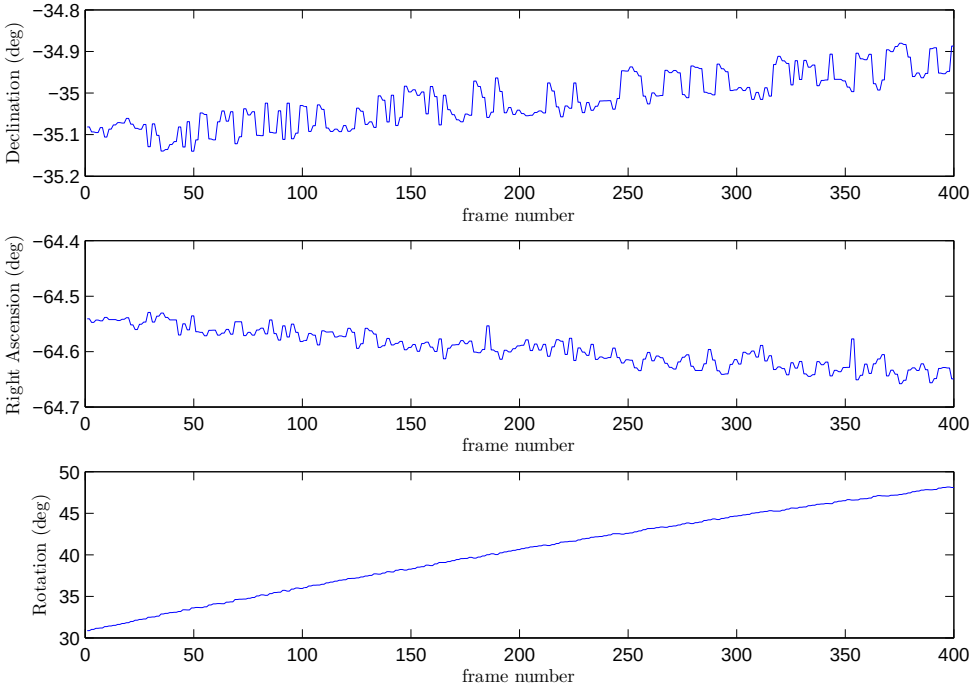


Figure 6.4: CubeStar's attitude output during the celestial-fixed test. The right ascension and declination were expected to stay constant while the rotation changed.

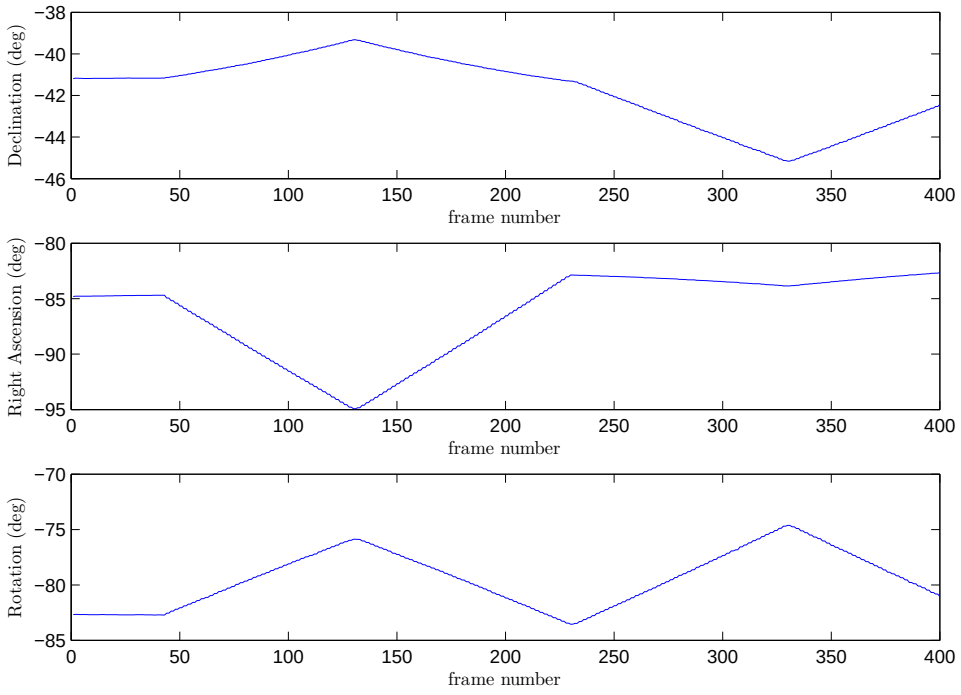


Figure 6.5: CubeStar's attitude output while slewing around different axes at 0.1333 deg/s

while slewing at 0.1333 deg/s were successful. At 0.1333 deg/s, CubeStar's tracking algorithm performs as expected and most detected stars are matched. Figure 6.5 shows CubeStar's attitude output as the telescope was slewed around different axes.

The slew test confirms that CubeStar will be able to track the stars while on a nadir pointing satellite. An upper limit on the tracking speed of 0.3 deg/s is imposed by the current star tracking algorithm, as explained in Section 4.5

6.3 Power Consumption

CubeStar's instantaneous power consumption depends on the task it is currently executing. To determine the instantaneous current consumption, CubeStar's power supply was monitored with an INA139 current shunt monitor, while it performed a typical star matching iteration. The results show that CubeStar goes through four distinct power consumption modes, as shown in Figure 6.6.

CubeStar begins in idle mode (Mode 1), where the processor is waiting for a new command and the imager is in low power/disabled mode. Next, the imager is switched on while the first, invalid image is clocked out (Mode 2). The largest power consumption

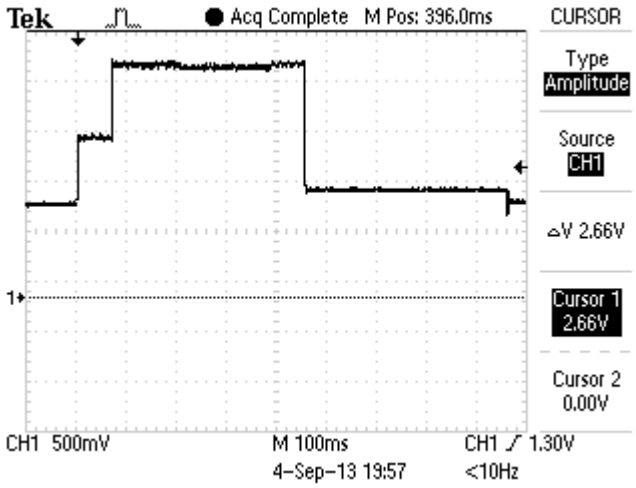


Figure 6.6: CubeStar's measured current consumption during a typical iteration of 1000 ms

Mode	Current Consumption (mA)	Time Period (ms)
1	71.2	140
2	121.2	68
3	177.3	388
4	81.81	404

Table 6.2: CubeStar's current consumption in different modes

occurs next, while the image sensor is taking a new image and the processor is processing the previous image (Mode 3). The fourth power consumption mode occurs while the image data is being transferred from the FPGA's SRAM to the processor's SRAM (Mode 4). The image sensor is disabled during the image transfer. Once the image transfer is complete, CubeStar goes back into idle mode.

Table 6.2 lists the current consumed during each mode, and the time spent in each mode during a typical 1000 ms iteration.

CubeStar operates off a single 3.3 V supply, so it is simple to determine its power consumption from its current consumption. From Table 6.2 it is evident that the maximum power consumption required by CubeStar is 0.585 W during mode 2. CubeStar's average power consumption over one iteration is given by Equation 6.3.1.

$$\text{Power}_{avg} = \sum_{n=1}^{4} P_n T_n = 396.2 \text{ mW} \tag{6.3.1}$$

where

P_{avg} = The average power in mW over a 1000ms iteration

P_n = The power consumed during mode n

T_n = The time spent in mode n in seconds

Without the two onboard LED's, which would be removed for flight models, the average power consumption would be closer to **350mW**.

More power could be saved by turning the camera off completely during the idle and image transfer modes, instead of disabling it. However, the current CubeStar hardware does not allow this option.

Chapter 7

Conclusion and Recommendations

This chapter presents a brief summary of the results of each section, draws conclusions with regards to the original goals of the project and makes recommendations for the future of CubeStar.

7.1 Summary and Conclusions

Chapter 1 gave a brief introduction to CubeSats, CubeSat ADCS performance and star trackers. It discussed the advantages of adding a star tracker to current CubeSat attitude determination and control systems and outlined the basic goals of the CubeStar project. The relevance of this research topic with respect to the CubeSat community was emphasized by the description of four other nano star tracker projects, most of which are still ongoing.

Chapter 3 described the design process of CubeStar. Existing subsystems would be reused from CubeSense and CubeComputer in order to develop a star tracker within two years. A suitable image sensor with enough sensitivity was found in the automotive industry. The image sensor costs under R500, requires relatively low power and has an interface similar to the CubeSense cameras. A high quality, commercial lens with a suitable FOV was found for under R400. Together, the image sensor and lens allow cubeStar to detect stars with magnitudes down to 3.8 over a 51 x2 7 degree FOV. This was proven to be sufficient for keeping at least three stars within the FOV over 99.9% of the celestail sphere.

Chapter 4 described the algorithms involved in matching stars and estimating attitude. The image plane search, region growing and centroiding algorithms were reused from SUNSAT's star tracker for detecting and extracting stars from the raw star images. It

was proven that these algorithms were efficient and could achieve sub-pixel accuracy by testing them on simulated and real sky images. CubeStar's star matching algorithm, called the Geometric Voting Algorithm, was described in detail. The matching algorithm was proven to provide a lost-in-space match over 93% of the celestial sphere and an assisted match over 98.5% of the celestail sphere. A tracking algorithm with coverage over 100% of the celestial sphere was also described.

The QUEST algorithm was chosen as CubeStar's attitude estimation algorithm due to its long space heritage. The complete set of algorithms was tested by inputting a simulated sky image, generated with a known attitude, and comparing this known attitude to the output of the QUEST algorithm. The algorithms were proven to provide an accuracy of better than 0.01 degrees in the absence of image noise or lens distortion.

Chapter 5 described the implementation of CubeStar. The hardware was implemented as three separate 3 x 4.5 cm circuit boards stacked behind one another to make optimal use of the limited space onboard CubeSats. An engineering model was completed and simple calibration procedures were developed. CubeStar weighs less than 90 g without a baffle or case and takes up less than 0.25 U volume, making it one of the smallest star trackers in existence. CubeStar achieves a 1 Hz update rate by performing the image capture and processing in parallel. However, the current hardware design wastes a significant amount of time transfering images between the FPGA and processor. This has a negative impact on the age of the outputted attitude data which may cause problems for accurate ADCS performance. A solution to the problem, involving the use of a single shared SRAM, was proposed for implementation on the next iteration of CubeStar.

Chapter 6 described the tests performed on the engineering model of CubeStar. CubeStar was taken outside for real night sky tests to determine its accuracy and tracking performance. The tests verified the operation of CubeStar in all modes and proved that CubeStar could achieve a 1σ accuracy of 0.0072 degrees across the boresight and 0.0203 degrees around the boresight. A slew test confirmed that CubeStar will operate correctly on a nadir pointing satellite and can handle slew rates up to 0.1333 deg/s. Slew rates up to 0.3 deg/s should be possible with the current firmware.

Table 7.1 lists the specifications of CubeStar V1. These specification can be compared with the original goals of the project. CubeStar's volume is under 0.5U, its average power consumption is under 0.5W and it achieves the desired accuracy of 0.01 degrees. Table 7.2 gives an approximate component cost breakdown. Considering commercial star trackers cost upwards of R100 000, a component cost of R1970 qualifies CubeStar

Specification	Value	Units
Weight	<90	g
Dimensions	46 x 33 x 70	mm
Accuracy (cross bore)	better than 0.01	deg RMS
Accuracy (roll)	better than 0.03	deg RMS
Power (avg/peak)	350/550	mW
Operating Voltage	3.3	V
Data Interface	I2C/UART	-

Table 7.1: CubeStar V1 Specifications (without enclosure or baffle)

Component	Cost (ZAR)
Image Sensor	410
Lens	380
Gecko Processor	100
FPGA	180
PCBs	400
Other	500
TOTAL	1970

Table 7.2: Approximate component cost breakdown

as a low cost star tracker. Therefore, all specifications have been met or exceeded. Most importantly, however, the CubeStar project has successfully delivered a working nano star tracker in under two years. A second generation CubeStar is scheduled to fly onboard ZA-AeroSat in 2015 as part of Stellenbosch University's contribution to the QB50 Project.

7.2 Recommendations and Future Work

This section discusses potential improvements to the CubeStar hardware and software and lists some relevant future research topics.

CubeStar's hardware design is largely a result of trying to reuse as many subsystems from CubeSense and CubeComputer as possible. This was necessary in order to have a working star tracker after two years. However, if a second generation CubeStar is going to be designed, the following hardware changes should be made:

- Implement Option 2 from Section 3.3, which uses a single shared SRAM. This will get rid of the 500 ms image transfer time and allow attitude data which is less than a second old to be output. The single SRAM could be replaced with a radiation hardened SRAM for extra robustness.

Figure 7.1: A rendering of CubeStar on ZA-AeroSat, which is expected to be launched in 2015

- Include mounting holes on the PCBs as the slotted enclosure system is ineffective.
- Include a processor controlled power switch for the image sensor. This will allow the camera to be turned off between images, reducing power consumption.
- Include a buffer on the satellite interface I2C lines.

Implementing the shared SRAM layout will have additional advantages. If a single redesigned FPGA/SRAM board is included in the stack, the image capture and processing will have to happen sequentially. This will allow an update rate of 1 Hz, which is acceptable for most CubeSats. However, if two of the redesigned FPGA/SRAM boards are included in the stack, an update rate of 2 Hz will be possible. This is achieved by saving an image to the SRAM on board one while simultaneously processing the image on board two and vice versa. Either one or two FPGA/SRAM boards could be included in the stack without requiring hardware changes.

CubeStar's firmware is capable of performing all tasks required of a star tracker. However, the following additions to the firmware will increase CubeStar's autonomy and performance:

- Set the star detection threshold automatically based on the average frame brightness information from the image sensor.
- Develop a better set of rules for transitioning between lost-in-space, assisted match and tracking modes.
- Include stars down to 4th magnitude in the onboard catalogue as CubeStar will likely be able to detect these while in space.
- Include more statistics on CubeStar's performance in the telemetry for help during commissioning. This is important as most CubeSats do not have enough telemetry bandwidth to downlink a full resolution star image for analysis.

Once the recommended hardware and software changes have been implemented, more advanced features can be added to CubeStar. These features still require substantial research before they can be implemented:

- A moon tracking mode. A crescent moon should allow 3-axis attitude determination.
- A stellar gyro mode. This will provide the same information as a three-axis gyro by tracking the movement of stars between frames.
- A planet tracking mode to be used for deep space navigation.
- A combined sun and star camera. This could potentially be implemented by placing an electrically controlled, optical filter (such as a liquid crystal display) in front of the lens.

List of References

[1] EE Publishers (Pty) Ltd, "SA student satellite ready for launch." [Online]. Available: http://eepublishers.co.za/article/sa-student-satellite-ready-for-launch.html

[2] University of Colorado Boulder, "CSSWE," 2013. [Online]. Available: http://lasp.colorado.edu/home/csswe/

[3] W. Larson and J. Wertz, Eds., *Space Mission Analysis and Design*, 3rd ed. Microcosm, 1999.

[4] R. Munakata, "CubeSat Design Specification Rev. 12," California Polytechnic State University, Tech. Rep., 2009.

[5] Clyde Space, "Some useful information about CubeSats," 2008. [Online]. Available: http://www.clyde-space.com/cubesat/som_useful_info_about_cubesats

[6] D. T. Gerhardt and S. E. Palo, "Passive Magnetic Attitude Control for CubeSat Spacecraft Conference on Small Satellites," in *24th Annual AIAA/USU Conference on Small Satellites*, 2010.

[7] A. Sofyali, "Magnetic Attitude Control of Small Satellites : A Survey of Applications and A Domestic Example," in *8th IAA Symposium*, 2011.

[8] C. Grant, D. D. Kekez, and R. E. Zee, "Canadian advanced nanospace experiment 2: on-orbit experiences with a three-kilogram satellite," in *22st Annual AIAA/USU Conference on Small Satellites*, 2008.

[9] Surrey Satellite Technology Ltd, "Procyon Star Tracker." [Online]. Available: http://www.sstl.co.uk/Products/Subsystems/Flying-Soon/Procyon-Star-Tracker

[10] SunSpace, "SunSpace - Innovative Satellite Solutions." [Online]. Available: http://www.sunspace.co.za/home/

[11] R. Zenick and T. J. Mcguire, "Lightweight , Low-Power Coarse Star Tracker," in *17th annual AIAA/USU Conference on Small Satellites*, 2003, pp. 1–8.

[12] NASA, "Fast, Affordable, Science and Technology SATellite (FASTSAT) Minisatellite," National Aeronautics and Space Administration, Tech. Rep., 2010. [Online]. Available: http://www.nasa.gov/centers/marshall/pdf/709025main_FASTSAT_Facts_11_2012.pdf

[13] M. R. Greene, "The Attitude Determination and Control System of the Generic Nanosatellite Bus," Ph.D. dissertation, University of Toronto Institute for Aerospace Studies, 2009.

[14] Space Micro, "Guidance and Navigation Products." [Online]. Available: http://www.spacemicro.com/G&NC/GNC.htm

[15] J. Enright, D. Sinclair, and K. C. Fernando, "COTS Detectors for Nanosatellite Star Trackers : A Case Study," in *25th Annual AIAA/USU Conference on Small Satellites*, 2011.

[16] A. Schwarzenberg-czerny, W. W. Weiss, A. F. J. Moffat, R. E. Zee, and S. M. Rucinski, "The BRITE Nano-Satellite Constellation Mission," in *38th COSPAR Scientific Assembly*, 2010.

[17] Blue Canyon Technologies, "About Us." [Online]. Available: http://bluecanyontech.com/about-us/

[18] S. Palo, G. Stafford, and A. Hoskins, "SSC13-III-5 An Agile Multi-use Nano Star Camera for Constellation Applications," in *27th Annual AIAA/USU Conference on Small Satellites*, 2013.

[19] Berlin Space Technologies, "About BST." [Online]. Available: http://www.berlin-space-tech.com/index.php?id=18

[20] BST, "ST-200 Autonomous Star Tracker Flyer," Berlin Space Technologies, Tech. Rep., 2013. [Online]. Available: http://www.berlin-space-tech.com/fileadmin/media/BST_ST-200_Flyer.pdf

[21] A. Kestila, T. Tikka, P. Peitso, J. Rantanen, K. Nasila, K. Nordling, H. Saari, R. Vainio, P. Janhunen, J. Praks, and M. Hallikainen, "Aalto-1 nanosatellite - technical description and mission objectives," *Geoscientific Instrumentation Methods and Data Systems*, pp. 121–130, 2013.

[22] H. E. Loubser, "The Development of Sun and Nadir Sensors for a Solar Sail CubeSat," Master's Thesis, Stellenbosch University, 2011.

[23] P. Botma, "The Design and Development of an ADCS OBC for a CubeSat," Dissertation, Stellenbosch University, 2011. [Online]. Available: http://scholar.sun.ac.za/handle/10019.1/18040

[24] P. Brown, "E2V CCD and CMOS device developments for spaceapplications," Tech. Rep., 2012.

[25] N. Truesdale and K. Dinkel, "DayStar: Modeling and test results of a balloon-borne daytime star tracker," in *Aerospace Conference, 2013 IEEE*, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6497411

[26] J. Li, J. Liu, and G. Li, "Study on the Detection Sensitivity of APS Star Tracker," in *3rd International Symposium on Advanced Optical Manufacturing and Testing Technologies: Optical Test and Measurement Technology and Equipment*, J. Pan, J. C. Wyant, and H. Wang, Eds., vol. 6723, Dec. 2007. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1305142

[27] Marshall Electronics, "Miniature Glass Lenses For Board Cameras." [Online]. Available: http://www.marshall-usa.com/optical/lenses/ccd_cmos/43fix.php

[28] Lensation, "Lensagon BL6012." [Online]. Available: http://www.lensation.de/en/shop/detail/25-light-sensitive/flypage/71-lensagon-bl6012.html?sef=hcfp

[29] B. Greyling, "A Charge Coupled Device Star Sensor System For a Low Earth Orbit Satellite," Ph.D. dissertation, Stellenbosch University, 1995.

[30] W. H. Steyn, M. J. Jacobs, and P. J. Oosthuizen, "A High Performance Star Sensor System for Full Attitude Determination on a Microsatellite," in *AAS Guidance and Control Conference*, 1997, pp. 1–13. [Online]. Available: http://staff.ee.sun.ac.za/whsteyn/Papers/AAS97_Star.pdf

[31] M. Kolomenkin, S. Polak, I. Shimshoni, and M. L. Member, "A Geometric Voting Algorithm for Star Trackers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 2, pp. 441–456, 2008. [Online]. Available: http://mis.haifa.ac.il/userfiles/file/ishimshoni_files/StarTracker.pdf

[32] M. J. Jacobs, "A Low Cost, High Precision Star Sensor," Master's Thesis, Stellenbosch University, 1995.

[33] D. Brown, "Decentering Distortion of lenses," D. Brown Associates, Inc, Eau Gallie, Florida, Tech. Rep., 1966.

[34] K. Huffman, "Designing Star Trackers to Meet Micro-satellite Requirements," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[35] M. Knutson and D. Miller, "Fast Star Tracker Centroid Algorithm for High Performance CubeSat with Air Bearing Validation," Ph.D. dissertation, Massachusetts Institute of Technology, 2012.

[36] B. B. Spratling and D. Mortari, "A Survey on Star Identification Algorithms," *Algorithms*, vol. 2, no. 1, pp. 93–107, Jan. 2009. [Online]. Available: http://www.mdpi.com/1999-4893/2/1/93/

[37] C. Padgett, K. Kreutz-Delgado, and S. Udomkesmalec, "Evaluation of Star Identification Techniques," Jet Propulsion Laboratory, Tech. Rep., 1996.

[38] C. Padgett and K. Kreutz-Delgado, "A Grid Algorithm for Autonomous Star Identification," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 1, 1997.

[39] C. R. Mcbryde, "A Star Tracker Design for CubeSats," Master's Thesis, The University of Texas at Austin, 2012.

[40] ESA, "The Hipparcos Space Astrometry Mission." [Online]. Available: http://www.rssd.esa.int/index.php?project=HIPPARCOS

[41] ——, "Access the Catalogue Data." [Online]. Available: http://www.rssd.esa.int/index.php?project=HIPPARCOS&page=Research_tools

[42] F. Markley, "Attitude Determination using Vector Observations and the Singular Value Decomposition," *Journal of the Astronautical Sciences*, vol. 38, 1988.

[43] G. Wahba, "A least squares estimate of satellite attitude," *SIAM Review*, vol. 7, no. 3, 1966.

[44] C. D. Hall, "Chapter 4: Attitude Determination," in *Spacecraft Attitude Dynamics and Control*. Virginia Tech, 2003, ch. 4. [Online]. Available: http://www.dept.aoe.vt.edu/~cdhall/courses/aoe4140/

[45] M. Shuster and S. Oh, "Three-Axis Attitude Determination from Vector Observations," *Journal of Guidance and Control*, vol. 4, no. 1, 1981. [Online]. Available: http://home.comcast.net/~mdshuster/Pub_1981a_J_TRIAD-QUEST_scan.pdf

[46] M. D. Shuster, "The Quest for Better Attitudes," *The Journal of the Astronautical Sciences*, vol. 54, no. December, pp. 657–683, 2006. [Online]. Available: http://www.malcolmdshuster.com/Pub_2006f_J_Brouwer_AAS.pdf

[47] Y. Cheng and M. D. Shuster, "An Improvement to the QUEST Algorithm," *Journal of the Astronautical Sciences*, no. 1294. [Online]. Available: http://www.malcolmdshuster.com/commp_010_031z_J_JAS1290_cquest.pdf

[48] S. Takahashi, L. de Souza, and M. C. Tosin, "EXECUTION TIME OF QUEST ALGORITHM IN EMBEDDED PROCESSORS," in *Brazilian Symposium on Aerospace Eng. & Applications*, 2009. [Online]. Available: http://www.cta-dlr2009.ita.br/Proceedings/PDF/59573.pdf

[49] F. Markley and D. Mortari, "Quaternion Attitude Estimation Using Vector Observations," *The Journal of the Astronautical Sciences*, vol. 48, no. 2-3, pp. 359–380, 2000. [Online]. Available: http://home.comcast.net/~mdshuster3/FC_MarkleyMortari_2000_J_MDScan.pdf

[50] P. Bourke, "Guassian Elimination," 1997. [Online]. Available: http://paulbourke.net/miscellaneous/gausselim/

[51] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab," 2010. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[52] C. Liebe, "Accuracy performance of star trackers - a tutorial," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 2, pp. 587–599, Apr. 2002. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1008988

Appendices

Appendix A

Ground Support Software

CubeStar Ground Support GUI

The CubeStar Ground Support Graphical User Interface (GSGUI) is a stand alone application which interfaces to CubeStar while on the ground for testing and calibration. The GSGUI software displays telemetry and allows the user to send a wide variety of commands to CubeStar. Commands include:

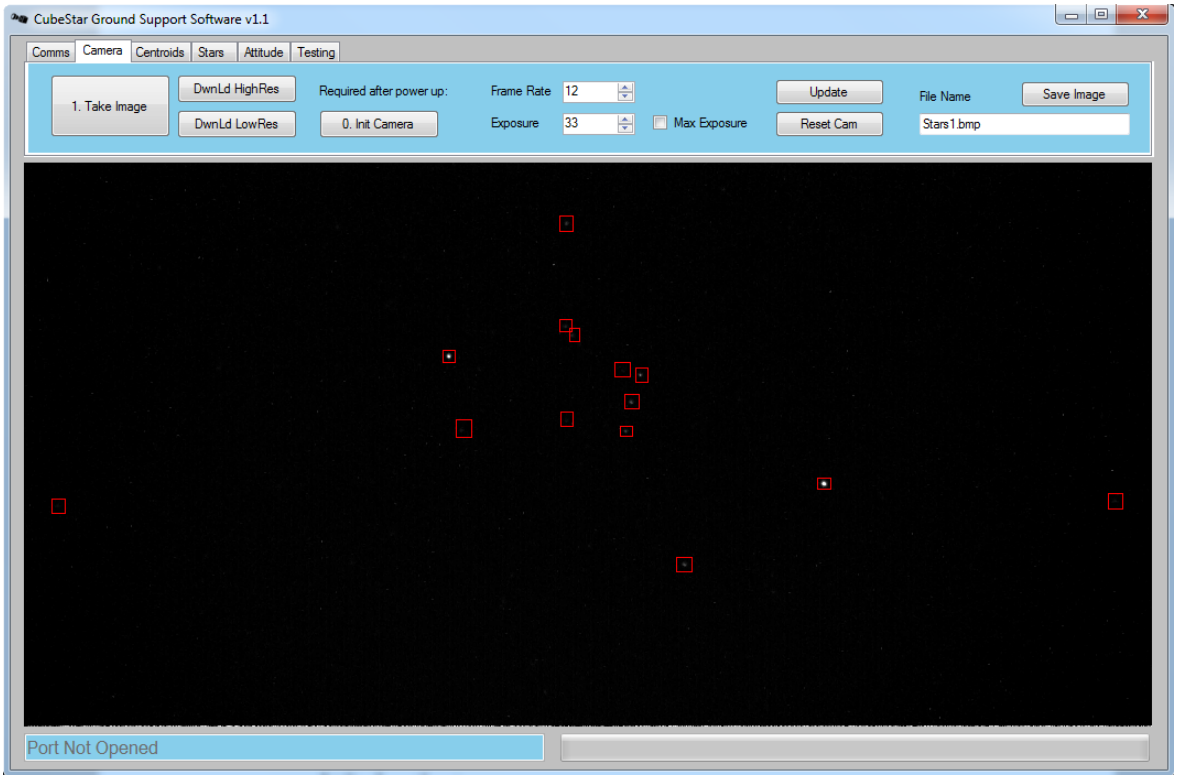


Figure A.1: The CubeStar Ground Support Graphical User Interface

- Take an image
- Find centroids and match stars
- Calculate attitude
- Set exposure, frame rate, detection thresholds

Telemetry display can include:

- Raw images
- Raw images overlaid with detected centroids
- Number of centroids detected and number of matched stars
- Average match confidence
- Attitude estimate

In addition, the GSGUI software can be used to upload simulated star images to CubeStar.

MATLAB Scripts

This section gives a brief overview of MATLAB scripts which may be useful for testing or calibrating CubeStar.

Star Catalogue Generator This script takes a reduced version of the Hipparcos star catalogue in text format and generates CubeStar's onboard star list and inter-star distance list. The magnitude cut-off can be specified. The script can be set to output comma separated lists which can be copied and pasted into CubeStar's source code.

Test Image Generator This script generates a simulated star image using a given pointing vector. The image has dimensions identical to those produced by CubeStar. Images produced by this script can be uploaded to CubeStar to test its algorithms. Alternately, this script can be used to see what CubeStar should be seeing given a known attitude.

Star Matcher This script can take a raw or simulated star image and output an estimated attitude. It contains all the same algorithms as CubeStar, so it can be used

to verify CubeStar's estimates. Changes to the matching algorithm or its parameters should be evaluated on this script before being applied to CubeStar.

CubeStar Interface This Simulink file can be used to interface to CubeStar. It polls CubeStar for new attitude information regularly and stores the attitude estimates in the workspace for later evaluation.

Appendix B

Derivations

Image Plane To Unit Vector Transformation

This is a derivation of the equation used in Section 4.1.5 to transform a point on the image plane into a 3-dimensional unit vector. The equation is given again below:

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} (x_u - x_c)\frac{pp_x}{f_{mm}}[1 + ((x_u - x_c)\frac{pp_x}{f_{mm}})^2 + ((y_u - y_c)\frac{pp_y}{f_{mm}})^2]^{-\frac{1}{2}} \\ (y_u - y_c)\frac{pp_y}{f_{mm}}[1 + ((x_u - x_c)\frac{pp_x}{f_{mm}})^2 + ((y_u - y_c)\frac{pp_y}{f_{mm}})^2]^{-\frac{1}{2}} \\ [1 + ((x_u - x_c)\frac{pp_x}{f_{mm}})^2 + ((y_u - y_c)\frac{pp_y}{f_{mm}})^2]^{-\frac{1}{2}} \end{bmatrix} \tag{B.0.1}$$

where

$$\begin{aligned} u_x, u_y, u_z &= \text{components of a unit vector} \\ x_u, y_u &= \text{undistorted centroid coordinates in pixels} \\ x_c, y_c &= \text{pixel coordinates of the principal point} \\ pp_x, pp_y &= \text{pixel pitches of the imager} \\ f_{mm} &= \text{focal length of the lens in mm} \end{aligned}$$

Derivation:

Referring to Figure B.1, which is a representation of the pinhole model of a camera, and using Pythagoras:

$$r_1 = \sqrt{x^2 + y^2} \tag{B.0.2}$$

$$r_2 = \sqrt{r_1^2 + f^2} = \sqrt{x^2 + y^2 + f^2} \tag{B.0.3}$$

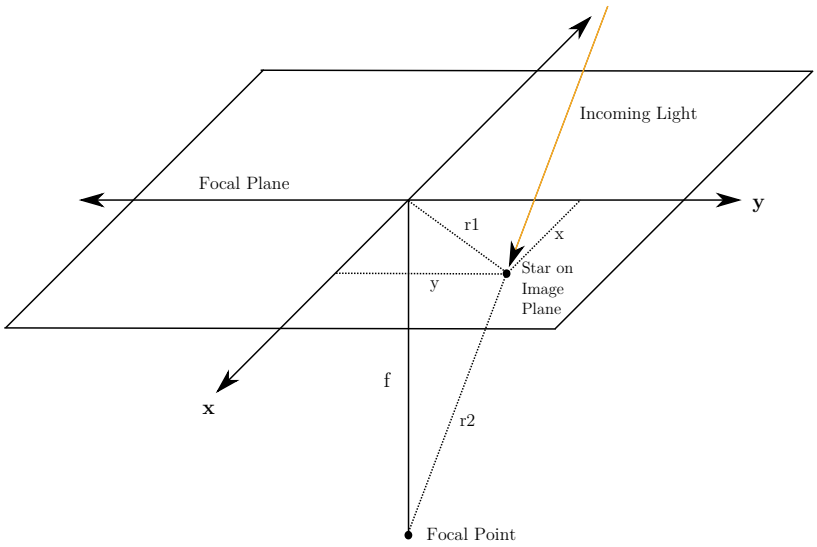


Figure B.1: Pin hole camera model. Note, the focal plane is placed in front of the focal point in this example to simplify the derivation.

The location of the star on the image plane in 3-dimensional space, with the focal point as the origin, is (x, y, f). To transform this into a unit vector, each of the coordinate components need to be divided by the scalar r_2, which is the magnitude of the vector (x, y, f).

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} x/r_2 \\ y/r_2 \\ f/r_2 \end{bmatrix} = \begin{bmatrix} (x/f)unitVect_z \\ (y/f)unitVect_z \\ f/r_2 \end{bmatrix} \tag{B.0.4}$$

Substituting Equation B.0.3 into Equation B.0.4 for the z component only at first:

$$unitVect_z^2 = \frac{f^2}{x^2 + y^2 + f^2} = \frac{1}{\frac{x^2}{f^2} + \frac{y^2}{f^2} + \frac{f^2}{f^2}} \tag{B.0.5}$$

$$unitVect_z = \frac{1}{\sqrt{\frac{x^2}{f^2} + \frac{y^2}{f^2} + 1}} \tag{B.0.6}$$

Substituting the result from Equation B.0.6 into Equation B.0.4 yields:

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} \frac{x}{f}[(\frac{x}{f})^2 + (\frac{y}{f})^2 + 1]^{-\frac{1}{2}} \\ \frac{y}{f}[(\frac{x}{f})^2 + (\frac{y}{f})^2 + 1]^{-\frac{1}{2}} \\ [(\frac{x}{f})^2 + (\frac{y}{f})^2 + 1]^{-\frac{1}{2}} \end{bmatrix} \tag{B.0.7}$$

The result from Equation B.0.7 is the unitless version of Equation B.0.1. The other terms in B.0.1 account for conversions between different units and an offset principal point.

Appendix C

Hardware Details

Camera Board Details

The CubeStar image sensor board is compatible with CubeSense V1. Unfortunately, the pin out got mirrored on the Melexis image sensor PCB, meaning the Melexis image sensor PCB must be rotated by 180 degrees compared to the CubeSense camera PCB when plugging it into CubeSense. The pinout of the image sensor board is given in Table C.1. Pin 1 is at the top right in Figure 5.3.

Pin	**Function**	**Detail**
1-8	parallel pixel data	upper 8 bits
10	camera reset	active low
11	I2C	SDAT
13	I2C	SCLK
14	Horizontal sync	low between lines
16	Vertical Sync	low between frames
18	Pixel Data Clock (PIXCLK)	high during each pixel output
20	Power Supply	5 V if regulator populated, else 3.3 V
21	Ground	
22	Power Supply	connected to pin 20
31	Ground	

Table C.1: Pin descriptions of the CubeStar image sensor board. Pin 1 is closest to the corner diagonally opposite the crystal.

The Melexis image sensor requires both 1.8 V and 3.3 V sources. Two linear voltage regulators are included on the image sensor board, allowing the board to be powered off a single 5 V source. The image sensor requires a maximum of 32 mA from its 1.8 V source and a maximum of 82 mA from its 3.3 V source, according to its datasheet. The

MCP1824S18 and MCP1824S33 300 mA lowdrop out regulators in SOT223-3 packages were selected. When interfacing to CubeSense, both regulators must be populated. When interfacing to the rest of the CubeStar stack, the 3.3 V regulator must be replaced with an electrical short. The image sensor board is then connected directly to the power supply of CubeStar, via a power switch on the processor board.

The Melexis image sensor is always an I2C slave. 10 kΩ pull-up resistors are present on the I2C lines. These must be removed when interfacing to CubeSense, as CubeSense supplies its own pull-up resistors.

Pins 1-8 of the image sensor board are connected directly to the upper eight bits of the parallel pixel data output of the Melexis image sensor. The Melexis image sensor outputs 12-bit data, but only the upper eight bits are used. This minimises the required SRAM storage, requires fewer connections between the boards and makes the software development easier as there is no native support for 12-bit data types in C.

An 11 MHz crystal on the image sensor board supplies the clock to the Melexis image sensor. The Melexis datasheet specifies a minimum input clock of 20 MHz. At lower input clock frequencies the automatic exposure controller, called *Autobrite*, seizes to function correctly. Fortunately, after consultation with Melexis support, it was confirmed that the sensor would continue to function as long as the automatic exposure controller was turned off and the exposure settings were set manually through the I2C port.

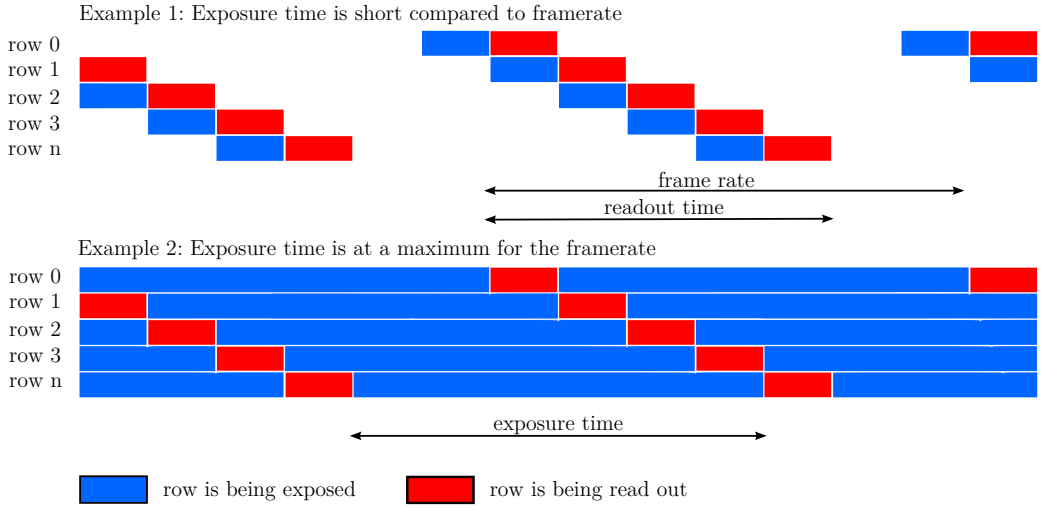


Figure C.1: Timing diagrams depicting the operation of a rolling shutter CMOS image sensor

The input clock to the Melexis image sensor determines the output data rate of the

pixel data. The Pixel Data Clock (PIXCLK) runs at the same frequency as the input clock. Whenever PIXCLK goes high, the eight parallel bits of pixel data can be read. With an 11 MHz input clock, 11 million pixels worth of data is output by the Melexis image sensor every second. Since the Melexis image sensor has a rolling shutter, each row of pixel data is output sequentially, taking approximately 100 μs per row. Figure C.1 displays the operation of a rolling shutter and shows that the maximum exposure time is limited by the frame rate. An 11 MHz input clock was chosen to allow exposure times of up to 500 ms. The minimum frame rate achievable with a given input clock is given by Equation C.0.1 from the Melexis datasheet. According to this equation, an 11 MHz input clock allows a minimum frame rate of 2.03 frames/s. More information on exposure times is given in Appendix D.

$$\text{Frame Rate}_{min} = \frac{clock_{input}}{5413590} \tag{C.0.1}$$

The Melexis image sensor outputs a constant stream of images. The HSync pin goes low briefly at the end of every row of pixel data and the VSync pin goes low between images.

The image sensor board basically implements the typical application circuit given in the Melexis datasheet. Besides the regulators and crystal, the only other supporting components in the typical application circuit are bypass capacitors and pull up resistors. The optional external EEPROM is not required. The datasheet notes that for the best, noise-free operation the analogue and digital grounds should be separated. However, since the typical application circuit has a common ground plane, the image sensor board was designed with one, too. Test images show very faint vertical noise, which could be due to the common ground. Fortunately, however, the noise is too small to negatively impact the operation of CubeStar.

Ball grid array components are very difficult to solder reliably without the right equipment. The first attempt at attaching the Melexis sensor to the PCB resulted in a hidden solder bridge. This subsequently damaged the image sensor when it was powered on. The second soldering attempt with a new image sensor was successful.

FPGA Board Details

The FPGA is connected to the SRAM module with 19 address lines and eight data lines.

Pin	Function	Detail
1-8	parallel pixel data	from image sensor
9	FPGA control	ACK (Acknowledge)
12	FPGA control	ENABLE
14	Horizontal sync	from image sensor
16	Vertical Sync	from image sensor
18	Pixel Data Clock (PIXCLK)	from image sensor
19	FPGA control	RNW (Read not Write)
20	Power Supply	3.3 V
21	Ground	
22	Power Supply	connected to pin 20
23-30	parallel data	to processor
31	Ground	
32	data clock	from processor

Table C.2: Pin descriptions of the CubeStar FPGA board. Pin 1 is closest to the corner diagonally opposite the LED (Pin numbers are the same as those for the image sensor board).

When enabled, the Melexis image sensor is always outputting image data. There is no snapshot command. Therefore, the flow of data from the image sensor to the FPGA is controlled by the image sensor. If a new image is requested by the processor by asserting the Enable line (and deasserting the RNW line), the FPGA immediately starts reading and storing data from the image sensor. However, the image sensor might be in the middle of outputting an image. If this is the case, the FPGA's HSync counter will keep incrementing as it stores data, but it will not reach 511 before a VSync. A new image will follow the VSync, which will be successfully stored by the FPGA. As such, an indeterminate amount of time will pass between the request for an image and the assertion of the ACK pin. In the worst case it can take up to two exposure times for an image to be successfully captured. This issue caused major problems when trying to get CubeStar's update rate to 1 Hz. The solution is to disable the image sensor and re-enable just before an image must be captured.

Processor Board Details

A mod to connect two unconnected pins on the crystal to ground is required on version one of the processor board.

The EBI consists of 16 pins, which allows a maximum of 8-bit address and 8-bit data throughput in a single cycle. Using a latch, two additional EBI modes are available: 16-bit address/16-bit data and 24-bit address/8-bit data. A minimum of 512 kB of external

Pin	Function	Detail
1	FPGA Control	ACK (Acknowledge)
2	camera reset	active low
3	I2C	SDAT
4	FPGA Control	ENABLE
5	I2C	SCLK
11	FPGA Control	RNW (Read not Write)
12	Power Supply	3.3V
13	Ground	
14	Power Supply	connected to pin 12
15-22	Parallel Data	from FPGA
23	Ground	
24	Data Clock	to FPGA

Table C.3: Pin descriptions of the CubeStar processor board. Pin 1 is closest to the corner diagonally opposite the Gecko processor IC.

memory is required to store a single image, however, a 1024 kB SRAM module was chosen to give some flexibility. A 1024 kB SRAM module has 19 address pins and 8-bit data, so the 24-bit address/8-bit data mode is used. The latch is an SN74LVCH16373A from Texas Instruments.

Current consumption is measured using INA139 current shunt monitors, which output a voltage proportional to the current flowing through a 1 Ohm resistor. Opamps serves as a buffers between the INA139s and the processor inputs. FPF2123 load switches, controlled by the processor, can cut current to the SRAM module or other boards in the stack if abnormal current consumption is detected.

Tests with the Gecko ARM Cortex processor clocked at 48 MHz indicate that its maximum data input rate is approximately 1 million bytes per second. This test was performed by toggling an I/O pin (Data Clock, see Section 5.2.2) as fast as possible while reading and storing a byte from eight parallel I/O pins. Since one image is 512 kB, it takes approximately 500 ms to transfer an image from the FPGA board to the processor's external SRAM.

An LED is connected to pin 51 of the Gecko processor. The LED can be toggled in software to aid in debugging code.

The pinouts of headers are given in Table C.4 and Table C.5.

Pin	Function	Detail
1	Power Supply	3.3 V
2	I2C	SDAT
3	I2C	SCLK
4	Ground	
5	UART	TX (transmit)
6	UART	RX (receive)

Table C.4: Pin descriptions of the CubeStar-satellite interface. Pin 1 is closest to the label *SAT*.

Pin	Function
1	nRESET
2	SWO
3	Ground
4	SWCLK
5	SWDIO
6	3.3 V

Table C.5: Pin descriptions of the programming interface. Pin 1 is furthest from the label *PROG*.

Appendix D

Software Details

Camera Interface Details

The Melexis image sensor is controlled over an I2C interface. The image sensor is always the slave. I2C drivers written for CubeComputer as part of the Board Support Package were reused for CubeStar. The Melexis image sensor is controlled by writing to its internal registers. A single write cycle is displayed in Table D.1.

S	Device Addr	0	A	Register MSB	Register LSB	A	Data	A	P

Table D.1: A single write cycle to an image sensor register. S - start bit, A - acknowledge bit, P - stop bit

The device address is seven bits long and is used when more than one Melexis image sensor is on the same I2C bus. Melexis image sensors are given a device address of zero by default. Since CubeStar uses only one image sensor, it is addressed as image sensor zero, which corresponds to a device address of 0101_100 according to the datasheet. The eighth bit of the address field is set to zero for write operations and one for read operations. All start and stop bits and acknowledge checks are taken care of by the I2C driver. After a register has been written to, it is often necessary to also write 0x03 to register 0x8500, known as the *sync register*, in order for the changes to take effect (The datasheet mentions which changes require the sync register).

Some important registers are discussed below:

0x8100 - Mode Select

Register 0x8100, known as the *Mode Select* register, is eight bits long and contains two important settings. Bits five and six control the output pixel data bit depth. Bit five should be set to one and bit six should be set to zero to set 8-bit mode. Bit zero enables the image sensor. If this bit is set to zero, no image data will be gathered and the image sensor will use less power. When the bit is set back to one an image with an undetermined exposure will start clocking out. This image must be discarded, but all following images will be exposed according to the register settings.

0x8102 - Hardware Feature Enables

Register 0x8102, known as the *Hardware Feature Enables* register, determines which of the hardware image processing features on the Melexis image sensor are enabled. Histogram remapping, controlled by bit six, tries to condense the 12-bit pixel output into eight bits without loss of information by emphasizing areas in the histogram that contain the most information. More information on this algorithm is available in the Melexis datasheet. The other hardware features controlled by this register are image sharpening, defective pixel correction, fixed-pattern noise correction and dark current subtraction.

For application on CubeStar, all hardware features except sharpening are enabled. Sharpening would cause undesired distortions around the edges of stars.

0x8502 - Firmware Feature Enables

The Melexis image sensor contains several software image processing functions, which are enabled by register 0x8502. AutoView, controlled by bit seven, enables the histogram remapping algorithm to calculate new remapping constants. If this bit is set to zero, the histogram will still be remapped (provided it is enabled in the Hardware Feature Enables register), but new constants will not be calculated. Barrier Update and Auto Dynamic Range, bits six and four, work together. The Melexis image sensor can be set to have a non-linear response to light intensity to enable high dynamic range performance. By setting bits six and four an algorithm will automatically adjust the image sensor's light response curve to prevent saturation. Bit five controls automatic exposure.

For imaging stars, the Melexis image sensor needs to be set to maximum sensitivity. This is achieved by turning off all firmware features except AutoView (histogram

remapping) and setting the exposure manually. Melexis support suggests putting the image sensor in linear mode by setting registers 0x840B-0x841F[1] to zero for maximum sensitivity.

0x8526-0x8527 - Exposure

Registers 0x8526 and 0x8527 control the exposure of the Melexis image sensor. The exposure time is expressed in row times. One row time corresponds to 1322 clock periods. 0x8526 contains the most significant byte and 0x8527 contains the least significant byte. If the exposure needs to be set manually, auto exposure must be disabled in the Firmware Feature Enables register.

The exposure time is limited by the frame rate, as shown in Figure C.1. The longest possible exposure time at a particular frame rate will be approximately equal to one over the frame rate (it will differ by the read out time of one line, which is equal to the row time). Instead of setting the exposure time manually, Melexis support suggests setting registers 0x8444[1] and 0x8545[1] to 0x00 and 0x02 respectively. This sets the exposure to the maximum allowable by the current frame rate. See the section on Frame Rate for more information.

0x851C-0x851D - Frame Rate

In order to set a specific frame rate, the Melexis image sensor first needs to know its input clock frequency. This is set by writing the number of clocks per second into registers 0x8518-0x851B. By default these registers contain a value consistent with 27 MHz (27 000 000 = 0x019BFCC0). The clock frequency registers only need to be set once during power-up. Thereafter the desired number of frames per second (fps) can be written to registers 0x851C-0x851D. By default, these registers contain the value 0x001E, indicating 30fps. Only integer numbers of frames per second can be set.

CubeStar has very tight timing requirements. An image needs to be taken, transferred to processor SRAM and processed within 1 second. Therefore, it is desirable to be able to set fractional numbers of frames per second around the 2 fps mark. Through experimentation, it has been found that leaving the clock frequency registers set to 27 MHz while using a 10 MHz crystal allows finer control over the frame rate. The actual frame rate is then 10/27 of the frame rate set in the registers. Table D.2 lists some useful frame rates, which were confirmed with an oscilloscope.

[1] These registers do not appear in the datasheet

Register Setting	Expected frames/s	Measured frames/s	Approx Exposure (ms)
30	11.111	11.110	90
9	3.333	3.333	300
8	2.963	2.966	337
7	2.593	2.593	386
6	2.222	2.224	450

Table D.2: Useful frame rates and their corresponding approximate maximum exposure times

A register setting of seven, giving 2.593 fps, is chosen as the default for CubeStar. Slower frame rates, while allowing longer exposures, do not leave enough time for the image processing and attitude determination algorithms to complete within 1 second.

Capturing an Image

Capturing an image with the CubeStar hardware is a multi-step process. The process is described in detail in this subsection.

Before capturing an image, several image sensor registers must be initialised:

- Set 8 bit output mode and enable the image sensor in the Mode Select Register
- Turn on all hardware options except image sharpening in the Hardware Feature Enables register
- Set the camera in linear mode by writing 0x00 to registers 0x840B-0x841F
- The frame rate must be set by writing to registers 0x851C-0x851D
- Write 0x03 to the Sync register

Next, the exposure must be set depending on the desired operating environment. If daylight images are to be taken for testing purposes:

- All firmware can be turned on in the Firmware Feature Enables Register[2]
- The exposure must be set manually by writing to registers 0x8526 and 0x8527

[2]Most of the firmware features will not function at low clock speeds and seem to have no effect when turned on.

- Write 0x03 to the Sync register

If star images are to be taken:

- All firmware except AutoView must be turned off in the Firmware Features Enable register
- The exposure can be set to the maximum allowed for the current frame rate by writing 0x00 and 0x02 to registers 0x8444 and 0x8445 respectively. Alternatively, the exposure can be set manually as before
- Write 0x03 to the Sync register

Once these registers have been initialised, an image can be captured. There are four functions involved in capturing an image: *getImage*, *imagerEnable*, *fpgaWriteToRam* and *imageTransfer*. The image capture process is depicted in Figure 5.6.

An image capture is initiated by calling the *getImage* function, which does not require any arguments. *getImage* calls each of the other functions in turn to complete the image capture, before returning.

The function *imagerEnable* can enable or disable the image sensor by writing to bit 0 of its Mode Select Register. Upon being enabled, the image sensor will start clocking out an image with undetermined exposure. This image must not be saved, therefore a delay must be inserted between enabling the image sensor and enabling the FPGA to save the image. By including this delay, the FPGA will save the second image to be clocked out, which will have the correct exposure.

The read out time of one line is 1322 clock cycles. Therefore, the read out time of the whole image is 512 x 1322 clock cycles. At 10 MHz, this equates to 67.69 ms. Ideally, the FPGA should not save an image if it is enabled even slightly after the start of the image readout. However, for unknown reasons, a delay of 70 ms seems to work best for capturing the second image. Shorter delays result in a horizontal split in the captured image, indicating that the captured image is actually a combination of two images captured at different exposures.

The function *fpgaWriteToRam* sets the FPGA into write mode, which then starts storing pixel data from the image sensor into external RAM. Once a complete image has been stored, the FPGA raises the ACK line, which is sensed by the *fpgaWriteToRam* function. The *fpgaWriteToRam* function returns once the ACK line has gone high. The

duration of the *fpgaWriteToRam* function is roughly equal to the current frame rate of the image sensor.

The function *imageTransfer* sets the FPGA into read mode and transfers the image data from the FPGA's external SRAM to the processor's external SRAM. The processor is in control of the transfer process. The transfer is sequential, beginning at the first byte of image data from the external SRAM. Every time the processor raises the Data Clock line, the FPGA presents the next byte of image data from the external SRAM onto the eight parallel data lines. The processor reads the byte from the eight parallel data lines and stores it in sequential locations in its own external SRAM. The process repeats until all image bytes have been transferred. The duration of the *imageTransfer* function is dependent on the clock speed of the processor. Experiments show that the maximum input data rate of the Gecko processor, when transferring data in this way, is approximately 1 MB per second. At this rate, the *transferImage* function takes approximately 500 ms (See Section 5.3.6).

Once *transferImage* has completed, *getImage* returns. The new image is then available in the processor's external SRAM, beginning at address zero. The complete image capture process can take up to 1000 ms if the image sensor's exposure time is close to 500 ms. Therefore, the star matching and attitude determination algorithms must be run in parallel with the image capturing process, as explained in detail in Section 5.3.6.

Appendix E

Important Datasheet Extracts

MLX75411, MLX75412

Avocet HDR Image Sensors Datasheet

Features and Benefits

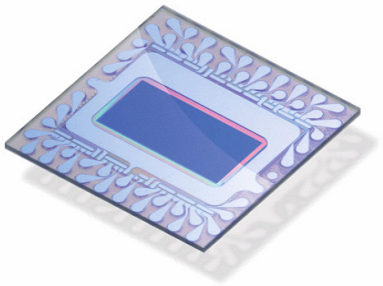
- 1024 x 512 pixels CMOS image sensor
- 154dB Extended HDR
- Low noise, low power rolling shutter
- 1/3” Optical format for 1024x512
- 1/4" Optical format with VGA subwindow
- Monochrome, standard Bayer, RCCC and special Bayer RGBi
- Parallel data output
  8/10/12 bits + CLK/HSYNC/VSYNC
- Operating Temperature Range:
  -40C to +85°C full performance
  -40C to +115°C reduced performance
- Storage Temperature Range:
  -40°C to +125°C
- Automotive Qualified AEC-Q100

MLX75412 Only

- Autobrite® auto-exposure (AE) and auto-HDR. This high performing function automatically sets the best exposure and HDR setting on a frame by frame basis.
- Autoview™ histogram remapping. This function enhances display viewing performance.

Application Examples

- Automotive Driver Assistance Systems (ADAS)
  - Lane departure warning (LDW)
  - Forward collision warning (FCW)
  - Automatic high-beam assist
  - HDR rear-view
- Cameras on trucks, trains, busses, emergency vehicles, agricultural vehicles, autonomous vehicles, heavy off-road vehicles
- Night vision cameras
- HDR surveillance cameras
- Traffic monitoring cameras
- Fleet Safety/ Black-box cameras



Ordering Information

Part No.	Temperature Code	Package Code	Option code
MLX75411	V (-40°C to 115°C)	TF (glass-BGA) or UC (wafer)	M or G or I or R
MLX75412	V (-40°C to 115°C)	TF (glass-BGA) or UC (wafer)	M or G or I or R

7 Sensor Specific Specifications

DC Operating Parameters T_A = 25°C, V_{DDIO} = 3.3V ± 5%

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
Sensitivity	SNR10	@535nm		25		nW/cm^2
Responsivity		@535nm		2.28		e-/DN12
Dark Leakage		T=65°C		1008		DN12/sec
Number of barriers					6	
Dynamic Range					154	dB
Chief ray angle the sensor has been optimized for.	CRA			10		degrees

Table 6: Specifications: Optical Characteristics

8 Device Overview

8.1 Avocet Sensor Overview

The Avocet image sensor integrates a high-sensitivity array, a feature-rich digital processor for monochrome images and camera control functions into a single chip.

Specification	Avocet	Comments
Active Resolution	1024 x 512	Wider horizontal resolution for next generation ADAS
Optical format	~1/3“(6.45mm)	Center 1/4“ can be used for VGA resolution
Pixel size	5.6µ square	Optimized for sensitivity at 1024x512 resolution
Max frame rate	60fps	At full resolution
Input clock range	20 – 54 MHz	Options for clocking: Crystal input, Oscillator
Exposure time range @ 60fps	12.5µs – 16.7 ms	At 54MHz and 60fps, at full resolution and speed. Minimum barrier time 1.22us.
Control interface	2 Wire Boot Loader Interface	Used to load register settings on recovery from a reset. Must be accessible (for programming of serial PROM) via other control interfaces.
	2 Wire Serial Slave	2-Wire, low speed, serial control interface used for short distances. Supports broadcast writes

		for writing to multiple imagers.
Video interface	LVTTL	12-bit Monochrome or Raw Color. Pixel clock, vsync and hsync compatible with DSPs. (i.e. TI DaVinci or ADI Blackfin)
Signal processing	Defect pixel interpolation FPN correction Histogram Optimization Dark Current correction Sharpening	Sensor provides on-chip processing required for vision applications or monochrome display applications. (Color processing is not included in on-chip functions.)
Scanning modes	Progressive	Required to support machine vision applications
	Subsample	2x and 4x vertical subsampling
	Subwindow	Single rectangular region. The starting point of the x- and y-address is programmable, as well as the window size.
Sensitivity	SNR10	25nW/cm2 @ 25degC @ 535nm
Responsivity	2.28 e-/DN12	In dark at 25°C 535nm
Dark leakage	1008 DN12/sec	At 65°C

Table 4 - Avocet image sensor highlights

EFM32

... the world's most energy friendly microcontrollers

EFM32GG280 DATASHEET

F1024/F512

Preliminary

- **ARM Cortex-M3 CPU platform**
  - High Performance 32-bit processor @ up to 48 MHz
  - Memory Protection Unit
- **Flexible Energy Management System**
  - 20 nA @ 3 V Shutoff Mode
  - 0.4µA @ 3 V Shutoff Mode with RTC
  - 0.9 µA @ 3 V Stop Mode, including Power-on Reset, Brown-out Detector, RAM and CPU retention
  - 1.1 µA @ 3 V Deep Sleep Mode, including RTC with 32.768 kHz oscillator, Power-on Reset, Brown-out Detector, RAM and CPU retention
  - 50 µA/MHz @ 3 V Sleep Mode
  - 200 µA/MHz @ 3 V Run Mode, with code executed from Flash
- **1024/512 KB Flash**
  - Read-while-write support
- **128/128 KB RAM**
- **85 General Purpose I/O pins**
  - Configurable Push-pull, Open-drain, pull resistor, drive strength
  - Configurable peripheral I/O locations
  - 16 asynchronous external interrupts
  - Output state retention and wakeup from Shutoff Mode
- **12 Channel DMA Controller**
- **12 Channel Peripheral Reflex System (PRS) for autonomous inter-peripheral signaling**
- **Hardware AES with 128/256-bit keys in 54/75 cycles**
- **Timers/Counters**
  - 4× 16-bit Timer/Counter
    - 4×3 Compare/Capture/PWM channels
  - 16-bit Low Energy Timer
  - 1× 24-bit and 1× 32-bit Real-Time Counter
  - 3× 16/8-bit Pulse Counter with asynchronous operation
  - Watchdog Timer with dedicated RC oscillator @ 50 nA
- **Backup Power Domain**
  - RTC and retention registers in a separate power domain, available in all energy modes
  - Operation from backup battery when main power drains out
- **External Bus Interface for up to 4×256 MB of external memory mapped space**
  - TFT Controller with Direct Drive
- **Communication interfaces**
  - 3× Universal Synchronous/Asynchronous Receiver/Transmitter
    - UART/SPI/SmartCard (ISO 7816)/IrDA/I2S
  - 2× Universal Asynchronous Receiver/Transmitter
  - 2× Low Energy UART
    - Autonomous operation with DMA in Deep Sleep Mode
  - 2× I^2C Interface with SMBus support
    - Address recognition in Stop Mode
- **Ultra low power precision analog peripherals**
  - 12-bit 1 Msamples/s Analog to Digital Converter
    - 8 single ended channels/4 differential channels
    - On-chip temperature sensor
  - 12-bit 500 ksamples/s Digital to Analog Converter
    - 2 single ended channels/1 differential channel
  - 2× Analog Comparator
    - Capacitive sensing with up to 16 inputs
  - 3× Operational Amplifier
    - 6.1 MHz GBW, Rail-to-rail, Programmable Gain
  - Supply Voltage Comparator
- **Low Energy Sensor Interface (LESENSE)**
  - Autonomous sensor monitoring in Deep Sleep Mode
  - Wide range of sensors supported, including LC sensors and capacitive buttons
- **Ultra efficient Power-on Reset and Brown-Out Detector**
- **Debug Interface**
  - 2-pin Serial Wire Debug interface
    - 1-pin Serial Wire Viewer
  - Embedded Trace Module v3.5 (ETM)
- **Pre-Programmed Serial Bootloader**
- **Temperature range -40 to 85 ºC**
- **Single power supply 1.85 to 3.8 V**
- **LQFP100 package**

32-bit ARM Cortex-M0+, Cortex-M3 and Cortex-M4F microcontrollers for:

- Energy, gas, water and smart metering
- Health and fitness applications
- Smart accessories
- Alarm and security systems
- Industrial and home automation
- www.energymicro.com/gecko

ZERO TINY GECKO LEOPARD GIANT WONDER

ENERGY micro