UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE FÍSICA

Development of a low-cost Star Tracker for CubeSats

Susana Fernandes Lopes

Mestrado Integrado em Engenharia Física

Dissertação orientada por:
Dr. Paulo Gordo
Prof. Dr. António Amorim

2019

Aos meus avós,
as estrelas que iluminam o meu caminho.

Agradecimentos

Em primeiro lugar, quero deixar um agradecimento à coordenação do meu curso, professora Dra. Olinda Conde, e a todo o pessoal pertencente ao Departamento de Física que contribuiu para a minha formação. Agradeço também à empresa Omnideia, em particular ao CEO Tiago Pardal, pela oportunidade de desenvolver a minha dissertação de mestrado, *Development of a low-cost star tracker for cubeSats*, em regime empresarial, o que me permitiu crescer tanto a nível pessoal como a nível profissional. A toda a equipa da Omnidea agradeço todos os dias que passei com cada um: a forma calorosa como me receberam será certamente algo que me acompanhará a vida toda e da qual nunca me vou esquecer. Ao meu orientador, Dr. Paulo Gordo, agradeço o tema que me foi proposto e o apoio incondicional que me deu durante todo o processo. Sem a sua paciência e capacidade de me guiar no caminho certo, nunca teria conseguido alcançar os resultados finais que alcancei. Agradeço também ao meu co-orientador, Professor Dr. António Amorim, por me ter ajudado com o trabalho de caracterização do sensor de imagem e ao aluno de doutoramento Bruno Couto. Quero deixar ainda um grande agradecimento ao Dr. Alberto Krone-Martins, por ter partilhado comigo todo o seu conhecimento sobre *star trackers* e por ter esclarecido todas as dúvidas que me foram surgindo ao longo do desenvolvimento deste trabalho. Aos meus amigos e colegas, agradeço todos os bons momentos que tivémos e toda a partilha de conhecimento que houve entre nós.

No âmbito do tema de dissertação que me foi proposto pela Omnidea, fui seleccionada para fazer parte da primeira edição do *workshop* da Agência Espacial Europeia *CubeSat hands-on training week*, em Fevereiro de 2018, na Bélgica. Esta oportunidade de uma vida nunca teria sido possível sem o professor Dr. André Moitinho, ao qual agradeço ter-me dado a conhecer a existência destes *workshops*. Agradeço de igual forma ao Professor Dr. José Augusto, que me ajudou com a documentação necessária para me candidatar e ao meu orientador, Dr. Paulo Gordo, pela carta de recomendação que me escreveu. Graças a este workshop adquiri todos os conhecimentos base e confiança necessários para escrever esta dissertação.

Este trabalho é o cuminar do atribulado percurso académico de uma aluna trabalhadora estudante que muitas vezes pensou em desistir. O meu agradecimento final vem para aqueles que nunca desistiram de mim e que, de igual forma, nunca me deixaram desistir. Agradeço assim aos meus avós, pelo apoio incondicional que me deram ao longo da minha vida académica, aos meus pais, ao meu irmão e ao meu namorado, que me tornaram a pessoa resiliente que sou hoje. Nada disto teria sido possível sem o vosso amor, paciência e suporte emocional.

Resumo

A presente dissertação de mestrado foi desenvolvida em parceria com a empresa Omnidea Lda e tem como principal objectivos desenvolver e testar um protótipo de desenvolvimento de um star tracker de baixo custo para ser implementado em missões cubeSat. Um star tracker é um instrumento que permite extrair a atitude de um satélite, ou seja, a direcção para a qual este está a apontar. O star tracker é actualmente o sensor mais caro e mais exacto para extracção da atitude e o seu funcionamento assemelha-se ao de uma câmara digital. A determinação de atitude com um star tracker inicia-se quando uma imagem das estrelas no campo de visão do sensor de imagem é extraída. Posteriormente, um algoritmo identifica as estrelas com base nas distâncias relativas entre estrelas vizinhas e calcula a atitude. Este processo é extremamente rápido, não durando mais do que fracções de segundo.

Dada a expansão actual do mercado dos satélites inferiores a 100Kg, e em específico dos cubeSats, é de alta prioridade construir um star tracker que possa ser usado em missões de baixo custo desenvolvidas por pequenas empresas e universidades.

Na presente dissertação é apresentada uma revisão do estado de arte de sensores sensores de controlo de attitude para satélites, de forma a definir os requisitos pretendidos para o protótipo de desenvolvimento do star tracker. Com base na revisão feita, foi escolhido um sensor de imagem CMOS monocromático de 10MP fabricado pela Aptina (actual On-Semiconductor), juntamente com um sistema de lentes de baixo custo com um f-number de 1.6 e 35mm de distância focal. O sistema final tem um campo de visão de cerca de 10°. A carta electrónica associada ao sensor de imagem faz interface com um driver (ArduCAM USB camera shield), que permite a sua fácil implementação. Com o ArduCAM USB camera shield, o sensor de imagem pode ser manipulado através de ligação USB com um computador portátil ou computadores embebidos como o Beaglebone ou o Raspberry Pi. De forma a entender os limites de funcionamento do sensor para a sua aplicação enquanto star tracker, foi feita uma caracterização do seu ruído e um estudo teórico de fotometria estrelar.

O algoritmo seleccionado para identificação das estrelas é o Tetra, que é um recente algoritmo desenvolvido por dois estudantes da universidade do MIT, nos Estados Unidos. Este algoritmo tem como base de funcionamento o algoritmo utilizado pelo Astrometry.net, uma ferramenta online de indentificação de corpos celestes. O algoritmo tetra utiliza como base de dados um catálogo com 9100 estrelas. De forma a testar o seu funcionamento com um catálogo mais amplo e recente (Hipparcos), com mais de 118218 estrelas, foram feitas alterações ao código do algoritmo Tetra. A performance do star tracker é testada com ambas as versões do algoritmo para diversas imagens.

Numa fase final, o sensor de imagem e sistema de lentes foram ligados a um computador embebido Raspberry Pi. Um programa que une o código de funcionamento da ArduCAM USB camera shield ao algoritmo Tetra permite que o star tracker funcione de forma autónoma, através do envio de comandos de um portátil para o Raspberry Pi via Wi-Fi. Uma caixa de protecção e suporte ao sensor de imagem e à board ArduCAM USB camera shield foi desenvolvido na fresadora manual da faculdade, de forma a garantir a protecção dos componentes electrónicos mediante exposição ao ambiente exterior.

O star tracker foi testado em ambiente nocturno, o mais longe possível de poluição luminosa, de forma a comprovar o seu funcionamento e estudar a sua performance.

Palavras-chave: star tracker, atitude, cubesats

// Abstract

This master thesis was developed in partnership with Omnidea Lda. The main goal of this work is to develop (design and test) a prototype of a low cost star tracker for cubeSat missions. A star tracker is an instrument that can be assembled in satellites and allows to extract their attitude. The attitude is defined as the direction the cubeSat is pointing, with respect to a reference. The determination of a satellite's attitude is made using the stars present in the field of view of the star tracker. An algorithm identifies stars by comparing the distance between neighbor stars with an internal catalog and computes attitude. The star tracker is currently the most expensive and accurate type of sensor for attitude determination. Since the cubeSats market have been evolving exponentially in the past years, it became high priority and mandatory to develop a star tracker that can be used in low cost missions developed by small companies and universities.

In this thesis a literature research about attitude determinations sensors was made. From this review, the star tracker's requirements were defined. Based on the requirements, the main parts chosen for the star tracker prototype to be developed consisted on a 10MP monochromatic CMOS image sensor from Aptina (actual On-Semiconductor), along with a low-cost lens system with a f-number of 1.6 and 35mm of focal distance. Considering the lens system's focal length and the sensor's size, the final system has an approximated field of view of 10°. The image sensor's board is connected to a driver (ArduCAM USB camera shield), that allows its easy implementation. The image sensor can be controlled through an USB connection to a laptop, Beaglebone or Raspberry Pi. At this point, a theoretical study of stellar photometry is made, along with an experimental study of the image sensor's noise. Both studies complement each other and allow to understand how the sensor behaves as a star tracker.

The selected algorithm to star identification is Tetra, a recent algorithm developed by two MIT students in the United states. This algorithm uses an internal catalog of 9100 stars. In order to test the code with a more extended and recent catalog (Hipparcos), with about 118218 stars, alterations were made in the original Tetra algorithm. Both versions of the algorithm are tested with several images in a way to study their performance.

In a final phase, the image sensor and lens system are finally implemented in Raspberry Pi. A program that links the ArduCAM USB camera shield's code and Tetra algorithm allows the star tracker to work in an autonomous way, using only user commands sent through a laptop to Raspberry Pi using Wi-Fi. A protection case was design and built using the university's manual milling machine. This case allows the electronic components to be safe when the star tracker is exposed to the humidity of the nocturnal environment.

The star tracker was finally tested in the night sky, the furthest possible from light pollution, in order to test is performance and functioning.

Keywords: star tracker, cubesats, attitude

Contents

Agradecimentos . . . v
Resumo . . . vii
Abstract . . . ix
List of Tables . . . xv
List of Figures . . . xvii
Nomenclature . . . xix
Acronyms . . . xxi

1 Introduction **1**
1.1 Overview . . . 1
1.2 Motivation and Goals . . . 1
1.3 Thesis Outline . . . 2

2 Background **3**
2.1 CubeSats . . . 4
2.1.1 CubeSats Design . . . 5
2.2 Introduction to attitude determination . . . 7
2.3 Attitude measurement sensors . . . 8
2.4 Coordinate systems commonly used to extract attitude . . . 9
2.5 Theory behind general attitude changes . . . 10

3 Star trackers overview **17**
3.1 Star tracker's architecture . . . 17
3.1.1 Star properties and their relation with a star tracker's optical system . . . 17
3.1.2 Image detection sensor and image formation . . . 20
3.1.3 Noise analysis . . . 22
3.2 Space environment impacts on star tracker performance . . . 23
3.2.1 Space radiation . . . 24
3.2.2 Radiation effects on electronic components in LEO . . . 25
3.3 Market survey and desired requirements for the star tracker to be developed in this work 25

4 Imaging sensor and optical system **29**
4.1 Image sensor specifications and USB ArduCAM camera shield . . . 29
4.1.1 Register values for the MT9J001 image sensor . . . 30
4.2 Experimental characterization of MT9J001 noise . . . 31
4.2.1 Determination of the charge conversion efficiency . . . 31
4.2.2 Determination of the readout noise . . . 34

4.2.3 Determination of the dark current 35
4.3 Lens system selected for testing 36
4.4 Prediction of the maximum magnitude values detected by the sensor 38
4.4.1 Approximations and assumptions for measuring flux of an A0 V star type 38
4.4.2 Determination of magnitude maximum limits detected by the image sensor outside Earth's atmosphere 39
4.5 Final balance 39

5 Tetra Algorithm **41**
5.1 Modes of operation for a star tracker 41
5.2 Tetra algorithm 41
5.2.1 Storage of the internal catalog in hash tables 42
5.2.2 Process of stars identification 43
5.2.3 Output 44
5.3 Migration to Hipparcos catalog 44
5.3.1 Alterations to Tetra algorithm 45
5.4 Algorithm's performance 45
5.4.1 Testing conditions 45
5.4.2 Results 45

6 Mechanical and electronic system implementation **49**
6.1 Raspberry Pi implementation 49
6.1.1 Configuring Raspberry Pi and its communication with the external computer . . 50
6.1.2 Programming ArduCAM board to function as a star tracker and share measurements with the external computer 50
6.2 Mechanical design of the star tracker's structure 51

7 Final tests and results **55**
7.1 Nocturnal testing 55
7.1.1 Test plan and required test conditions 55
7.1.2 First test results 56
7.1.3 Second test results 58
7.2 Discussion 59

8 Conclusions **61**
8.1 Achievements and difficulties 61
8.2 Final Outcome 62
8.3 Future work 63

Bibliography **67**

A Register addresses and default values for MT9J001 and MT9J003 **71**

B Algorithm codes **81**
B.1 C code to filter information from Hipparcos catalog 81
B.2 Tetra algorithm with alterations from this work 83

C Instruction manual for Raspberry Pi **101**

List of Tables

2.1 Categorization of small satellites according to their mass . . . 3
2.2 Categorization of Orbits for Earth satellites . . . 4

3.1 Flux calibration values for an A0 V star . . . 19
3.2 Characteristics of the star tracker from Berlin Space technologies . . . 26

4.1 MT9J001 image sensor specifications . . . 29
4.2 Specifications from the laser beam used to characterize the noise of the image sensor . . . 32
4.3 Specifications from the photo-detector used to measure the incoming flux . . . 33

5.1 Tetra algorithm performance analysis: right ascension values . . . 46
5.2 Tetra algorithm performance analysis: declination values . . . 47
5.3 Tetra algorithm performance analysis: roll values . . . 47
5.4 Tetra algorithm performance analysis: right ascension values . . . 48
5.5 Tetra algorithm performance analysis: mismatch probability . . . 48

6.1 Raspberry Pi Model 3 B+ Specifications . . . 49

7.1 GPS locations of the places chosen to test the star tracker . . . 55
7.2 Maximum magnitude values detected by the star tracker . . . 57
7.3 Identified frame results . . . 58
7.4 Combinations of gain and integration times used to test the star tracker. . . . 59

List of Figures

2.1 ESAT: A fully functional cubeSat developed for education by Theia Space 4
2.2 Representation of a cubeSat system 5
2.3 Primary structure of a 1U cubeSat and a 3U cubeSat 6
2.4 Flow chart representing the attitude determination and control system 7
2.5 Coordinate systems used to determine attitude 10
2.6 Center of mass of a satellite 11
2.7 Changes in trajectory by application of an external force 12

3.1 Star tracker design and its subsystems 17
3.2 Johnson-Cousins UBVRI filter curves. 18
3.3 Spectral sensitivity curve for a Silicon CMOS 19
3.4 Point spread function of a star in an image sensor 21
3.5 Representation of the relation between signal and noise 21
3.6 Materials commonly used in lens glass and how several doses of radiation affect them . . 24
3.7 Radiation effects on a star tracker's image 25
3.8 Small star tracker integrated in an ADCS 26

4.1 Block diagram of the electronic connection between the MT9J001 board and the ArduCAM USB camera shield board 30
4.2 Image sensor MT9J001 mounted on ArduCAM USB camera shield board 30
4.3 Experimental setup of the integrating sphere and laser beam 32
4.4 MT9J001 image sensor mounted in an integrating sphere's cavity 33
4.5 Plot showing mean counts vs variance counts for several images extracted by the MT9J001 with the laser beam on. 34
4.6 Plot showing mean counts vs variance counts linear relation for several images extracted by the MT9J001 with the laser beam on. 34
4.7 Dark current for the image sensor MT9J001 36
4.8 Commercial lens system to be used in the star tracker 36
4.9 Representation of the FOV measurement 37
4.10 Zoomed frames before and after the focusing of the system. 37
4.11 Approximation used in Johnson-Cousins system to measure flux for an A0 V type star. . 38
4.12 Prediction of the maximum magnitude detected by MT9J001 sensor for different integration times. 39

5.1 Sky coverage for Yale bright star catalog (BSC5) 42
5.2 Pixelized star detected in an image 43
5.3 Virtual coordinate system created in the image by Tetra 43

5.4 Image before and after being analyzed by Tetra algorithm. 44
5.5 Picture analyzed with both catalogs 46

6.1 Schematic functioning of the star tracker implemented in Raspberry Pi 50
6.2 Fluxogram for the star tracker's code 51
6.3 Raspberry Pi Model 3 B+ and its official case 51
6.4 Star tracker's case designed using CAD software. 52
6.5 Case designed to protect the image sensor and ArduCAM USB camera shield board with the help of a manual milling machine 52
6.6 Star tracker developed in this work. 53

7.1 Frame extracted with the star tracker after tetra algorithm 57
7.2 Magnification of Vega star and its analysis using Thorcam 58

8.1 Lens system proposed to be tested in the future 64
8.2 Spot diagram of the lens system proposed to be tested in the future 64
8.3 Electronic system proposed to be implemented in the future 65

Nomenclature

Greek symbols

α Right ascension.

δ Declination.

η Rate of change in angular momentum due to an external torque.

ι Dark current.

λ Wavelength.

Ω Rotation of the orbital plane due to a change in the moment of momentum.

ω Angular speed.

Φ Flux.

ϕ Roll.

ψ Yaw.

σ Standard deviation.

θ Pitch.

υ Noise.

ξ Rate of change in a trajectory's direction due to an external force.

Roman symbols

$\vec{F}$ Force vector.

$\vec{L}$ Angular momentum vector.

$\vec{M}$ Moment of momentum vector.

$\vec{p}$ Linear momentum vector.

$\vec{r}$ Position vector.

$\vec{T}$ Torque vector.

$\vec{v}$ Velocity vector.

c Velocity of light

d Diameter

E Energy

f Focal length

h Planck's constant

I Inertia.

m Mass of a particle

m_T Mass of a closed surface

n Number of photons

P Generic point in space

t Time

U, B, V, R, I Ultraviolet, Blue, Visible, Red and Infrared wavebands

C Center of mass.

O Origin.

QE Quantum efficiency.

x, y, z Cartesian components.

Subscripts

i Index used for sommatory.

n Normal component.

x, y, z Reference to the cartesian components' direction.

bw Bandwidth.

C Reference to the center of mass.

DN Dark noise.

ecef, orb, eci Reference to the coordinate systems ecef, orb and eci.

ext Reference to external.

O Reference to origin.

P Reference to position P.

ref Reference condition.

RN Readout noise.

SN Shot noise.

Acronyms

ADC Analog to digital converter.

ADCS Attitude determination and control system.

ADU Analog to digital units.

BSC5 Yale's bright star catalog.

CCD Charged coupled device.

CCE Charge conversion efficiency.

CMOS Complementary metal-oxide semiconductor.

COTS Custom of the shelf.

ECEF Earth-centered, Earth-fixed.

ECI Earth-centered inertial.

EEE Electrical, electronic and electro-mechanical.

FOV Field of view.

fps Frames per second.

FWMH Full width half maximum.

GEO Geostationary orbit.

GNSS Global navigation satellites system.

GPS Global positioning satellites.

GSO Geosynchronous orbit.

HEO High elliptical orbit.

ID Identification.

LEO Low earth orbit.

LET Linear energy transfer.

MEO Medium earth orbit.

OBC On-board computer.

OCS Orbital coordinate system.

PSF Point spread function.

QE Quantum efficiency.

ROI Region of interest.

SEE Single event effects.

SNR Signal to noise ratio.

SPE Solar particle events.

SSH Secure shell.

TID Total ionization dose.

Chapter 1

Introduction

The purpose of the work carried out in this master thesis is to develop and test a low-cost star tracker using only commercial parts. This document summarizes the study made on star trackers and also the steps taken in order to achieve the main goal.

1.1 Overview

The star tracker is one of the several types of sensors that can be part of the attitude and determination control system in a satellite. A star tracker can be described in a simple way as a digital camera with the capability to extract attitude with high accuracy using only the stars that are visible to its field of view. This is traditionally a very expensive device that is hard to miniaturize without losing performance quality.

This work focus in the development of a star tracker for cubeSats, a popular and low cost type of small satellite. Since the star tracker is the best device to be used in space for determining position with accuracy, it is important and high priority to create one to be used in cubeSat missions, mainly due to the lack of accurate and precise attitude systems in this type of satellites.

Some achievements have already been made by some companies in order to miniaturize them to be used in smaller satellites. However, the price tag is still too much high to justify their use in a simple cubeSat mission. This high cost is mostly because of the EEE components present in the star tracker, that require high resistance to radiation in order to maintain good functioning. Since new and more evolved COTS technology is been made every year, some actual studies suggest there are already in the market COTS components that can guarantee the development of a star tracker with satisfactory performance and adequate cost to cubeSats' developers.

1.2 Motivation and Goals

The main goal of this work is to show that it is possible to develop a star tracker which price is low enough to adjust to cubeSats mission's budgets. The low price can be achieved using new COTS components available in the market and simple laboratory tests. The main inspiration for this work was Infante, the Portuguese cubeSat. With the number of units growing, the need for a more rigorous extraction of the attitude is needed. Yet the mission budget stills too tight to invest in a commercial star tracker. For that reason, Omnidea felt inspired to propose the creation of a low-cost star tracker.

A simple design is proposed and developed in a Raspberry Pi, using a commercial image sensor from

OnSemiconductor and a highly sophisticated new algorithm called Tetra. Some improvements are also suggested having in account the experimental tests made with the developed model.

1.3 Thesis Outline

The structure of this work is organized as follows:

Chapter 2

Chapter two introduces satellites and its miniaturization until the arrival of cubeSats, based on literature research. Since this work focus on an instrument present in some attitude and determination control subsystems, a deep study on this subsystem is also presented, because in order to understand how attitude is extracted, first it is important to know what it means and how it relates with the satellite's position.

Chapter 3

A review on the architecture of star trackers is made on this chapter. Star trackers are introduced as instruments composed by an optical system, an electronics and signal processing system and an image sensor connecting both. The difficulties behind the construction of low-cost star trackers are also presented at the end of this chapter.

Chapter 4

Chapter four reviews the whole work behind the development of the optical system. An experimental characterization of the image sensor's noise is presented. The optical system used for testing is introduced. It is also made in this chapter a photometry study in order to predict the star magnitude the star tracker may be able to achieve outside Earth's atmosphere.

Chapter 5

This chapter reviews the work made on the star tracker's algorithm. The algorithm was changed in order to work with a more expansive catalog of stars. A comparison between both catalogs was experimentally made. The results are presented at the end of the chapter.

Chapter 6

Chapter six describes the work made in order to implement the star tracker into a single board computer, the Raspberry Pi. A case to protect the image sensor was designed and constructed. The final star tracker design is shown.

Chapter 7

The results obtained with the implemented star tracker are presented in this chapter. The information regarding the nocturnal tests is also reviewed.

Chapter 8

Last but not least, this final chapter presents the conclusions from this work. Future improvements to be made in the star tracker are also suggested in this chapter.

Chapter 2

Background

On October 4th 1957, a small aluminum sphere made history by becoming the first artificial object in space. Sputnik-1 had approximately 58 centimeters diameter and 83 kilograms and it was launched by the Soviet Union as a way to show the United States its technological power. The only function Sputnik 1 had was to transmit a beep sign that could be caught by any radio amateur on Earth. This was the first satellite orbiting the Earth and it was responsible for the beginning of the space age, that would take humanity to the moon in 1969[1].

Since Sputnik's launch, space technologies have been evolving exponentially. A big part of this evolution is due to the increasing availability of electrical, electronic and electro-mechanical (EEE) components capable to resist radiation, vibration, severe thermal changes and able to perform in vacuum. The outbreak of integrated circuits made the EEE components became much smaller and lighter[2]. As a direct consequence, satellites also became lighter and less voluminous, once their components became miniaturized. Nowadays, satellites are categorized according to its total mass.

Types of Small satellite	Usual mass limits
Minisatellites	Mass between 100kg and 500kg
Microsatellites	Mass between 10kg and 100kg
Nanosatellites	Mass between 1kg and 10kg
Picosatellites	Mass between 100g and 1kg
Fentosatellites	Mass between 10g and 100g

Table 2.1: Categorization of small satellites according to their mass. This categorization is not an official convention and sometimes these limits can vary[3].

The manufacturing of small satellites made the launch costs decrease, since it is possible to send more than one satellite to space in only one launch. Space missions have become more frequent. Currently, it is also common to have more than one miniaturized satellite in space for a single mission, in order to form a constellation. A constellation of miniaturized satellites allows data communications and gathering of scientific data from multiple points.

Custom off the shelf (COTS), technology was not used in space in the past because of their impossibility to resist space environment. However, in low orbits the environment is not so hostile for short missions and some modern COTS components are capable to perform with enough reliability to achieve the mission's goals in the mission time frame. This is why the current trend in small satellites is set on CubeSats. A CubeSat is a satellite constructed with one or more aluminum alloy cubes and COTS tech-

nology. Each cube is designed to provide 10 cm^3 of useful volume and weight less than $1,33$ kg[4][5]. A CubeSat can be categorized as a picosatellite, nanosatellite or microsatellite according to its total mass.

2.1 CubeSats

CubeSats were initially proposed in 1999 by Jordi Puig-Suari, a professor at California Polytechnic State University and Bob Twiggs, a professor at Stanford University's Space Systems Development Laboratory[4][5]. The cubeSat project would allow affordable access to post graduate students to project, design, build and test operations in space. The goal was to create a satellite with similar capabilities as Sputnik-1, using COTS technology for their electronics and structure.

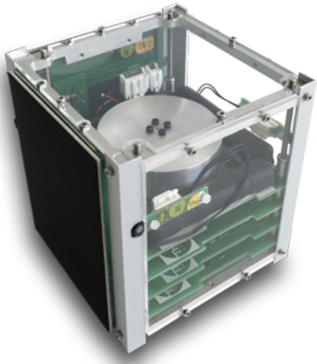


Figure 2.1: ESAT: A fully functional cubeSat developed for education by Theia Space. Image courtesy of Theia Space

CubeSats quickly became a low cost mean to get a payload into orbit not only for universities but also for governmental agencies and companies who recognized that the standardized design of cubeSats could help reduce the costs of scientific investigations and technology development. CubeSats are easier and faster to develop due to its standardized structure. Plus they don't need to fulfill all quality requirements larger satellites do. Nonetheless, they need vibration and structural analysis to ensure the payload can resist launch.

CubeSats usually fly in Low Earth Orbit (LEO), which is the easiest and cheapest orbit to get into. Relatively small rockets can achieve LEO.

Types of Orbits	Usual altitude limits related to Earth
High Elliptical Orbit (HEO)	$>$ 35,786 Km
Geosynchronous and Geostationary Orbits (GSO/GEO)	Exactly 35,786 Km
Medium Earth Orbit (MEO)	2000 - 35,786 km
Low Earth Orbits (LEO)	80 - 2000 km

Table 2.2: Categorization of orbits for satellites that orbit around the Earth. Geosynchronous and Geostationary Orbits (GSO/GEO) differ in inclination and both have a translation period equal to the rotational period of Earth. GEO orbits are perfectly circular orbits around the Earth without inclination[6].

According to the Nanosats & Cubesats Database [3], up to August 2018 there were more than 850 cubesats launched. Small companies are now developing bigger units cubeSats in order to expand their

industry and take payloads into orbit in a shorter time. There are more cubeSats predicted to be launched in a near future like Infante, the portuguese microsatellite. Infante is a portuguese satellite being fully developed in Portugal and it is scheduled to be launched in 2020. This satellite is predicted to be a 32U cubeSat and will be around 50kg[7]. Infante will be launched into a LEO orbit. Its exact altitude is yet to be determined, although it is expected to be 500km. Infante has a design life from 6 month to 3 years and is the forerunner to a constellation of 12 spacecraft in LEO for Earth observation and telecommunications services.

2.1.1 CubeSats Design

Like any other spacecraft, a cubeSat can be divided in two major sections: the bus, also called service module, and the payload. The payload depends on the nature of the mission and it contains the scientific instruments needed to fulfill the mission's goals. The bus contains the control subsystems that allow the payload to perform its mission.

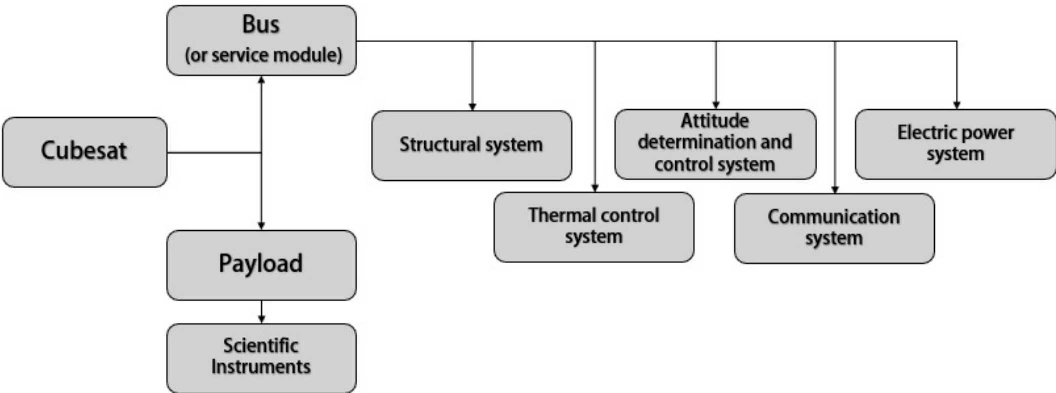


Figure 2.2: Representation of a cubeSat system, which is the same for any kind of satellite.

Structural system

A 1U cubeSat has a volume of only one cube (10 x 10 x 10cm^3). It is possible to combine more cubes in order to create a larger cubeSat. 3U and 6U cubeSats are the most popular sizes nowadays, although there are other configurations that can be developed. The number that precedes the letter U represents the number of units that compose a cubeSat.

The structural system can be considered divided in a primary structure and a secondary structure. The cubes in a cubeSat are shaped by a primary structure, or chassis, that serves as a main support. The failure of the primary structure is usually catastrophic for the mission, since as a rule it affects the whole systems or most part of them. The secondary structure serves as support for instruments that need to be accommodated outside the cubes, as can be the case of solar panels. A failure in the secondary structure usually affects the overall mission, but generally is not catastrophic.

The cost of a mission is highly proportional to the weight, so the material to be used in the structural system should be lightweight as possible. It is also important to have the hostile space environment regarding vacuum, radiation and thermal variations in account. Materials should be resistant enough to support all these adverse conditions. All the materials used in the structure must have the same thermal expansion coefficient as the deployer, in order to prevent jamming [8]. The primary structure of a cubeSat is usually made of aluminum alloy, usually 6061, 5052, 7075 or 7071 [9]. However, there has been

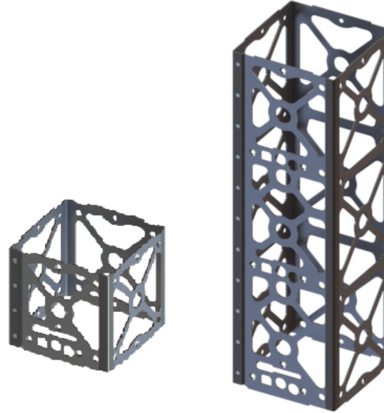


Figure 2.3: Primary structure of a 1U cubeSat (left) and a 3U cubeSat (right) made of Aluminum 5052-H32. Chassis Walls are available in multiple sizes. Image adapted from *Pumpkin, Inc* company website

developments in 3D printed chassis for cubeSats with Polyether Ether Ketone[10]. The Polyether Ether Ketone is a strong thermoplastic that offers stability and temperature resistance. The cubeSat market provide several primary structures for sale, specially the popular cubeSat sizes like 3U and 6U.

Electrical power system

The electrical power system controls power generation and manages the consumption of stored power. The power in a cubeSat can be stored in either primary or secondary batteries. Primary batteries are charged once before launch and secondary batteries are rechargeable. The power is usually generated by photovoltaic cells or solar panels placed outside the structural system[11].

Thermal control system

The cubeSat has an intrinsic temperature range that allows the components to function effectively. When in orbit, the heat from the Sun's radiation and the one reflected from the Earth, may cause an overheat of the electrical components. The temperature inside the cubeSat is measured with temperature sensors placed on critical components. Temperature changes can commonly be predicted if the orbit of the cubeSat is well known, which allows the temperature inside the cubeSat to be controled by the thermal control system. Nonetheless, the components should be chosen according to the temperature changing factors for determined orbit. To determine which thermal management components should be chosen, an analysis and simulation of the cubeSat's thermal model should be made before launch.

The temperature control is achieved by using reflectors, thermal insulation and heaters. Thermal insulation reduces the heat transfer between objects in thermal contact and/or in range of a radiation source. Surface coatings are placed in the structure. Multiple layer blankets are installed in places that don't receive direct sun light. These multiple layer blankets are not placed in the structure because in small sizes its not very efficient in protecting a body against direct sun radiation [12]. They can be replaced by low emissivity coatings, in order to preserve the budget and use less volume.

Communication system

The communication system allows the transmission of data and telemetry from the cubeSat to the ground control, on Earth. It is also the system responsible to receive commands from Earth. Communication between Earth and the cubeSat is made in the radio spectrum, between 30MHz and 40GHz[13]. The received signal power on Earth is inversely proportional to the square distance between the cubeSat and the Earth. This is why deep space satellites need dish antennas. A parabolic dish antenna is quite difficult to integrate in a cubeSat, because of its size and weight. Since cubeSats fly in LEO, lower gain patch antennas can be used for communication systems. The low directionality of a patch antenna allows the cubeSat to maintain communication even when it is tumbling.

Attitude determination and control system

Attitude can be defined as the three-dimensional orientation of a spacecraft's main structure with respect to a specified reference frame. The attitude determination and control system (ADCS), measures the attitude of a spacecraft in real time and if needed, stabilizes and orients it in desired directions. The orientation required for a cubeSats's structure is determined by the mission and will frequently be related to an Earth-based frame of reference. The accuracy needed for the measurement will be set by the payload[14].

The ADCS board in a cubeSat is composed by a set of sensors and actuators. The sensors provide the attitude information of the cubeSat while it is orbiting the Earth. This information is processed by the On-Board Computer (OBC) and sent to the ground control. If needed, the cubeSat can be stabilized using actuators like magnetorquers or rotation wheels.

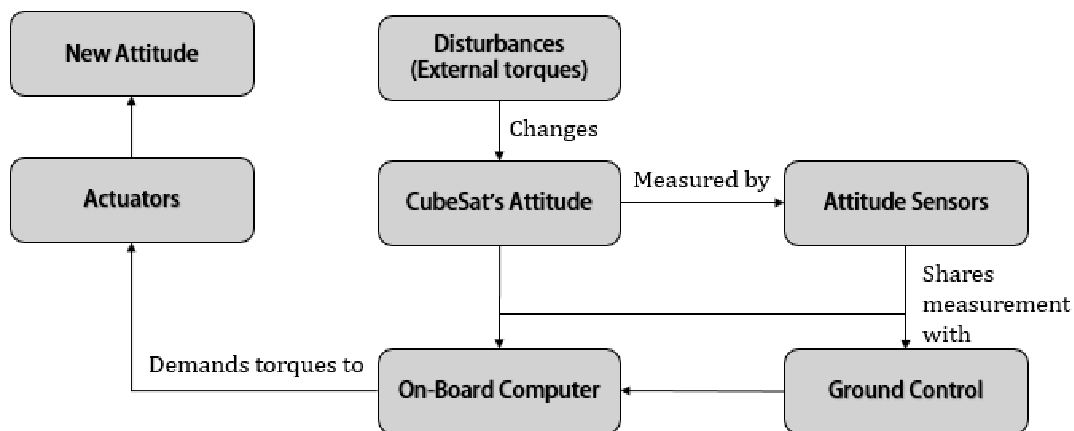


Figure 2.4: Flow chart representing the attitude determination and control system

2.2 Introduction to attitude determination

As addressed before, a cubeSat can maintain its communication with Earth much easier than a conventional spacecraft, due to its small size and low orbit. Due to this fact, the stabilization needed for a cubeSat is not often as strict as the stabilization needed for a bigger size satellite. However, the cubeSat still has scientific payload that sometimes needs to be pointed into the right direction. The pointing is achieved by stabilizing the cubeSat when it is tumbling. It is pretty logical to understand that a cubeSat

with more units carries more volume of scientific payload than a 1U cubeSat and that the pointing accuracy needed may be more rigorous. A cubeSat may only need to be stabilized in two axis or it may need to be stabilized in its three axis. The stabilization in only two axis is also called spin stabilization. The spin stabilization is achieved by putting the cubeSat spinning around one of its axis[15]. Three axis stabilization implies that the cubeSat is maintained fixed in the desired orientation without any rotation. In order to know for which direction the cubeSat needs to point and where it is at a determined moment pointing, a measurement of its real time attitude is made using attitude measurement sensors. There are many ways to specify the attitude, such as Euler angles, direction cosines matrix and quaternions. The easier way is to use the three Euler angles, because of its physical meaning: yaw (ψ), measures the rotation along the z axis, pitch (θ), the rotation along the y axis and roll (ϕ) the rotation about the x axis. Despite the method used to represent attitude, information measured by the ADCS should be extracted with accuracy and simplicity.

2.3 Attitude measurement sensors

There are two categories of sensor which complement one another: the reference sensors and the inertial sensors[16]. Inertial sensors have the ability of measure changes in attitude in a continuously way, but they need a reference point to fully extract attitude information. The reference point direction needed is provided by reference sensors. Reference sensors measure the direction of an object such as the Sun, or a star. They are highly affected by periods of eclipse because during these periods the information won't be available. There should be at least three pieces of information in order to relate the satellite's position to a referenced axis. With the exception of star sensors, reference sensors only measure the direction of a single vector, the reference vector. This means reference sensors can't make a full extraction of the attitude information alone, however they can provide calibration for inertial sensors. Reference sensores calibrate inertial sensors at discrete times and will then save in memory the reference object's direction until the next calibration. This allows a period in an eclipse to be covered [14]. A Kalman filter can also be implemented in order to minimize attitude measurements errors. The inertial sensors commonly used in cubeSats are the gyroscope or the accelerometer. A set of three orthogonal gyroscopes are able to measure only the changes on the three-dimensional components of a cubeSat or any spacecraft's angular velocity. A fourth measurement at a skew angle can be made to avoid a single point failure. There are many reference sensors to choose from. A good system accuracy implies a good accuracy from the reference sensors and a low degradation rate from the inertial sensors. The accuracy required is normally set by the payload, its pointing direction and the required measurement accuracy. There is an ultimate accuracy of measurement that is determined by the object of reference used by the reference sensor.

Sun sensors

A sun sensor uses the Sun as reference object, providing a well-defined vector because of its position towards Earth. The simplest form of this type of sensor work as a mere presence detector that determines if the sun is in a specified field of view (FOV) or not, which is quite useful for pointing solar arrays. Sun sensors can provide sufficient accuracy for cubeSats[17]. Unfortunately, this type of sensors are highly affected by eclipses, when the Earth is placed between the Sun and the cubeSat.

Earth sensors

An Earth sensor uses the Earth as a reference object and is typically used in LEO missions[14]. By sensing the position of the Earth's horizon, using infrared, it can provide the meanings to determine the Nadir vector. The Nadir vector corresponds to the x_{orb} vector in the OCS referential frame previously defined. Earth sensors are not as accurate as others because of the existence of the CO_2 layer in the atmosphere. This layer extends for tens of kilometers, which causes the Earth horizon to be fuzzy[17].

Magnetometers

Magnetometers measure the direction and the strength of the Earth's magnetic field. Unfortunately, the abnormalities of the magnetic field bring limitations to this sensor for attitude sensing. Magnetometers can only be used in LEO[14].

Global Navigation Satellites Systems (GNSS)

GNSS is a system that uses other satellites like the Global Positioning Satellites (GPS), to provide spatial positioning. It is commonly used to determine the orbital position, but it can also be used to determine attitude, using a set of three or more antennas as distant as possible from each other.

Star sensors

A star sensor can track more than one star within its field of view, which can provide the three vectors needed to obtain the complete attitude information needed. There are three categories of star sensors: the star scanners, the star mappers and the star trackers. Star scanners are mounted on a spin stabilized spacecraft and the star mappers and the star trackers, mounted on a three-axis stabilized spacecraft. In the star scanner, the spin of the spacecraft provides means for the sensor to scan the sky. The characteristics of the stars passing through the FOV can be compared to a star directory in order to determine attitude. The star mapper locates and records the position of each star allowing the spacecraft's orientation about the sensor axis to be determined[14]. Finally, the star tracker uses one stellar image to select, locate and determine the spacecraft's attitude with precision, comparing the star's position with an existing star catalog.

2.4 Coordinate systems commonly used to extract attitude

In order to understand how attitude is measured and how the ADCS stabilizes and orients a spacecraft, an overview of the coordinate systems often used is needed. The Earth-centered inertial (ECI) coordinate frame has its origin at the center of mass of the Earth and it is the most used one to define celestial coordinates. Since it is an inertial referential, it does not accelerate nor rotate with respect to the rest of the Universe. The z_{eci} axis intercepts the true North Pole and the x_{eci} and y_{eci} axis lie in the equatorial plane of the Earth. The x_{eci} axis has the direction of the Vernal mean equinox of epoch J2000. The Vernal equinox of epoch J2000 is defined as the first equinox of the Spring in the North Hemisphere at 12:00 Terrestrial Time on 1 January 2000. The y_{eci} axis is ortogonal to x_{eci} and z_{eci}.[18] Right ascension, α, is measured from the Vernal equinox and declination, δ, is measured from the celestial equator. The ECI coordinate frame is often used to work with Euler angles. There is yet another coordinate system to have in account. The Earth-Center, Earth-Fixed (ECEF) coordinate frame has the same origin as the ECI, but all the axis remain fixed with respect to the Earth [19]. The x_{ecef} axis points from the origin

to the intersection of the equator with the International Reference Meridian, defining 0° longitude. The y_{ecef} axis points from the origin to the intersection of the equator with the 90° East Meridian. The z_{ecef} axis points to the geographic North Pole, like the ECI system. Since this work focus in measurement of the orbital plane with right ascension and declination, the ECEF reference frame wont be taken into account, although it is important to be refereed, since it allows to convert celestial coordinates to GPS coordinates.

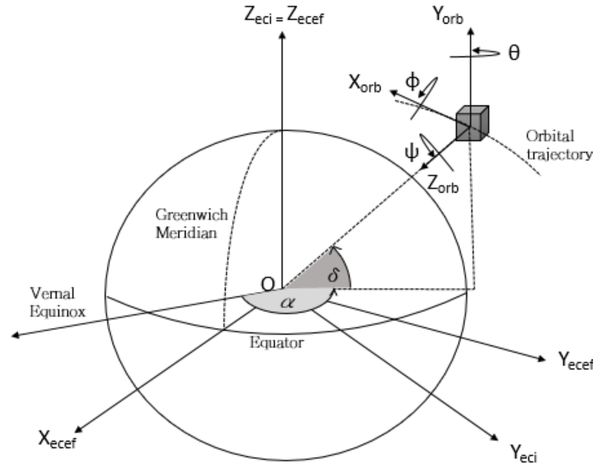


Figure 2.5: Coordinate systems used to determine a spacecraft's attitude. The three Euler angles are represented in the OCS and they all follow the right hand rule.

The orbital coordinate system (OCS), is the coordinate system whose origin is placed in the center of mass of a cubeSat or any spacecraft orbiting the Earth. In this coordinate system, the z_{orb} axis points right to the center of mass of the Earth, i.e. the origin of the ECI reference frame. The x_{orb} axis' direction is the same as the angular velocity of the cubeSat. Finally, the y_{orb} axis is perpendicular to the orbital plane and orthogonal to the other two axis.

2.5 Theory behind general attitude changes

The determination of the attitude depends on the type of stabilization is chosen for a particular mission. In section it is going to be taken into account the three axis stabilization for a cubeSat.

The translation motion and the rotation motion of a satellite can be considered independent of each other. This is because the forces that cause a satellite to rotate along its own axis passing through the center of mass are not dependent upon their direction of travel and the gravitational forces that determine their trajectory are not dependent on their attitude. This approximation makes possible to describe each type of motion separately, which simplifies the understanding of the dynamics[14].

A satellite can be considered a closed surface made of particles, with a center of mass C. The dynamics of this surface, can now be described in terms of its linear momentum, and its angular momentum, considering the dynamics of the center of mass. These two quantities lead to equations that describe the trajectory, which will lead to equations that describe the attitude motion. The properties of the center of mass makes it possible to separate the motion into the motion of the center of mass and the motion relative to the center of mass. Considering a generic particle inside the closed surface, its position relatively to the origin of a reference axis can be described using a position vector, $\vec{r}_{op}$. The continuous mass distribution of the closed surface, m_T, can then be related to the mass of the particle, m:

$$m_T \quad \vec{r}_{oc} = \int dm \quad \vec{r}_{op} \tag{2.1}$$

The vector $\vec{r}_{oc}$ is the position vector of the center of mass relatively to a generic coordinate system.

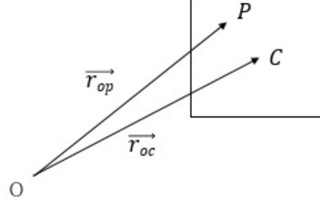


Figure 2.6: Definition of the center of mass for a satellite of mass m_T. The vector $\vec{r}_{oc}$ is the position vector of the center of mass C relatively to the origin of a reference axis. Considering a generic particle P inside the closed surface, its position relatively to the origin of a reference axis can be described using a position vector, $\vec{r}_{op}$.

Considering the coordinate systems previously described in section 2.2, the center of mass can be considered to match the origin of the OCS reference axis, which leads to $\vec{r}_{oc} = 0$:

$$\int dm \quad \vec{r}_{cp} = 0 \tag{2.2}$$

Since the velocity of the generic particle in consideration is $\vec{v}_{cp} = \frac{d\vec{r}_{cp}}{dt}$ considering the constants resulted from the derivative null, the previous equation leads to

$$\int dm \quad \vec{v}_{cp} = 0 \tag{2.3}$$

For trajectory purposes, the surface can be treated as an equivalent particle with the same mass considered for the surface, m_T. The equivalent particle is located in the center of mass, moving with its velocity, $\vec{v}$. The linear momentum of the satellite can be considered to be

$$\vec{p} = m_T \cdot \vec{v} \tag{2.4}$$

According to Newtonian mechanics, there are two ways for a satellite to change its linear momentum: by the application of external or internal forces and by the loss of mass. The loss of mass can happen during rocket propulsion. From now on, in order to simplify the understanding of the momentum changes, the mass of the particle is considered to be constant. This means the trajectory dynamics is going to be described without propulsion.

Linear momentum and perturbations to the translational movement

The application of external forces, $\vec{F}_{ext}$, to the satellite changes its linear momentum. The effect of this external forces will not only depend on its magnitude and direction but also on the satellite's magnitude of the linear momentum. The additional linear momentum, produced during a time dt can be described as a Newtonian equation:

$$\frac{d\vec{p}}{dt} = \vec{F}_{ext} \tag{2.5}$$

The external force can be divided in two components: $\vec{F}_x$ along the trajectory, and $\vec{F}_y$ normal to the

trajectory. Each component has a different effect on the satellite's linear momentum.

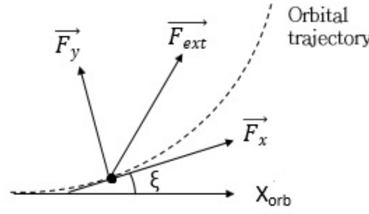


Figure 2.7: Trajectory changes due to an external force. ξ is the angle of the new trajectory's direction with the previous defined x_{orb} due to an external force, $\vec{F}_x$.

$\vec{F}_x$ changes the magnitude of the linear momentum, by changing the velocity of the spacecraft. Considering the time to change the orbital trajectory to be $t_2 - t_1$, the change in linear momentum can be given by

$$m_T(\vec{v}_{t_2} - \vec{v}_{t_1}) = \int_{t_1}^{t_2} \vec{F}_x \, dt \tag{2.6}$$

where $\vec{v}_{t_2}$ is the velocity at $t = t_2$ and $\vec{v}_{t_1}$ the velocity at $t = t_1$. If the external force has the same direction as the x$_{orb}$ axis, then $\vec{F}_y = 0$. In this case, the orbit trajectory remains the same and only the velocity's magnitude changes.

$\vec{F}_y$ changes the direction of the linear momentum by changing the direction of the velocity vector. The rate of this change can be described as

$$\frac{d\xi}{dt} = \frac{\vec{F}_y}{m_T \cdot \vec{v}} \tag{2.7}$$

where ξ is the angle of the trajectory with the previously defined x_{orb} axis. If the speed remains constant, $\vec{F}_x = 0$

$$m_T \cdot \vec{v}(\xi_{t_2} - \xi_{t_1}) = \int_{t_1}^{t_2} \vec{F}_y \, dt \tag{2.8}$$

Moment-of-momentum and perturbations to the orbital plane

The moment-of-momentum is a concept used very often in celestial mechanics and the study of attitude motion. The moment of momentum for a generic particle of mass m, $m \cdot \vec{h}_o$, can be described in reference to to the origin of a generic coordinate system, O, as

$$m \cdot \vec{h}_o = \vec{r} \times m\vec{v} \tag{2.9}$$

where $\vec{r}$ is the position vector of the generic particle from the origin of the referential to its location and $\vec{v}$ its velocity. This equation can also be used for the equivalent particle described before, considering the mass to be m_T. In this case, it is being considered the referential to be the ECI coordinate system, and the moment-of-momentum vector is normal to the orbit plane. The aggregate of all the moments of the momenta of the all particles in a satellite is defined as the angular momentum, $\vec{L}$. In a satellite, its angular momentum referred to the ECI coordinate system $\vec{L}_o$ is the sum of the moment of momentum

of its equivalent particle referred to the ECI coordinate system and its angular momentum referred to the center of mass, which corresponds to the origin of the OCS referential:

$$\vec{L}_o = m_T \vec{h}_o + \vec{L}_c \tag{2.10}$$

The moment-of-momentum of a satellite is only changed if there is a force acting on it with a moment $\vec{M}_o$ about the origin of the ECI reference frame:

$$\frac{dm_T \quad \vec{h}_o}{dt} = \vec{M}_o \tag{2.11}$$

This equation can be related to 2.5, and the conclusion is that the moment-of-momentum responds to a moment $\vec{M}_o$, the same way linear momentum responds to an external force $\vec{F}_{ext}$. The component of $\vec{M}_o$ normal to the orbit's plane changes the magnitude of the moment-of-momentum but not its direction. In this case, the orbit will remain in the same plane, but changes its shape. The component of $\vec{M}_o$ in the orbit's plane causes a rotation of the orbital plane, Ω,

$$\Omega = \frac{|\vec{M}_o|}{|m_T \quad \vec{h}_o|} \tag{2.12}$$

Angular momentum and perturbations to the rotational movement

If a force acts for only a short period of time (from t_1 to t_2), then it is called an impulse. The applied impulse is equal to the change in momentum that the force causes: $\vec{p}_{t_2} - \vec{p}_{t_1}$. The impulse is used in situations in which the movement caused by the force is insignificant, like a small collision.

The relationship between a torque and angular momentum is the same as the relationship between a force and linear momentum. The angular momentum of a rigid body is defined as a measure of the torque impulse that is needed to create rotational motion in a satellite which is the same as saying the impulse caused by a torque is equal to the change in angular momentum that it causes $\vec{L}_{t_2} - \vec{L}_{t_1}$.

As refereed previously, the angular momentum referred to the origin of the ECI reference frame is defined as the sum of all the moments of the moment for all the N particles inside the closed surface defined by a spacecraft.

$$\vec{L}_o = \sum_{i=1}^{N} \vec{r} \times m_i \vec{v} \tag{2.13}$$

The total angular momentum of a satellite in relation with its center of mass can only be changed in two ways. Otherwise it is conserved due to the conservation law of the angular momentum. The first way to change angular momentum is by the ejection of mass, as an analogy for the linear moment. The second way is by applying an external torque, $\vec{T}_{ext}$

$$\frac{d\vec{L}_c}{dt} = \vec{T}_{ext} \tag{2.14}$$

The torque can be divided in two components: the one normal to the movement and the one with the direction of angular momentum. The component of the torque with the same direction as the angular momentum only changes its magnitude. The component of the torque that is orthogonal (or normal), to the angular momentum causes a change in direction of the angular momentum towards the its direction.

The rate of this change is given by

$$\frac{d\zeta}{dt} = \frac{\vec{T}_n}{\vec{L}_c} \tag{2.15}$$

Mathematical definition of Euler angles and its relation with the angular momentum of a satellite

Inertia can be defined as the resistance of a rigid body to any change in its position and state of motion. Since a satellite is considered to be a rigid body, this can also be applied to satellites. The angular momentum of a satellite referred to its center of mass can be expressed in terms of its angular speed, $\vec{\omega}$. The angular velocity is equal to the ratio between the satellites' velocity and the radius of the orbit:

$$\vec{L}_c = [I_c] \cdot \vec{\omega} \tag{2.16}$$

$[I_c]$is the inertia matrix based upon the center of mass:

$$[I_c] = \begin{bmatrix} I_{x_{orb}x_{orb}} & -I_{x_{orb}y_{orb}} & -I_{x_{orb}z_{orb}} \\ -I_{y_{orb}x_{orb}} & I_{y_{orb}y_{orb}} & -I_{y_{orb}z_{orb}} \\ -I_{z_{orb}x_{orb}} & -I_{z_{orb}y_{orb}} & I_{z_{orb}z_{orb}} \end{bmatrix} \tag{2.17}$$

$I_{x_{orb}x_{orb}}$, $I_{y_{orb}y_{orb}}$ and $I_{z_{orb}z_{orb}}$ are the moments of inertia. The other components are the products of inertia and represent a lack of mass symmetry. Considering the axes of the satellite to be orthogonal, the products of inertia are zero which leads to:

$$\vec{L}_c = \begin{bmatrix} I_{x_{orb}x_{orb}} \cdot \omega_{x_{orb}} \\ I_{y_{orb}y_{orb}} \cdot \omega_{y_{orb}} \\ I_{z_{orb}z_{orb}} \cdot \omega_{z_{orb}} \end{bmatrix} \tag{2.18}$$

The rotation of a satellite as a sequence of separate rotations about the coordinate axis can be described using Euler angles: yaw (ψ), pitch (θ) and roll(ϕ) . For separate rotations, we can consider $[X(\phi)]$ the rotation matrix for a rotation through an angle ϕ about the x$_{orb}$ axis, $[Y(\theta)]$ the rotation matrix for a rotation through an angle θ about the y$_{orb}$ axis and $[Z(\psi)]$ the rotation matrix for a rotation through an angle ψ about the z$_{orb}$ axis. These matrices are

$$[X(\phi)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{2.19a}$$

$$[Y(\theta)] = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{2.19b}$$

$$[Z(\psi)] = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19c}$$

Assuming the rotation matrix for transforming a vector's components from a axis to B axis is $[X(\phi)][Y(\theta)][Z(\psi)]$, then the inertia matrix may be transformed between the same sets of axis using

$$[I_B] = [X(\phi)][Y(\theta)][Z(\psi)][I_A]([X(\phi)][Y(\theta)][Z(\psi)])^{-1} \tag{2.20}$$

Chapter 3

Star trackers overview

Star trackers have the ability to extract attitude with high accuracy in fractions of a second by analyzing a frame with the stars present on its field of view (FOV). The attitude is extracted using an algorithm that compares the stars in the FOV with an internal catalog. If used together with the information from gyroscopes, a higher accuracy can be achieved, since star trackers can contribute to the correction of the incremental error in attitude determination provided by gyroscopes. In order to identify the important characteristics to have in account designing a star tracker, a study of its architecture and functioning is made on this chapter.

3.1 Star tracker's architecture

Any star tracker is composed by an optical system, an electronics and signal processing system and a image sensor. The optical system focus the light from the stars into the image sensor. The image sensor extracts the frame and ensures the connection between the optical system and the electronics and signal processing system.

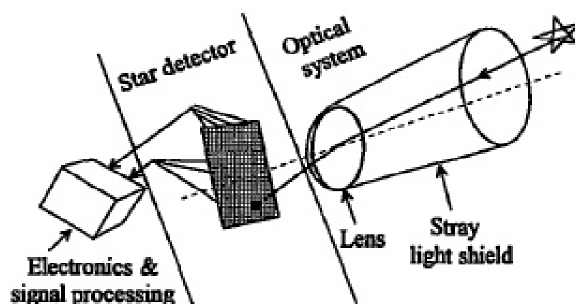


Figure 3.1: Star tracker design and its subsystems. The star sensor can be a Complementary metal-oxide sensor, CMOS or a Charged coupled device, CCD. Image adapted from [17].

The electronics system holds the memory and the processor, which are responsible for imaging processing and extraction of attitude.

3.1.1 Star properties and their relation with a star tracker's optical system

The optical system is composed by a set of lenses and a baffle. The baffle acts as a stray light shield and ensures the star tracker is protected from light that affects its functioning, such as the Sun light. When the Sun is within the star tracker's FOV, the pixel array in the image sensor becomes saturated and

the star tracker is not operable, since it cannot acquire frames with stars. There are other indirect sources of light that can also compromise the operation of the star tracker like the Earth albedo and moonlight [17]. In short, any source of light may saturate the image sensor and compromise the process of attitude determination through star identification.

Star Photometry

The incoming light from stars is measured by the flux density, which gives the power of radiation emitted by a star per unit area in determined bandwidth. The flux depends on the star's brightness, which is described in terms of magnitude. The magnitude is defined as the logarithmic measure of an object's brightness, calculated in a specific bandwidth. There are two distinct types of magnitude: the apparent magnitude and the absolute magnitude. The apparent magnitude is the apparent brightness of a star when observed in the Earth's night sky. The absolute magnitude is the intrinsic brightness of a star when observed 10 parsecs away from the Earth. Star trackers for Earth orbit missions only use apparent magnitude. The magnitude scale works in reverse and varies from negative values to positive values, with objects with a negative magnitude being brighter than the ones with a positive value. This means the most bright star has the smallest magnitude. The apparent magnitude, *mag*, of a star is measured using its measured flux, Φ:

$$mag - mag_0 = -2,5 log \frac{\Phi}{\Phi_0} \tag{3.1}$$

where Φ_0 is the flux of a reference star with magnitude mag_0 [20]. 3.1 can be rearranged in order to give the flux, if a magnitude of a star is known.

$$\Phi = \Phi_0 \cdot 10^{-\frac{mag - mag_0}{2,5}} \tag{3.2}$$

It is extremely difficult to calibrate astronomical photometric equipment to measure magnitude, because of the atmosphere's variability and because light is not monochromatic. This is why astronomical magnitude measurement systems are defined in terms of the brightness of sources in specific wavebands relative to a set of standard stars, selected by agreement[21]. For instance, the Johnson-Cousins system is a system designed to measure apparent magnitude of stars in ultraviolet (U), visible (V), blue (B), red (R) and infrared (I) wavebands.

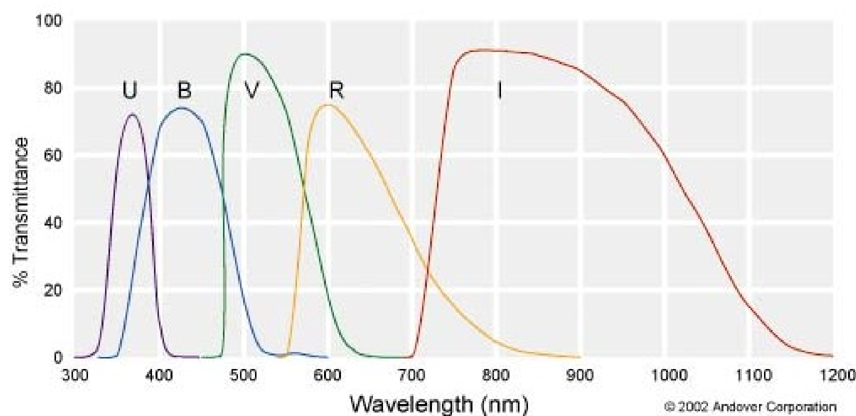


Figure 3.2: Johnson-Cousins UBVRI filter curves. Image adapted from the Andover Corporation website

The standard stars usually used for magnitude measurements are A-type main-sequence stars with magnitude zero in all bandwidths, A0 V. This type of star is defined as the most brilliant star in the night sky to the human eye, which in Earth's it corresponds to Vega star. Vega has a magnitude of 0,03 in the Johnson-Cousins system.

Symbol	Flux W m^{-2} $\mathring{A}^{-1}$	Full width half maximum (FWHM) $\mathring{A}$	$\lambda_b w$ μm
U	$4,22 \times 10^{-12}$	600	$0,36$
B	$6,40 \times 10^{-12}$	940	$0,44$
V	$3,75 \times 10^{-12}$	880	$0,55$
R	$1,75 \times 10^{-12}$	1380	$0,71$
I	$8,40 \times 10^{-10}$	1490	$0,97$

Table 3.1: Flux calibration values for a star of the spectral type A0 V with zero magnitude on the Johnson system. The FWHM is also defined as the bandwidth. $\lambda_b w$ is the central wavelenght of the filter, also known as effective wavelength. Adapted from [20]

Theoretically, the total incident flux of an A0 V star is calculated using $\Phi_b w$, which is the spectral flux in a determined bandwidth $\Delta\lambda_b w$

$$\Phi_0 = \int_{\Delta\lambda_b w} \Phi_b w d\lambda \tag{3.3}$$

To determine the flux emitted by a star in several bandwidths, the integral should cover all bandwidths and have in account the overlap between them. This can be quite difficult to do, and sometimes approximations need to be made. The amount of flux from a given star detected by the image sensor depends also on the wavelength region in which the sensor operates. This region is often represented in a spectral sensitivity curve for each sensor.

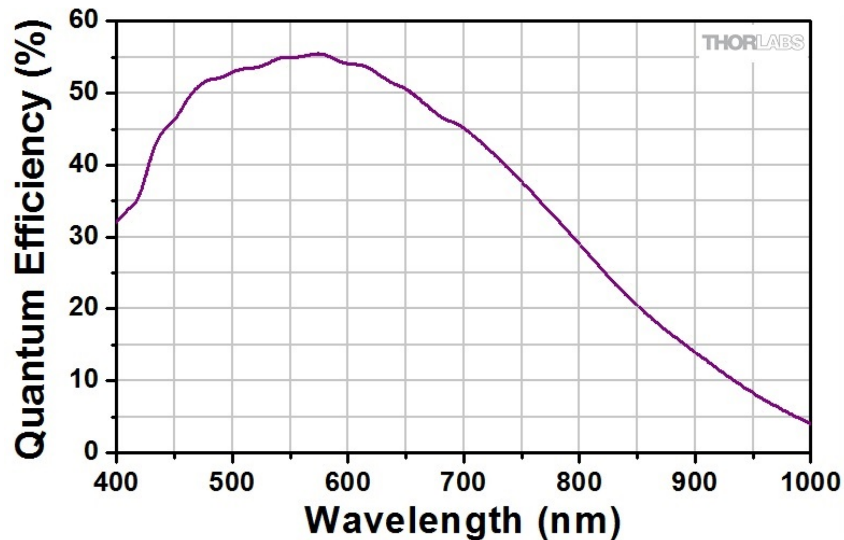


Figure 3.3: Spectral sensitivity curve for a Silicon CMOS (DCC1545M by Thorlabs). Image courtesy of the Thorlabs website

The spectral sensitivity curve gives the quantum efficiency (QE), of the sensor for each wavelength in which it operates. The quantum efficiency represents the ratio between incident photons and converted

electrons. The right way to measure the flux is to transpose filter curves from Johnson-Cousins system with the image sensor's spectral sensitivity curve.

Wave to particles

Equation 3.2 considers the wave-like nature of light, which is important for optics. However the particle-wave like nature of light also needs to be considered in order to study the quantum efficiency. The energy of a photon emitted by a star with a given wavelength is given in Joule (J):

$$E_\gamma = \hbar \cdot \omega = \frac{hc}{\lambda_b} \tag{3.4}$$

This relation can be used in order to transform the flux transmitted by a star in determined bandwidth to photons. The number of photons emitted by a star n_e, decrease after crossing the lens system, due to the transmission coefficient of its composing lens. In the case of one lens, the photons that pass through the lens are easily determined by multiplying the incoming photons by the lens transmission coefficient. Since there should be more than one lens in a star tracker's optical system and assuming each lens *i*, has a different transmission coefficient, the number of transferred photons, n_t, that arrive to the image sensor are given by

$$n_t = n_e \prod_{i=1}^{N} \tau_i \tag{3.5}$$

The number of photons that crosses the lens system is also dependent on the f-number of the lens system. The f-number gives the ratio between the focal length and the diameter of the system's aperture. Higher the aperture, higher the number of photons that arrive to the image sensor in determined time.

$$f - number = \frac{f}{d_lens} \tag{3.6}$$

Higher the f-number, smaller the aperture, which means less photons pass through lens in order to achieve the image plane where the sensor is.

3.1.2 Image detection sensor and image formation

The image sensor is responsible for the conversion of photons coming from the stars into photo-electrons. The image sensor can be a charged-coupled device (CCD) or a complementary metal-oxide semiconductor (CMOS). Each type of sensor has its own advantages and disadvantages, e.g. CCD's tend to have lower noise, but CMOS are more resistant to radiation damage and consume less power. Both types of sensor are composed by a grid of pixels. When the light coming from a star achieves the array, it is spread by a few group of pixels. The resulting image is a product of the convolution between the object (star), and the image system's point spread function (PSF).

Star signal and noise

A PSF is defined as the impulse response of a focused optical system to a point source. The shape of the PSF depends on which factors are limiting image quality. If an optical system image quality is no longer limited by imperfections in the lenses, but only by diffraction, then it is said to be diffraction limited. Due to diffraction, the smallest point to which a lens or mirror can focus a beam of light is the

size of the Airy disk. The image of the stars observed with a perfect circular objective lens is expected to be an Airy diffraction disk.

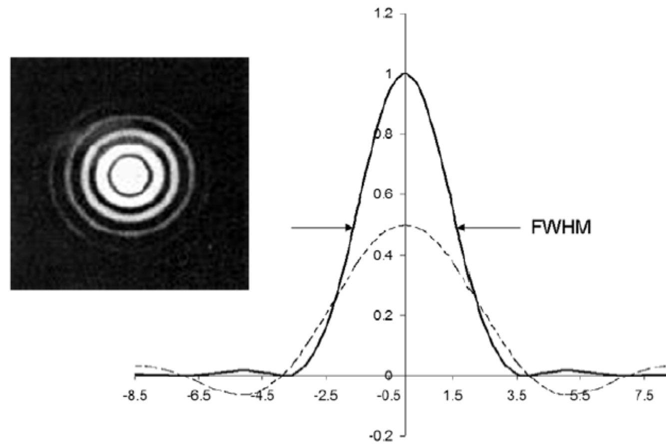


Figure 3.4: Point spread function of a star in an image sensor using a circular objective lens. The expected format for the PSF is the Airy diffraction disk, with maximum and minimum points.

The PSF's central peak for an Airy diffraction disk has a Gaussian-like form, as seen in figure 3.4. The Airy disc diameter can be determined through the specifications of the lens system and the wavelength of a specified filter:

$$D_{AiryDisk} = 1,22\lambda\frac{f}{D_{lens}} \tag{3.7}$$

Image formation

During image acquisition, the light from a specified star is spread over multiple pixels in the image, in a circular shape. This circular light distribution is described by a convolution between the object and the PSF plus the noise and depends on the stars magnitude. Brighter stars have more intense distributions than dimmest stars.

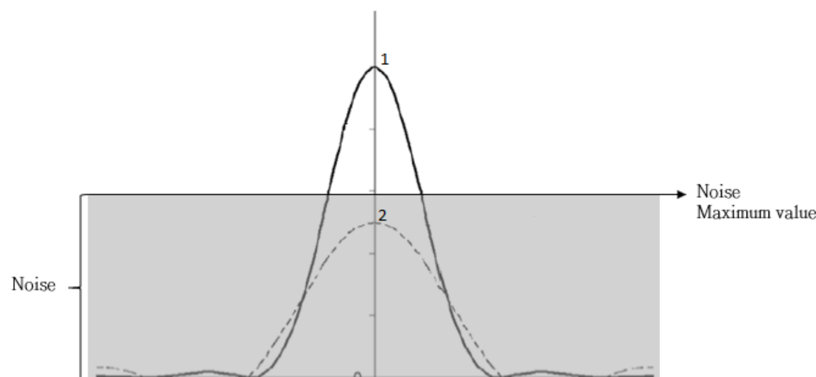


Figure 3.5: Two star light distributions: one from a brighter star (1) and another from a dimmer star (2). The one from the brighter star is more intense and its signal overlaps the existent noise, which means this star is going to be detected. Light distributions with low intensity are hidden by noise, since the signal is not strong enough to overlap the noise. Stars with a light distribution peak lower that noise are not detected.

The noise has a negative impact on the process of image formation and affects the image sensor's efficiency, i. e., its capacity in converting photons to electrons and electrons to measured counts. Despite that, the output signal should be linearly proportional to the photon arrival rate. There is a limit for the magnitude of the stars detected, because light distribution of dimmest stars are often hidden by the noise of the sensor. The most useful way to know the magnitude limit expected for determined sensor, is by finding out the ratio of signal that can be detected without being hidden by the noise.

3.1.3 Noise analysis

The noise describes the electronic fluctuations in the equipment. The collection of light within a period of time and within a single pixel follows Poisson distribution [22], due to the fact that light arrives in a discrete form (photons). Poisson distribution is very useful to model the number of times an event occurs in an interval of time or space.

The total noise, υ, of an image detector depends mostly of three critical quantities: the dark noise υ_{DN}, the photon shot noise υ_{SN}, and the readout noise υ_{RN}:

$$\upsilon = \sqrt{\upsilon_{DN}^2 + \upsilon_{RN}^2 + \upsilon_{SN}^2} \tag{3.8}$$

The dark noise and the shot noise are highly proportional to the sensor's integration time. This time is defined as a period in which the detector is already processing a photon and is not sensitive to another incoming photon. A high integration time implies a higher number of photons absorbed in each pixel, but also implies a higher noise. In the particular case of a star tracker in orbital movement, a high integration time may result in stars moving in the focal plane, which causes a ”tail effect” in the image. This ”tail effect” creates a systematic error in determining centroids [23]. Stars may not even be recognized as so by the algorithm.

Charge conversion efficiency

CMOS image sensors usually include analog to digital converters (ADCs) on their boards, which are responsible to convert the analog signal coming from the pixel into digital numbers (ADUs). When the noise is quantified through image acquisition, which is the case in this work, the charge conversion efficiency (CCE) should always be determined. The dark noise and readout noise are extracted through the counts in an image and they need to be converted into units with physical meaning, i.e. electrons. The CCE defines the relationship between the number of electrons converted and the resulting output in counts. The CCE also allows to determine quantum efficiency which gives the spectral sensitivity curve for the image detector.

Dark Noise

Dark noise is the statistical variation in the number of electrons thermally generated within a pixel. The dark noise increases linearly with integration time and the dark current of the detector, ι_{DC}. The dark current is the signal received while the detector is working in the dark. This value allows to predict the behavior of noise during thermal cycling in orbit, since dark current increases along with temperature.

$$\upsilon_{DN} = \sqrt{\iota_{DC} \cdot t} \tag{3.9}$$

The dark current increases with the rising of the system's temperature. To maintain low dark noise

values a thermal cooling system should be programmed to cool the star tracker when the temperature in the electronic system rises above the ideal.

Readout Noise

The readout noise is the amount of noise generated by all the electronics. This value depends on the charge conversion efficiency of the detector and it is directly measured by extracting a frame in the dark with an integration time close to zero:

$$\upsilon_{RN} = \frac{CCE \cdot \sigma}{\sqrt{2}} \tag{3.10}$$

where σ is the standard deviation from the pixel signal of the frame taken. A lower readout noise is always desirable, because while dark noise and shot noise can be controlled, readout noise doesn't since it only depends on the quality of the EEE components.

Shot noise

Shot noise is a consequence of the discrete nature of light. It describes the fluctuations in the number of photons detected and converted to photo-electrons in the image sensor.

$$\upsilon_{SN} = \sqrt{\gamma} \tag{3.11}$$

γ is the number of photons detected in the given exposure time. Since the number of detected photons increase with exposure time, shot noise depends also on the exposure time. Higher the input light levels, the total noise starts to be dominated by shot noise.

Signal to noise ratio

The signal to noise ratio (SNR), is a quantity used to compare the amount of signal received with the level of the background noise and it is given in decibels (dB) by

$$SNR = 10 \log_{10} \frac{signal}{\upsilon} \tag{3.12}$$

If this value is positive, this means that the signal overlaps the noise and can be read by the sensor.

3.2 Space environment impacts on star tracker performance

Space is a very hostile environment. The orbits have debris flying around and temperatures can vary greatly from extremely cold to extremely hot. All these adverse conditions affect star tracker performance: the varying temperatures can deform the lens system, displacing stars' position in images. The transmission coefficient in lens can also be affected by radiation, decreasing star light signal on the sensor. Radiation effects on the lens system can be overridden by choosing the right lens materials as can be seen on figure 3.6.

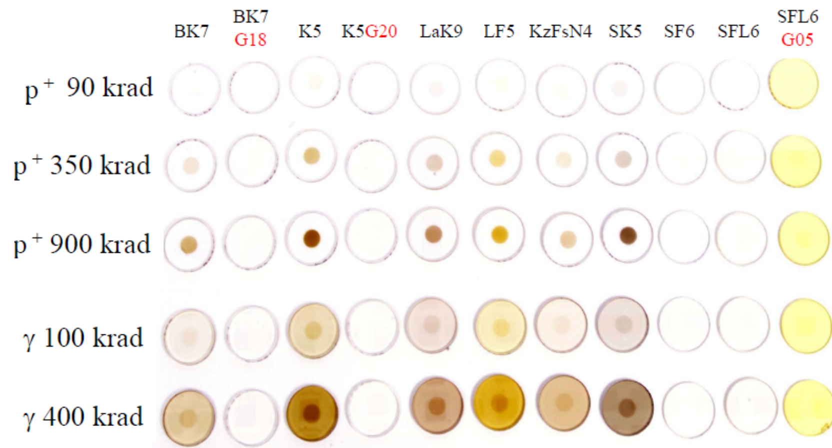


Figure 3.6: Materials commonly used in lens glass and how several doses of radiation affect them. The radiation affects the transmission coefficient of the lens [24]

Radiation doses can also have negative effects on the electronics of the star tracker. Radiation and sun light reflected in debris manifest themselves in frames as brilliant spots the star tracker confuses as stars. Nowadays evolved algorithms have the capability to ignore these spots. Although, the method is not infallible and some spots can still be considered to be stars by the star tracker. Some types of radiation can also cause the degradation of the image sensor and electronics, reducing significantly the star tracker's lifetime.

3.2.1 Space radiation

Space radiation is generated by particles emitted from a variety of sources both within and beyond our solar system. Radiation can be ionizing or non-ionizing, being the first one the most common in space radiation. Ionizing radiation is radiation with energy high enough to remove electrons from their orbits in atoms, resulting in charged particles. This radiation includes gamma rays, neutrons and protons. Unlike non-ionizing radiation, which also exists, ionizing radiation is quite difficult to be shielded. There are three types of space ionizing radiation: trapped radiation, galactic cosmic rays and solar particle events (SPE)[25].

Galactic cosmic rays have origin outside the solar system and consist in ionized atoms and protons. The flux of these particles is not so high, however their velocity and sometimes high mass produce intense ionization as they pass through matter [26].

The higher fluxes of radiation in LEO orbits come from solar activity and trapped particles. The fusion process on the Sun's core produces electrons, protons and alpha particles in great abundance. Those products of fusion travel from the Sun into all directions of space. When the ejected particles travel towards Earth, they are called solar winds. Some particles are deflected by the magnetosphere and others become trapped in the Earth's magnetic field, creating the Van Allen radiation belts [27][25].

When an event such as solar flare or coronal mass ejections happen, the flux of ejected particles from the Sun is higher and there is an SPE. During an SPE the ejected particles collide with the magnetic field surrounding the Earth. Such collisions can eject heavy ions and electrons into the Earth's magnetosphere, affecting the terrestrial magnetic field, allowing particles to reach previously unattainable altitudes and inclinations[25].

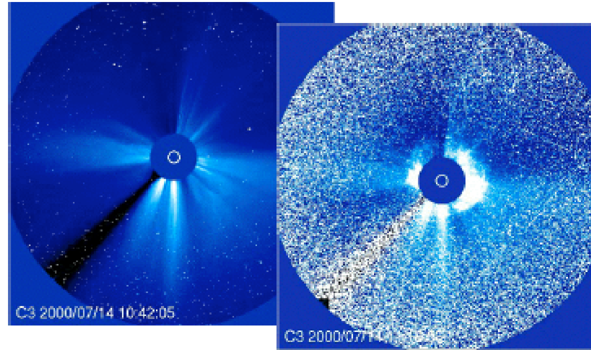


Figure 3.7: Radiation effects on a star tracker's image. The spikes caused by usual radiation doses on the left image can be considered to be stars if the algorithm is not ready to filter them. During and SPE the radiation dose may be so high that the star tracker's image sensor gets saturated. During this high radiation doses the star tracker is not capable of star detection. Image courtesy of ESA.

3.2.2 Radiation effects on electronic components in LEO

The worst source of ionizing radiation dose in LEO are the particles trapped in the Van Allen belts. Nonetheless, environment in LEO is fairly benign. The total ionizing dose (TID), is defined as the material damage caused by ionizing radiation and it is quantified by deposited energy per mass for a given material in units of Gray or Rad [28]. In LEO, the typical TID rate varies between 4 Krad/Year and 40 Krad/Year [1] for effective shielding thicknesses of approximately 0,3 mm of aluminum [29].

Heavy ions resulting from occasional occurrence of solar flares or coronal mass ejections lead to Single event effects (SEE), which are defined as a disruption in function of electronic circuits [28]. During times of high solar activity the ejection of particles may last periods of hours or days. There is a convenient way to express the rate at which energy is deposited in matter as heavy ions and/or charged particles travel through, called linear energy transfer (LET), expressed in MeV cm^2/mg [28].

A space graded component is one that is designed to provide specific radiation performance. Usually these types of instruments tend to withstand radiation doses superior to 100 krad. Space graded components are extremely expensive and appropriate for very high dose environments like an interplanetary mission, or a high altitude orbit. An alternative to this expensive components are coined careful COTS, which involves providing radiation tolerance to improve reliability of COTS components[28].

Some COTS components may have already some small tolerance to radiation.[28]. There is no way to know how a commercial component is going to react in space environment unless is tested. Since the image sensor is the most critical component in a star tracker, a full study on its behavior related to noise and radiation tolerance is needed.

3.3 Market survey and desired requirements for the star tracker to be developed in this work

Despite the difficulty to minimize star trackers without loosing accuracy, there are nowadays some star trackers developed to integrate small satellites, particularly cubeSats. Unfortunately, the price tag is still too high to justify their presence in a ordinary cubeSat low-budget mission designed by a small company or university. An example of that is the star tracker developed by Berlin Space Technologies, ST200, presented in figure 3.8.

[1]Rad is a unit of absorbed radiation dose, defined as 1 rad = 0,01 Gy = 0,01 J/kg

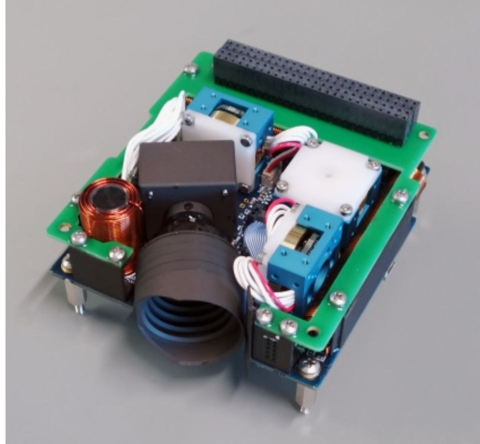


Figure 3.8: Small star tracker developed by Berlin Space Technologies integrated in a ADCS for a cubeSat. Image adapted from the Berlin Space Technologies website

A star tracker like the ST200 can easily reach 80.000 euros. The goal of this work is to propose a star tracker whose price doesn't even reach a quarter of that price. This work intends to develop a star tracker with characteristics similar to the Berlin Space Technologies one, as seen in table 3.2.

ST200 star tracker from Berlin Space Technologies	
Accuracy	10 arsec
Mass	40g (without baffler)
Volume	$30 \times 30 \times 38$ mm^3
Update rate	200 ms
Temperature resistance	-20°C to 40°C
Maximum magnitude of detected stars	6
Radiation Test	11Krad (proton source)
Design Life	5 years

Table 3.2: Characteristics of the star tracker from Berlin Space technologies. Courtesy of Berlin Space Technologies website.

Considering all star tracker characteristics resumed in this chapter's previous sections, the requirements for the star tracker to be developed in the future are the following:

1. The FOV shall be 10º.
2. The lens system should have a f-number lower than 2,5.
3. The full star tracker (image sensor, processing board and lens system), should weight less than 250g.
4. The full star tracker volume should not be higher than 50 mm^3.
5. The algorithm runtime shall not be higher than 1s.
6. The star tracker shall resist the temperature from -20°C to 40°C.
7. The magnitude limit shall not be less than 5.
8. The accuracy should be better than 20 arcsec.
9. The star tracker shall resist a radiation test of 10Krad from a proton source.

Some of the requirements listed are not as good as the ST-200 specifications. These requirements are used as a baseline and inspiration in this work to construct a development prototype using COTS

technology available at the university. The ST-200 is always being improved, and its first commercial versions were not as good as the actual one [30] . The requirements presented can be improved along with time and experience.

Chapter 4

Imaging sensor and optical system

There is no sensor with 100% efficiency. Several intrinsic properties , e.g. the noise, affect the efficiency and performance of an image sensor. This chapter reviews the work made to characterize the noise of a commercial CMOS image sensor header board from Aptina (actual On Semiconductor). Using the calibration of an A0 V star and a cheap integrated lens system, a study on star photometry was also made in order to theoretically preview the magnitude of the stars detected by the test sensor and other commercial image sensors based on data from their datasheets.

4.1 Image sensor specifications and USB ArduCAM camera shield

The selected image sensor for this work was MT9J001, a CMOS 10MP monochromatic image sensor[1]. The datasheet from this sensor describes it as a breakthrough low-noise CMOS imaging sensor that achieves near-CCD image quality. The most relevant characteristics from MT9J001 are described in table 4.1.

Some MT9J001 specifications from the datasheet	
Active imager size	6,440mm × 4,616mm
Active pixels	3856 × 2764
Pixel size	1,67 μm × 1,67 μm
Dynamic range	65,2 dB
Operating temperature	-30°C to 70°C

Table 4.1: Main specifications of the MT9J001 sensor, taken from the datasheet.

MT9J001 camera header board has the possibility to be integrated with the ArduCAM USB camera shield, a universal camera control board that provides the user fast evaluations. When MT9J001 is mounted on the USB camera shield board is only able to achieve a frame rate of 6fps. This frame rate is obviously not ideal for space, however the ArduCAM USB camera shield is used only with the goal to test and characterize the image sensor using low-cost means. The ArduCAM board can be embedded in Raspberry Pi and TI Beaglebone and allows a connection via USB cable to computers with Windows or Linux operative systems. The sensor board connects the ArduCAM USB camera shield control board

[1]Aptina had two 10MP sensors: MT9J003 (color) and MT9J001 (mono). Due to its similarities, On Semiconductor gave the name MT9J003 to both versions: MT9J003-color and MT9J003-mono

with a 30 pin ribbon cable, which only allows 8 bit data bus. The communication between the sensor board and ArduCAM USB camera shield board is described in figure 4.1.

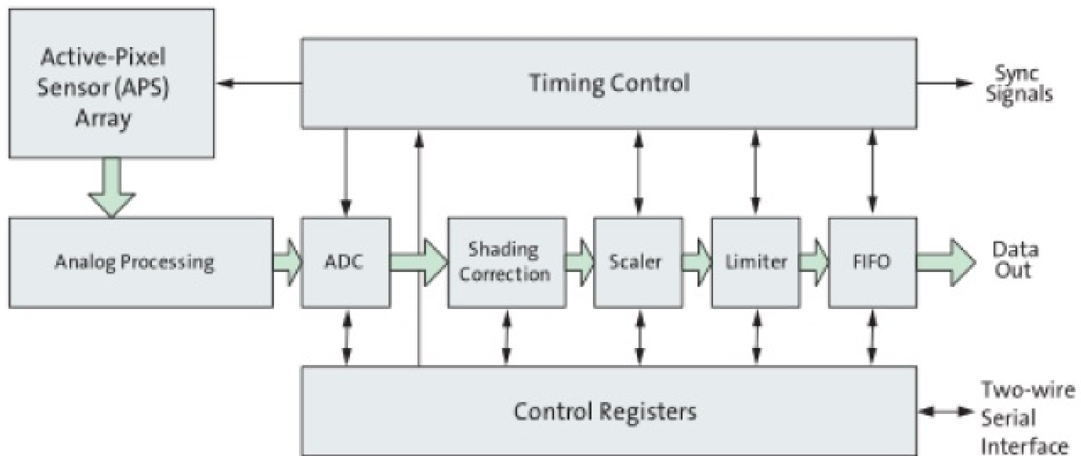


Figure 4.1: Block diagram of the electronic connection between the MT9J001 board and the ArduCAM USB camera shield board. This block diagram was taken from ArduCAM USB camera shield board datasheet.

The parallel connection between both boards is not possible for this specific image sensor.

In this configuration, the sensor is controlled with the user interface available at ArduCAM USB camera shield github. The sensor is programmed using registers, that accept hexadecimal values that can be changed in order to achieve the desired specifications for integration time and others. The code for the ArduCAM USB camera shield board to be used in windows is already implemented in an interface user software called USBTest that can be downloaded in the official ArduCAM github.

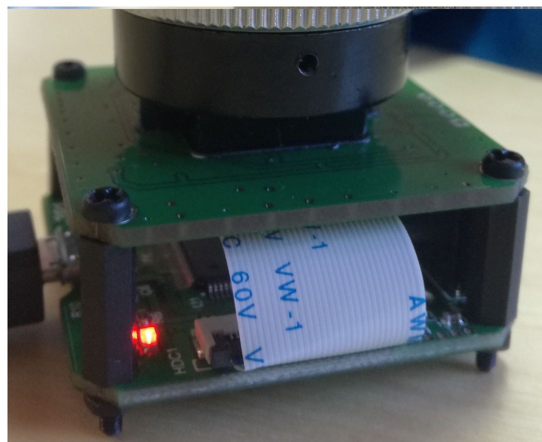


Figure 4.2: Image sensor MT9J001 connected on ArduCAM USB camera shield board through a 30 pin ribbon cable. The ArduCAM USB shield board has a USB input that allows it to be connected to a portable computer, Beaglebone or Raspberry Pi. When connected, the camera has a red led that turns on.

This software still allows the user to control camera registers, but in a more user friendly way.

4.1.1 Register values for the MT9J001 image sensor

There are some critical register values for the MT9J001 image sensor in this work's practical implementation. The integration time registers allow the control of the star tracker's exposure time while the

gain registers control the balance between the photons read and their conversion. All MT9J001 register values are in Annex A to be consulted.

Integration time

The integration time depends on some different register values. According to the datasheet, integration time is measured, in miliseconds, using this equation,

$$integration\ time = \frac{(coarse_integration_time \times line_length_pck) + fine_integration_time}{vt_pix_clk_freq_mhz \times 10^6} \tag{4.1}$$

The values for line_length_pck and vt_pix_clk_freq_mhz are fixed for this work's application. The last one is related with the clock frequency of the image sensor header board, and has a value of $1/6.25$ MHz. The first one has a decimal value of 7440. The minimum value allowed for the coarse_integration_time is 0 and the minimum value allowed for the fine_integration_time is 1010. This means the minimum integration time for this camera is 6,31μs.

Gain

The brightness is a parameter also important for a star tracker, specially when it is being tested on Earth. Stars with higher magnitudes achieve lower pixel values. These stars might show up on a frame and be visible to human eye, however, it doesn't mean it is recognizable by the algorithm. The maximum pixel count values achieved by the star may not be high enough for the algorithm to consider it a star.

Higher the sensor's gain, higher the star tracker's pixel count for a given star. Higher the pixel count, higher the central pixel value. This is why it is important to consider the analog gain register of the sensor, Analogue_gain_capability. By default this value is 1 but it may be changed in order to increase pixel values. However, there is a downside in increasing the gain: the noise also increases.

4.2 Experimental characterization of MT9J001 noise

Some important noise values are missing from the MT9J001's datasheet, like the readout noise and dark current. The readout noise is quite important because the algorithm can be coded to extract it from an image.

All methods used to determine the CCE, the dark current and the readout noise were based on the articles [31] and [22].

4.2.1 Determination of the charge conversion efficiency

In chapter 3, the CCE was introduced as the relationship between the number of electrons converted and the digital counts. The digital counts result from converting the analog signal to digital signal in the ADC. This quantity can be measured by focusing a monochromatic and homogeneous source of light into the image sensor. The most convenient way to determine the CCE is to plot the variance vs mean counts of images extracted with the source of light on for different integration times, until saturation is achieved. Saturation occurs when pixel values achieve its maximum values. For the monochromatic sensor used along with 8 bit data bus, this value is 255.

Image sensor's counts follow Poisson's distribution. One of Poisson's distribution characteristic is that the mean is equal to the variance. This is why increase of light level is expected to increase not also the mean but also the variance. In order to extract the mean counts and variance for each integration time, 10 frames were extracted for each integration time. For each group of 10 frames, a mean was made in order to have just one frame per integration time. This process allows to smooth effects caused by abrupt changes in the environment or other random effects.

Materials used and experimental setup

As a light source it was used a laser beam. A laser beam guarantees the incoming flux has only neglectful variations. The one used was an Neon-Helium gas laser available in the optics lab and its specifications are represented in table 4.2.

HeNe gas laser specifications	
Wavelength	632,800 nm
Beam diameter	0,480 $\pm$ 0,014 mm
Beam divergence	1,700 $\pm$ 0,051 mrad

Table 4.2: Main specifications of the laser beam used to characterize the noise of the image sensor. All these values were taken from the datasheet of the HeNe laser.

An integrated sphere assures the light beam from the laser reaches pixel arrays in the sensor homogeneously. This sphere also acts as a baffler, preventing stray light beams coming outside the sphere from interfering with measurements. The integrating sphere has three cavities where the detector and other instruments can be mounted. Any cavity can be closed, if there is no instrument to be mounted there. The incoming flux from the laser beam can be measured with a photo-detector in order to guarantee it doesn't vary significantly with time.

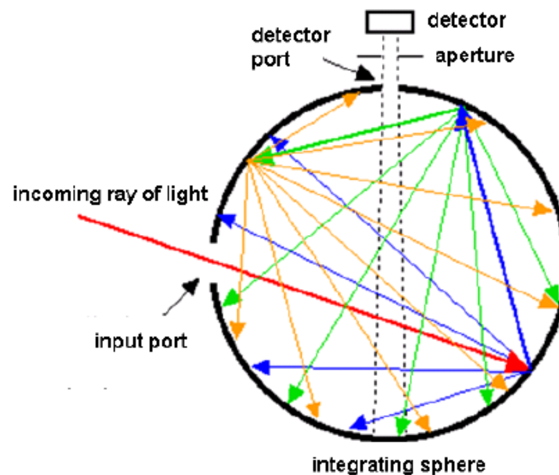


Figure 4.3: Experimental setup of the integrating sphere and laser beam. The integrating sphere acts as a baffler keeping stray light from reaching the photo-detector. The cavities that don't have any instrument mounted are closed.

The photo-detector is placed in one of the integrating sphere's cavity and the laser beam in other, as seen in figure 4.3. The remaining cavities are closed in order to guarantee the only light that achieves the

sensor is the one coming from the laser beam. The photo-detector used to measure the flux coming from the HeNe laser beam was model 918D-UV-OD3R from Newport.

918D-UV-OD3R photo-detector specifications	
Spectral range	200 up to 1100 nm
Linearity	±0,5%
Uniformity	±2%
Calibration Uncertainty	1%

Table 4.3: Main specifications of the photo-detector used to measure the incoming flux from the HeNe laser beam, taken from the datasheet.

The resulted measured flux is a mean from several values taken in a short period of time. To finally measure the CCE, the previous configuration used in figure 4.3 was maintained, but the photo-detector was replaced with the image sensor MT9J001, as seen in figure 4.4.

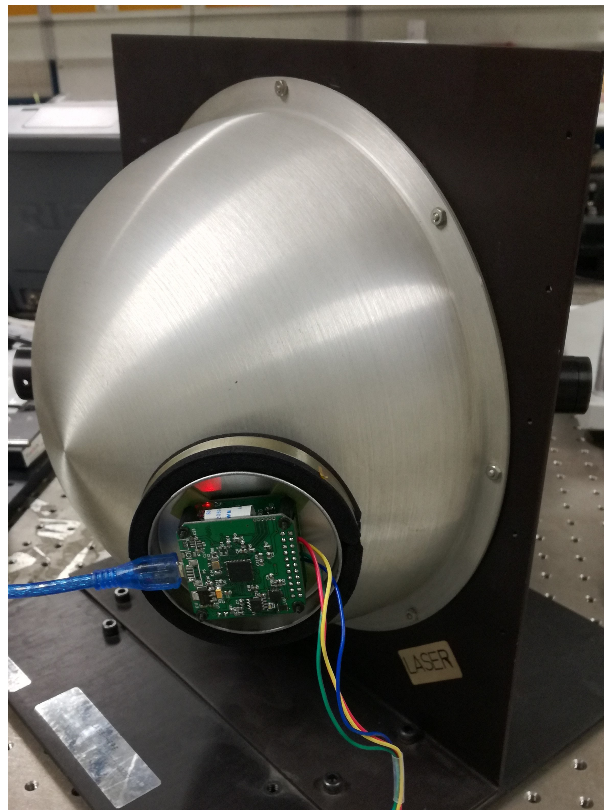


Figure 4.4: MT9J001 image sensor mounted in an integrating sphere's cavity

Measurement and results

The integration time was varied between 6,31 μs, which is the minimum value allowed to be used for MT9J001 and 71,43 ms. This last value corresponds to the value when the image sensor became completely saturated. For each frame, the mean counts and variance counts were extracted. The results are shown in figure 4.5.

Despite the theoretical model, the graph of mean counts vs variance is not linear in all the domain of integration time. For higher input levels of light, the total noise of the sensor is dominated by shot

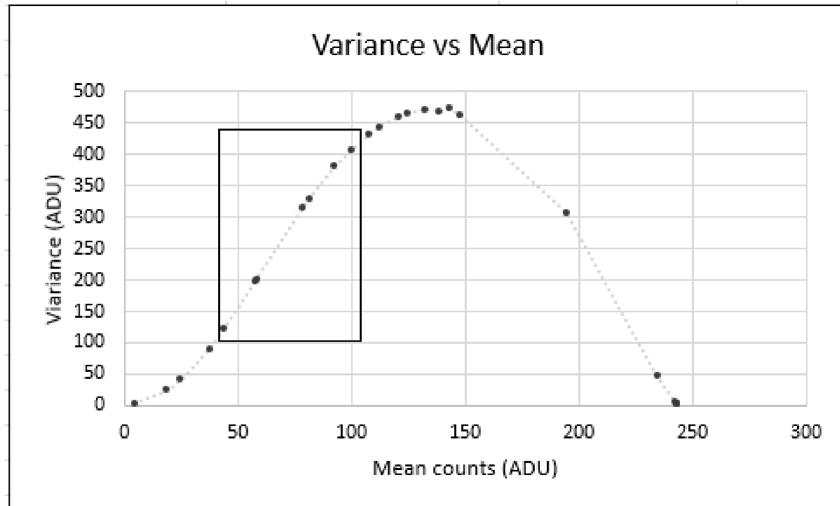


Figure 4.5: Plot showing mean counts vs variance counts for several images extracted by the MT9J001 with the laser beam on. The rectangular form is the linear region chosen to plot and extract CCE.

noise, and the the mean counts and variance don't appear to have a linear relation. The same happens for low levels of light: the total noise is dominated by the readout noise. The trick is to truncate the graph to a region where the mean counts and variance are linear. In figure 4.5, this region is located inside the rectangular form and corresponds to this plot shown in figure 4.6.

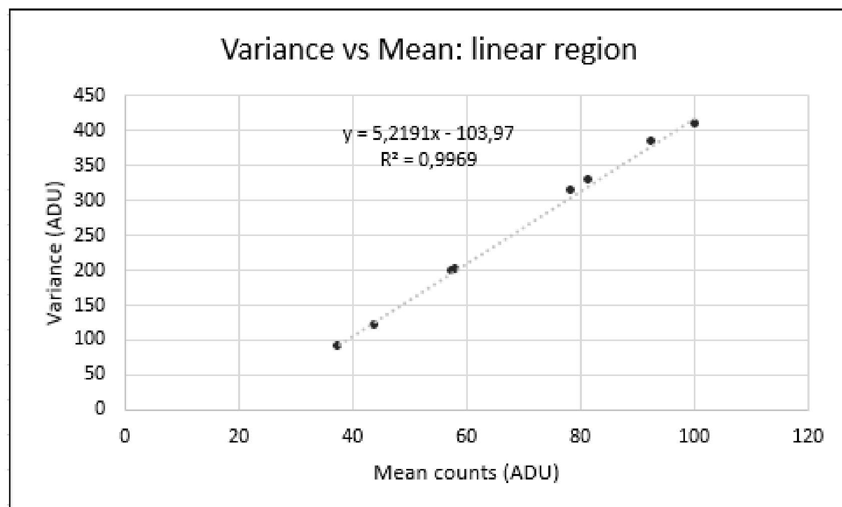


Figure 4.6: Plot showing mean counts vs variance counts linear relation for several images extracted by the MT9J001 with the laser beam on. This plot corresponds to the linear regime chosen in picture 4.4 (inside the rectangular form). The slope of the linear fit is the CCE of the image sensor.

The slope resulting of the linear fit for the plot in figure 4.6 is the CCE of the image sensor.

4.2.2 Determination of the readout noise

The readout noise is the equivalent noise level in the dark and at zero integration time. This quantity can be measured with a bias frame, which is an image extracted with zero integration time and the shutter closed. Since zero integration time is a value impossible to achieve (as seen a few paragraphs above), the value most close to zero should be used: 6,31 μs.

Materials and experimental setup

The experimental setup and materials used to measure readout noise are the same used to determine the CCE, with the exception of the laser beam. The laser beam is not needed, since it is mandatory for the image sensor to be in the dark.

Due to the fluidity of light and in order to guarantee there is no light coming to the sensor, the sensor was mounted shutter closed in the integrating sphere's cavity and the light in the optics laboratory was off. This time, all cavities of the integrating sphere were closed except for the one where the MT9J001 sensor was mounted. The sensor header board was connected to a personal computer via USB cable in order to extract images and control register values.

Measurements and results

Two bias frames were extracted in the *fits* format, the format recommended to process astronomy scientific data. The *fits* format guarantees the image is stored in a table that matches the pixel grid. Each cell has the count of digital numbers processed in the matching pixel. The program used to process images was $QfitsView$. The second bias frame was subtracted from the first one, in order to leave only the structure of the noise.

The standard deviation of the resulting frame was σ = 3,24 ADU. This value can be used with the CCE value in equation 3.12 to give the resulting value in electrons: 11,97 e^{-}.

4.2.3 Determination of the dark current

Ideally, this quantity would be measured with a thermal control system, since it varies according to temperature and it can intrude in the readout noise measurements. Since this work only intends to prove the conception process behind a full functional low-cost star tracker, it was decided the dark noise measurements were only made at environmental temperature: 24°C.

Dark current is measured by taking dark frames with different integration times. A dark frame is an image extracted with the camera in the dark, shutter closed, in a similar way it was made for readout noise. The difference between bias frames and dark frames is that dark frames are taken with integration times different from zero and bias frames are taken with a theoretical zero integration time.

Materials and experimental setup

In this work, the extraction of dark current was also made with the integration sphere, using the configuration used for the readout noise extraction. 10 dark frames were taken for different integration times, until camera achieved a reasonable level of saturated pixels (hot pixels).

Measurements and results

A mean of the 10 dark frames taken from each integration time was made in order to have only 1 picture for each integration time. The mean count of pixels was extracted from each frame. With the data extracted, it was created a plot of mean counts vs integration time:

Some values of the counts were discarded because they maintained approximately constant for low integration times. A linear fit was made using the remained values. The slope of the linear fit indicates the dark current: 0,2098 ADU/s. This value can be converted to electrons per second, giving 1,095 e^{-}/s.

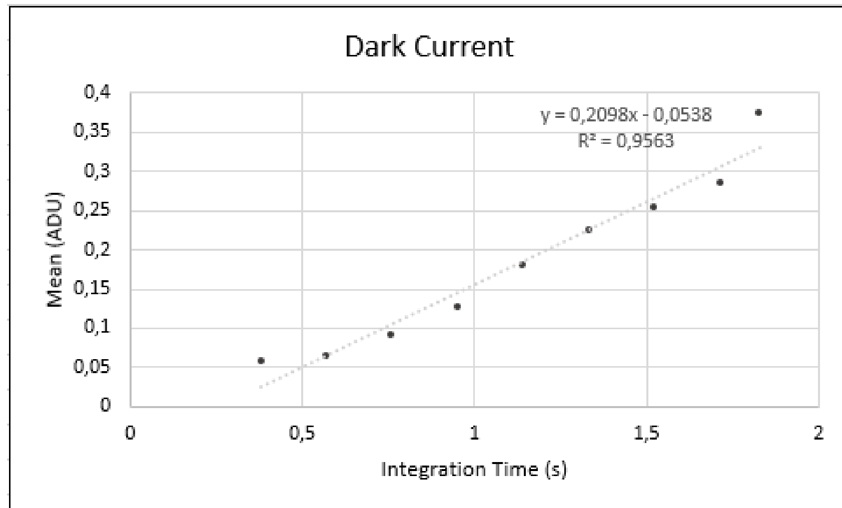


Figure 4.7: Dark current for the image sensor MT9J001

The value of the dark noise is measured according to the integration time chosen to the camera, using equation 3.9. The total noise is measured using equation 3.8.

4.3 Lens system selected for testing

The design of an ideal optical system to be used in a star tracker is not an easy task to do. The lens mounting is quite sensible, and the minimum disturbance in the system can cause an error in focus that leads to a poor attitude determination or a critical error in the system.

Figure 4.8: Commercial lens system to be used in the star tracker. This lens has a f-number of 1,6, with a focal length of 35mm.

The lens system to test the sensor and the algorithm is ideal to be a commercial one, in order to save time and costs. A lens with an adjustable aperture allows to test the system for several f-numbers which can be ideal if this property is not quite defined in the requirements yet. The chosen lens was one from Fujian, a chinese brand. This lens has a f-number of 1,6, with a focal length of 35mm. The FOV of the star tracker (lens system and image sensor), can now be measured using twice the inverse tangent of the ratio between the sensor's half height and the focal length as seen in figure 4.9. This angle was measured to be horizontally 10,5° and vertically 7,5°.

There is also a focus wheel that allows to focus the lens system for different distances. The lens

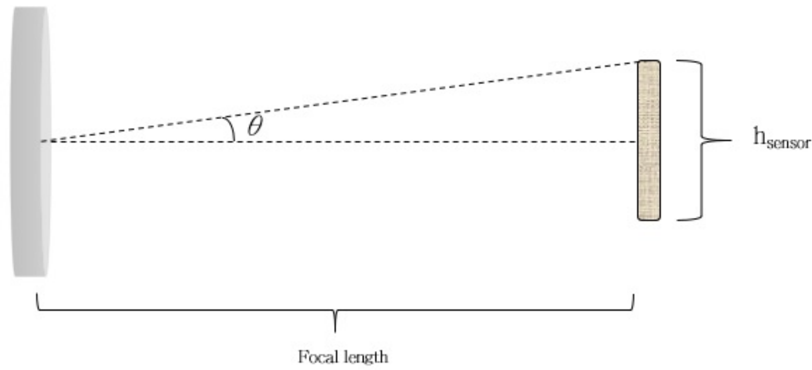


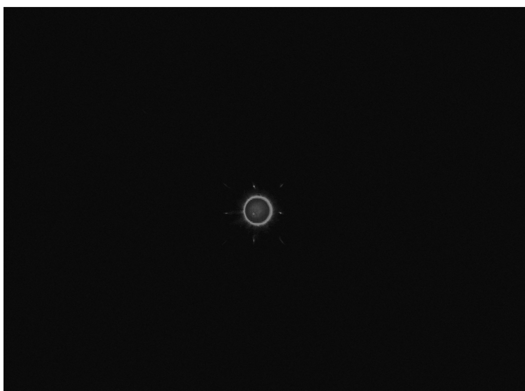
Figure 4.9: Representation of the FOV measurement. The FOV can be measured using the focal length of the lens system and half

system connects the image sensor board using a c-mount adaptor. The bad part of buying a cheap commercial lens online is that there is no datasheet for it, so the transmission coefficient can only be predicted.

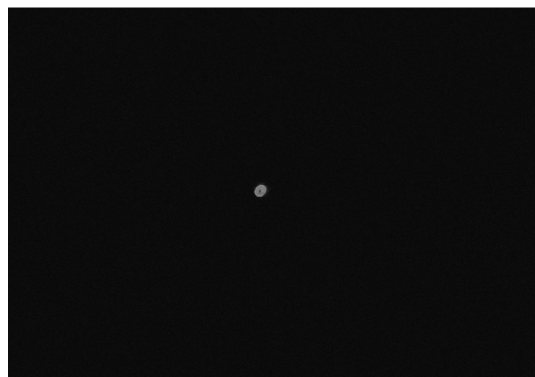
Focusing the lens system

Since stars are light years from Earth, in optics they can be considered to be in infinity as an approximation. The lens system acquired has an adjustable focus, which allows it to be focused to infinity. Due to its sensibility, it is not practical to focus it while taking pictures from stars. This is why the collimated beam generated by a Fizeau interferometer was used to focus the lens system.

The image sensor and the ArduCAM USB camera shield were mounted in a interferometer and connected by USB to a laptop computer. The integration time used for the focusing process was approximatly 1 ms and a an analogue total gain of 21 (decimal value). This values ensured there was no saturation on the surrounding pixels of the image formed.



(a) Frame taken with the image sensor MT9J001 mounted in the ArduCAM USB shield board before focusing.



(b) Frame taken with the image sensor MT9J001 mounted in the ArduCAM USB shield board after focusing.

Figure 4.10: Zoomed frames extracted with the image sensor and the lens system before (left) and after (right) the focusing process.

The focus wheel was adjusted until a round image with the minimum value of pixels was formed. The process of focusing is represented in figure 4.10. After focusing the image into a point , it was

determined using ThorCam (a software for image processing), that the number of mean pixels occupied by the image was 15. This means that the system is not perfectly focused. It was quite hard to achieve this value due to the sensibility of the lens' focusing wheel. An slightly unfocused star tracker should not be a serious issue, because star light spreads over more pixels and it may get easier to the algorithm to distinguish stars from hot pixels.

4.4 Prediction of the maximum magnitude values detected by the sensor

The image sensors can only detect stars whose light distribution counts in the image sensor are higher than noise values. The prediction of the stars that can be detected using a determined integration time can be made using the Johnson-Cousins system, presented in chapter 3 along with the noise values obtained in section 4.2.

4.4.1 Approximations and assumptions for measuring flux of an A0 V star type

The prediction of stars detected through their flux and magnitudes, needs to have in account the color for the stars. A-type stars like Vega, for instance, have a blue-whitish color. The flux emitted by an A-type star is not going to be the same as the flux emitted for another type star (i.e. other color stars), because they have different chemical compositions and, as a consequence, different emission spectrum. This means the magnitude limit for the image sensor may vary accordingly to the star's temperature and type. In order to predict the magnitude of the stars to be detected by MT9J001, only A-type stars were used. It was considered a star like Vega, with a temperature of 9500K and magnitude 0,03.

The spectral sensitivity curve could not be measured for MT9J001 since only a laser with a wavelength of 632,8 nm was used as a source of light to characterize the sensor (the spectral sensitivity curve needs measurements in a higher range of wavelengths). For this reason, the spectral sensitivity curve for the monochromatic Silicon CMOS presented in figure 3.3 in chapter 3 was used. Since this CMOS and MT9J001 are composed of Silicon and are monochromatic, their capacity of photon conversion should not differ much. The presented CMOS has a wavelength range from 400 to 1000 nm. This means only B, V, R and I bandwidths should be considered to measure flux. Values of flux for this bandwidths are present in table 3.1, in chapter 3. It was assumed that all bandwidths are represented by squares instead of curves, as represented in figure 4.11.

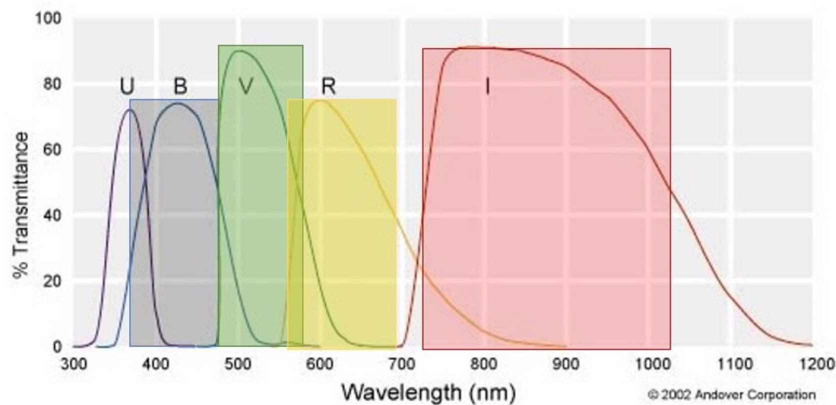


Figure 4.11: Approximation used in Johnson-Cousins system to measure flux for an A0 V type star.

4.4.2 Determination of magnitude maximum limits detected by the image sensor outside Earth's atmosphere

Using the assumptions previously mentioned, the flux emitted by an A0 V star in the bandwidths that are read by a typical CMOS Silicon image sensor can be approximated by

$$\Phi_0 = \sum_{bw=B,V,R,I} \Phi_b w \cdot FWHM_b w \cdot QE_{bw} \tag{4.2}$$

where QE_{bw} is the quantum efficiency value of the sensor for the wavelength that corresponds to the effective wavelength value of a determined bandwidth.

This approximation is quite optimistic, since overlaps between the squares are not considered. Also, a big part of the I waveband that is not even detected by the image sensor is considered for the counts. Same happens with a part of the U waveband. This assumptions give a higher flux than it was supposed too.

Once the flux for A0 V star is determined, equation 3.2 is used to measure flux for other magnitudes, in W/m^2. This flux is converted into photons per square meter and second using the relation in equation 3.4. Then, the resulting value is multiplied by the lens system transmission coefficient and aperture area in m^2, in order to have the number of photons that reach the image sensor per second. Since the lens system considered is not known, a transmission coefficient of 0,7 was considered. The number of photons that reach the image sensor per second can be divided by the number of pixels a star of this type is expected to occupy and multiplied by the integration time, which gives the signal.

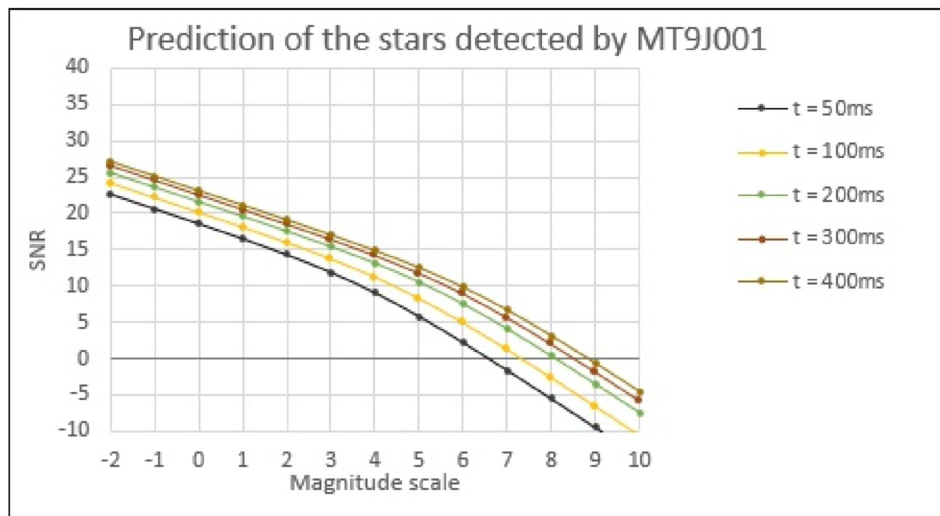


Figure 4.12: Prediction of the maximum magnitude detected by MT9J001 sensor for different integration times.

The signal's shot noise is given by its square root. Finally, the total noise was determined by applying equation 3.8 to the shot noise and the other noise values previously measured in section 4.2. The results for different integration times can be seen in figure 4.12.

4.5 Final balance

Figure 4.12 predicts the maximum magnitude limit for the image sensor presented is 6.5 for an integration time of 50ms in space. These results are not enough to guarantee the sensor will read stars

of magnitude 6.5 and superior, because of the approximations and assumptions made in the photometry study. Since quantum efficiency was not measured, the spectral sensitivity curve for the sensor was not used. It was used a typical curve for a Silicon CMOS image sensor instead. The values of limiting magnitude obtained should be corroborated with testing in space.

Chapter 5

Tetra Algorithm

The algorithm contains the method used by the star tracker to extract attitude. If an algorithm is not capable of identifying stars in microseconds, even with a perfect optical system, the whole purpose of the star tracker gets lost. A star tracker needs to be capable to extract attitude in an almost continuous way, in order to achieve high accuracy and comply its goals.

5.1 Modes of operation for a star tracker

Any star tracker has two main modes of operation regarding its algorithm: the tracking mode and initial attitude acquisition. The initial attitude acquisition is also called sometimes as lost-in-space mode. In this mode, the star tracker searches its FOV in order to find the brightest cluster of pixels, which correspond to the brightest stars, and computes centroids. Several properties of this stars such as brightness and relative distance between them, can be used to match them to several entries of a star catalog. This match can be achieved in microseconds, using sophisticated algorithms for fast pattern matching. After the matching of several stars with the star tracker catalog, the tracker follows them by choosing a region of interest (ROI). This process corresponds to the tracking mode.[32] After a fixed integration time, the star tracker reads the number of accumulated photo-electrons in the pixels in the ROIs around the expected position of the matched stars. The brightest pixels in each ROI are identified, and the appropriate block of pixels is used to compute the centroid of each star. This makes the process of determine the attitude of the satellite really easy and fast.

The slowest process of the star tracker is the initial attitude acquisition because of the extensive number of stars in the star catalog. This implies several accesses to the database in order to match the visualized stars with the stars in the catalog. This several accesses lead to a high computation time. Despite that, there are some new high sophisticated algorithms capable of determining the attitude with only one access to the database, being one of them Tetra algorithm.

5.2 Tetra algorithm

Tetra is a star identification algorithm created by Julian Brown and Keaton Stubis, two MIT Students. The creators defend that algorithms nowadays vary in their trade-offs between run-time, robustness and hardware requirements, which for them corresponds to the old rule of ”Fast, good, cheap pick two”. Tetra is an algorithm developed to be used in COTS hardware that promises to use the minimum possible computation time and number of database accesses to solve the problem of the initial attitude acquisition.

This goal is achieved by making use of hash tables, a generalization of arrays, which allows the program to identify star patterns with a single database access. The development of the code focused solely on the position of stars in the image, ignoring almost all the information which may be available in a night sky image like asteroids and planets. The brightness and color of the stars are also ignored during star identification.

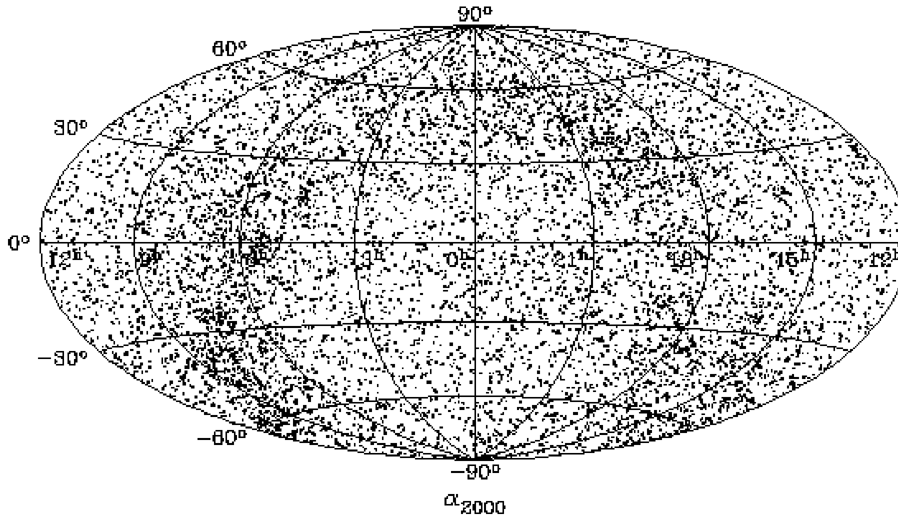


Figure 5.1: Sky coverage for Yale bright star catalog (BSC5), the star catalog used originally by Tetra algorithm. Image courtesy of Harvard.edu

Tetra uses Yale bright star catalog (BSC5), a star catalog containing 9110 stars with 6,5 magnitude or brighter. Each star is stored with its identification, ID, number and other relevant information like the right ascension and declination.

5.2.1 Storage of the internal catalog in hash tables

Hash tables are structures of data which implement arrays. The logic of functioning behind hash tables is very similar to the one from dictionaries. In a dictionary, each definition is assigned to its own word. In hash tables, each object has a matching key. In the specific case of Tetra algorithm, the object is the star and the key is its catalog ID. Each star has an array of information about itself useful to the algorithm.

The creation of hash tables starts by extracting the information for each star in the catalog to be used: right ascension, declination, magnitude and catalog ID. Since the computation time increases with the number of stars in the catalog, it doesn't make sense to store stars that cannot be detected by the image sensor. This is why stars with magnitudes above a determined limit specified by the user are ignored. Patterns of four neighbor stars are computed and the distances between them are divided by the higher one. Each star is stored in the hash table using its ID as a key. The matching object for each ID is the array with the normalized distances between that star and their neighbors. The hash table creation process only needs to be made once and, in a space situation, the star tracker flies only with hash tables already made. It is the task that requires more computational time: at least one hour.

5.2.2 Process of stars identification

The identification of stars usually takes milliseconds at most and starts after the acquisition of an image. The image is first converted into greyscale. The program creates a virtual inertial coordinate system with origin in its center. Some of the image parameters are then extracted: mean counts, standard deviation of the counts, height and width. After this extraction, the counts are read pixel by pixel. The pixels with counts equal or higher than 5 times the standard deviation, are selected to analysis. From these selected pixels, the ones that are neighbors are stored in a group. The groups near the limits of the image are excluded, as well as the ones without the minimum number of pixels to be considered stars (this number is usually 4 or 5 and it is defined by the user).

Figure 5.2: Pixelized star detected in an image. The light from the star occupies more than one pixel.

This exclusion process allows tetra to avoid false detection caused by debris or hot pixels. The remained groups are presumed to be stars and are from now on treated as stars. The center of mass for each star is assumed to be its brightest pixel. Each group is matched with the three groups that are closed to it, creating patterns of four groups (patterns of four presumable stars). The coordinate system previously considered allows for each pixel to have coordinates in the image plane.

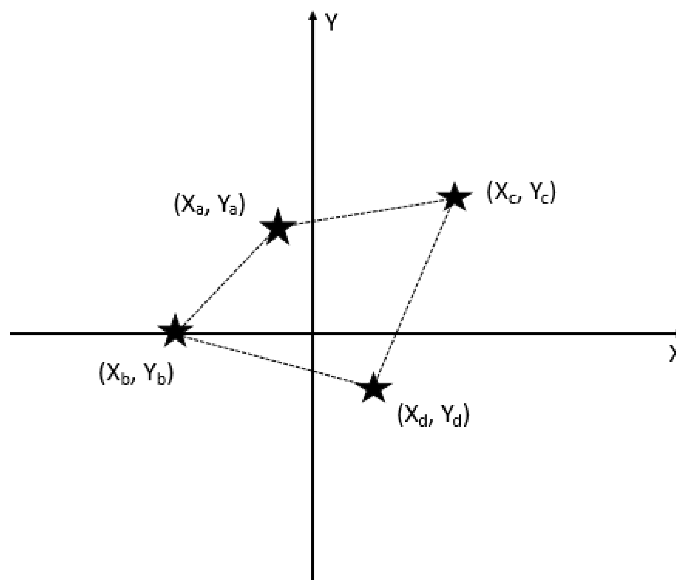


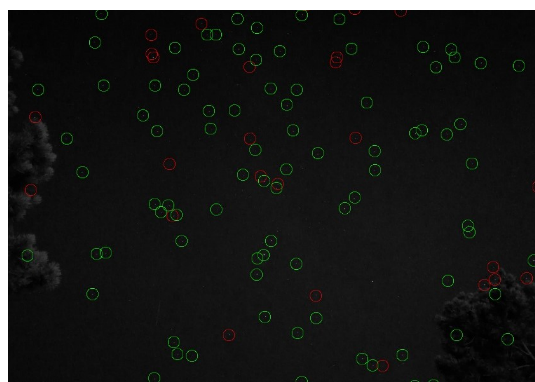
Figure 5.3: Virtual coordinate system created in the image by Tetra. To each star is given a coordinate. The distance between presumable stars can be extracted and normalized to be compared with the internal catalog.

Using this coordinates, the distances between same pattern groups can be calculated. In each pattern, distances are divided by the higher one. Each group has now an array of information with the distances to its neighboring groups. This arrays are stored and then compared with the ones from the catalog. The matched ones are identified by the ID number in the hash table and the right ascension, declination and roll are determined. The FOV and mismatch probability can also be estimated. Since Tetra works with patterns of four stars/groups, the image sensor requires to detect at least four stars in order to extract right ascension, declination, roll, FOV and mismatch probability.

5.2.3 Output

After the extraction of attitude parameters, the code gives as an output the greyscale version of the picture with all the recognized stars circled in red or green. The stars circled in green were the ones recognized by the algorithm and the ones circled in red are the stars that weren't recognized by the algorithm but were considered to be stars.

(a) Image taken with a cellphone before being analyzed by Tetra algorithm.



(b) The same image but this time after being analyzed by Tetra algorithm.

Figure 5.4: Image before (left) and after (right) being analyzed by Tetra algorthm.

There are several reasons for a presumed star to not be matched with the internal catalog. The most simple reason is that it may not be listed in the catalog. Not all the stars are present in the catalog, and there is also the possibility of it not being a star: it can be a planet or an asteroid, for instance. If the star is in the catalog, then maybe it has a magnitude number outside the algorithm's defined domain.

5.3 Migration to Hipparcos catalog

The probability of matching stars increases with the number of stars in the catalog. The accuracy of a star tracker depends on the number of stars identified by the algorithm. It is then reasonable to use an internal catalog with a high number of stars. In real space applications, star trackers tend to combine the information of several catalogs to achieve a higher number of stars. Tetra is a new algorithm and its behavior with a higher number of stars or with other catalogs is not known (at least not publicly). The algorithm can be changed to work with a more recent catalog, like Hipparcos. Hipparcos catalog was one of the primary products from a European Space Agency's astrometric mission. Hipparcos satellite extracted high quality scientific data from thousands of stars between 1989 and 1993, which took to the production of a catalog with 118218 stars with magnitude 12,4 or brighter. Hipparcos database is

available online on VizieR in a file called hip_main.dat. The information in this file can be directly read by tetra algorithm.

5.3.1 Alterations to Tetra algorithm

Unfortunately, the use of a higher dimension catalog implies more computation time when creating hash tables. In order to minimize some computation time of hash tables, a new code was created in C just to delete blank lines and information not needed from the VizieR database. The remaining information is stored in a new file. This code can be consulted in Annex B.1. The way Tetra reads the catalog also needed to be changed. The original code was prepared to read 32 bit integer values for star ID's. This was enough for BSC5, but it is not enough for Hipparcos catalog, since it has roughly 10 times more stars. The code was changed to read 64 bit integer values, which covers Hipparcos ID's values. Tetra was also coded to perform with old versions of Python libraries (Numpy and Scipy), which could be an inconvenient. An update to most recent libraries was also made. All the changes are available to be consulted in Annex B.2.

5.4 Algorithm's performance

In order to understand how tetra would react to the catalog migration, ten pictures were used to test both versions of the code. Four pictures from the sky were taken with a cellphone and six random pictures were extracted from several places of the internet. The results for right ascension and declination were also compared with Astrometry.net, a calibrationless lost in space algorithm that identifies patterns of 4 stars in a very similar way tetra identifies them [33].

5.4.1 Testing conditions

The cellphone used to extract images had a f-number of 2, focal length of 35mm and a CMOS image sensor (Sony IMX 2014). The image sensor is a 13 MP sensor (4160 x 3120), and has a pixel size of 1,12 μm. The algorithm was tested in an ASUS computer with 4GB RAM and processor Inter(R) Core(TM) i3-2365M 1.40GHz.

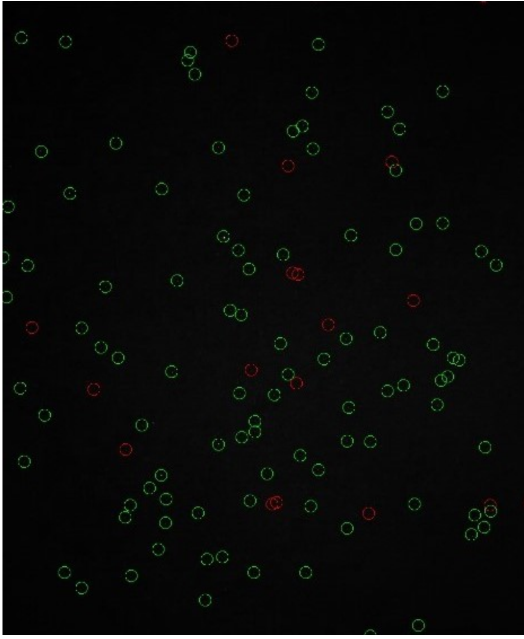
All of the pictures were tested with both versions of the catalog. Both codes ran with the default configurations which limited the maximum magnitude to 5.5, the minimum number of 3 pixels to classify a group as a star and patterns of 5 stars.

5.4.2 Results

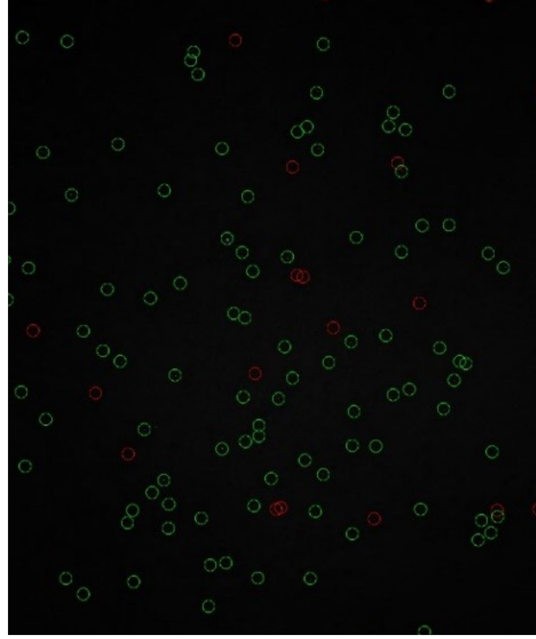
Both codes started by compute their hash tables. The difference starts right on the hash tables. With BSC5 catalog the algorithm found to compute 2887 stars, and with Hipparcos the algorithm found 2952.

For both tests, all ten pictures were put in the same folder and analyzed with only one run. Tetra algorithm took 102,15s to identify all images with BSC5 catalog and 110,367s to identify all images with Hipparcos catalog, which gives a mean of 10,2 s per picture to BSC5 catalog and a mean of 11s oer picture for Hipparcos catalog.

Pictures 1 to 6 were the ones extracted from a google search and pictures 7 to 10 were the ones taken with the cellphone.



(a) Picture analyzed by Tetra algorithm with BSC5 catalog



(b) The same picture analyzed by Tetra algorithm with Hipparcos catalog.

Figure 5.5: Picture 5 analyzed with Tetra algorithm, using BSC5 catalog (left) and Hipparcos (right). It is visible that Hipparcos made the algorithm capable of analyzing more stars.

Right Ascension, declination and roll

The results obtained for right ascension, declination and roll are respectively represented in tables 5.1, 5.2 and 5.3.

	Right Ascension values (^{o})		
	BSC5 Catalog	Hipparcos Catalog	Astrometry.net
Picture 1	242,0269	242,0295	241,4280
Picture 2	189,3434	189,3439	189,4530
Picture 3	94,4787	94,4843	94,4780
Picture 4	256,3333	256,3283	256,1750
Picture 5	270,6980	270,6975	270,3210
Picture 6	345,6992	345,6994	345,1840
Picture 7	252,1971	252,1915	Failed
Picture 8	296,0123	296,0069	295,9980
Picture 9	294,5795	294,5787	294,5540
Picture 10	293,6867	293,6887	293,6800

Table 5.1: Tetra algorithm performance analysis: right ascension values

Astrometry.net was not successful in extracting pointing values for picture 7. In general, Hipparcos values are slightly closer to astrometry.net values. However, both catalogs results are closer to each other than to astrometry.net. The values between both catalogs were already expected to be slightly different due to the increasing of stars in the catalog from BSC5 to Hipparcos migration.

Declination Values (º)			
	BSC5 Catalog	Hipparcos Catalog	Astrometry.net
Picture 1	32,3290	32,3232	32,0180
Picture 2	71,0563	71,0595	70,9790
Picture 3	-3,3886	-3,3905	-3,4160
Picture 4	-28,5721	-28,5651	-28,6190
Picture 5	-26,4272	-26,4254	-26,3630
Picture 6	59,1821	59,1818	59,2330
Picture 7	10,1850	10,1849	Failed
Picture 8	14,8265	14,8267	14,8150
Picture 9	37,1487	37,1497	37,1240
Picture 10	35,9500	35,6887	35,9620

Table 5.2: Tetra algorithm performance analysis: declination values

The maximum difference between declination values obtained with BSC5 and Hipparcos catalog is 0,2613º, for picture 10. For right ascension, the maximum difference between both catalogs is 0,0056º, for pictures 3 and 7. For roll, the maximum difference registered was 0,0270º for picture 2.

Roll values (º)		
	BSC5 Catalog	Hipparcos Catalog
Picture 1	34,0370	34,0373
Picture 2	358,3251	358,2981
Picture 3	336,7343	336,7342
Picture 4	83,5900	83,5953
Picture 5	334,3993	334,3973
Picture 6	218,1240	218,1227
Picture 7	354,1567	354,1595
Picture 8	4,7674	4,7502
Picture 9	176,5585	176,5595
Picture 10	91,8120	91,8104

Table 5.3: Tetra algorithm performance analysis: roll values

Roll values were not compared with Astrometry.net, since Astrometry.net only extracts right ascension and declination.

Field of view and mismatch probability

In the particular case of this work, the field of view of the star tracker is already known to be 10°. However, it order to understand if the migration was successful, it was important to compare both output results from Tetra. The FOV values for both catalogs are shown in table 5.4.

The maximum difference on the values registered for both catalogs was 0,0304 rad for picture 2.

Field of View (rad)		
	BSC5 Catalog	Hipparcos Catalog
Picture 1	56,1006	56,0838
Picture 2	61,8789	61,9093
Picture 3	63,5986	63,5982
Picture 4	62,1588	62,1590
Picture 5	54,4288	54,4284
Picture 6	65,3178	65,3179
Picture 7	62,7671	62,7689
Picture 8	62,5243	62,5210
Picture 9	62,5373	62,5385
Picture 10	49,1531	49,1554

Table 5.4: Tetra algorithm performance analysis: right ascension values

The mismatch probability gives the probability of false star detection. These values are presented in table 5.5.

Mismatch Probability		
	BSC5 Catalog	Hipparcos Catalog
Picture 1	3,325E-52	2,023E-60
Picture 2	1,049E-63	1,217E-57
Picture 3	1,521E-24	7,369E-88
Picture 4	6,826E-111	3,382E-119
Picture 5	3,618E-60	1,130E-65
Picture 6	7,432E-70	5,276E-71
Picture 7	2,315E-34	2,245E-34
Picture 8	2,847E-43	7,282E-46
Picture 9	2,076E-114	5,159E-118
Picture 10	4,962E-124	3,121E-124

Table 5.5: Tetra algorithm performance analysis: mismatch probability

The mismatch probability decreases with the number of stars so it was expected to be smaller for Hipparcos catalog. By analyzing the table, it is quite evident this is true.

Final balance

By analyzing the results it is secure to say that the migration was successful. The mismatch probability decreased as expected and the attitude values were quite similar between both catalogs. It is safe to use Hipparcos catalog for nocturnal testing. The algorithm runtime was much less at the beginning of this work (approximately 8ms per picture). However, the computer used for this work degraded along with time and use. It was not possible to replicate the work made by Julian Brown and Keaton Stubis.

Chapter 6

Mechanical and electronic system implementation

Star trackers are usually integrated in the ADCS board of a cubeSat. The ADCS board in turn, has a direct communication with the on board computer, which connects the whole subsystems of a cubeSat together. It is not possible in this work to test the star tracker developed in a cubeSat mission, in Space. However, it is possible to test it in a single board computer, like Raspberry Pi.

6.1 Raspberry Pi implementation

The MT9J001 image sensor together with ArduCAM USB shield board allows it to be connected to single-board computers like Beaglebone and Raspberry Pi. ArduCAM USB camera shield board already limits the frame rate to 6 fps maximum when connected to a computer. When connected to a single board computer with an USB cable, the frame rate declines since single board computers have weak processors. Raspberry Pi 3 B+, is equipped with a 1.4GHz 64-bit quad-core processor, which guarantees the frame rate is kept acceptable for lower integration times. Some relevant Raspberry Pi specifications are presented in table 6.1.

Raspberry Pi Specifications	
Processor	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC 1,4GHz
RAM	1GB LPDDR2 SDRAM
Ports	Ethernet, 4 USB, HDMI MicroSD

Table 6.1: Raspberry Pi Model 3 B+ Specifications, taken from the Raspberry Pi website.

This Raspberry Pi model also has Wi-Fi implemented which can be used to test the autonomy of the star tracker via Secure Shell (SSH). SSH is a cryptographic network protocol for operating network services securely over an unsecured network. This means the implementation of the image sensor with the Raspberry Pi can be controlled by Wi-Fi.

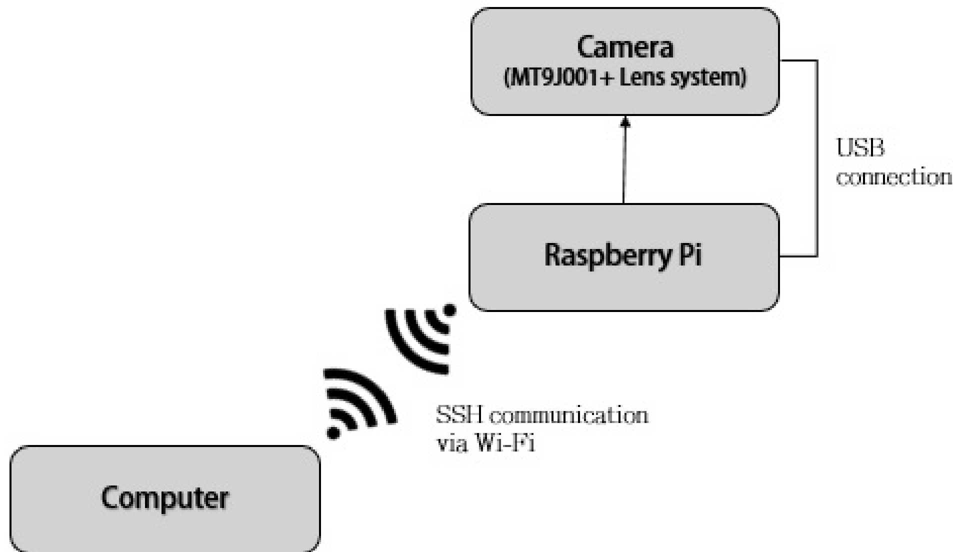


Figure 6.1: Schematic functioning of the star tracker implemented in Raspberry Pi. The computer gives the order for Raspberry Pi to run the main code via SSH. The code opens the camera, extracts a frame and runs Tetra algorithm on it. The output is the attitude which is shared directly to the computer also via SSH.

It is possible to have an autonomous star tracker this way and study how it performs in a Raspberry Pi. For that reason, it is important to program the Raspberry Pi board to be controlled by an external computer, which in a real life space mission would be controlled by ground control.

6.1.1 Configuring Raspberry Pi and its communication with the external computer

Raspberry Pi board needs an operating system installed on an SD card to function. The recommended operating system to be installed is Raspbian, the official Raspberry Pi operating system available in their website. The communication via SSH and ethernet with the main computer should be enabled. The ethernet connection allows the Raspberry Pi to be programmed easily and to smooths the process of correcting possible errors that may show during code implementation.

An instruction manual about how to prepare Raspberry Pi to connect ArduCAM USB camera shield board and the computer was developed and can be consulted in Annex C.

6.1.2 Programming ArduCAM board to function as a star tracker and share measurements with the external computer

There are two codes available in ArduCAM github to be used in Raspberry Pi with ArduCAM USB camera shield board and MT9J001. One of them, uses Python 2 and the other uses C. Since Tetra algorithm works with Python 3, it is convenient to use the same version for both codes. However, until the date, only the libraries for Python 2 are available in the ArduCAM github. Both codes were tested in order to take pictures and the conclusion is that the C code has more fluidity. Python code has some bugs, and the frame is not always extracted when desired. It is possible to merge both ArduCAM USB shield codes and tetra algorithm code in a bashfile, despite being written in different languages. However, this shall not be a definitive solution, since it takes a lot of computational time and should be used only for testing the star tracker in the night sky. Both codes need to be merged in order to create a flux of tasks that a star tracker has. This flux of tasks can be consulted in figure 6.2.

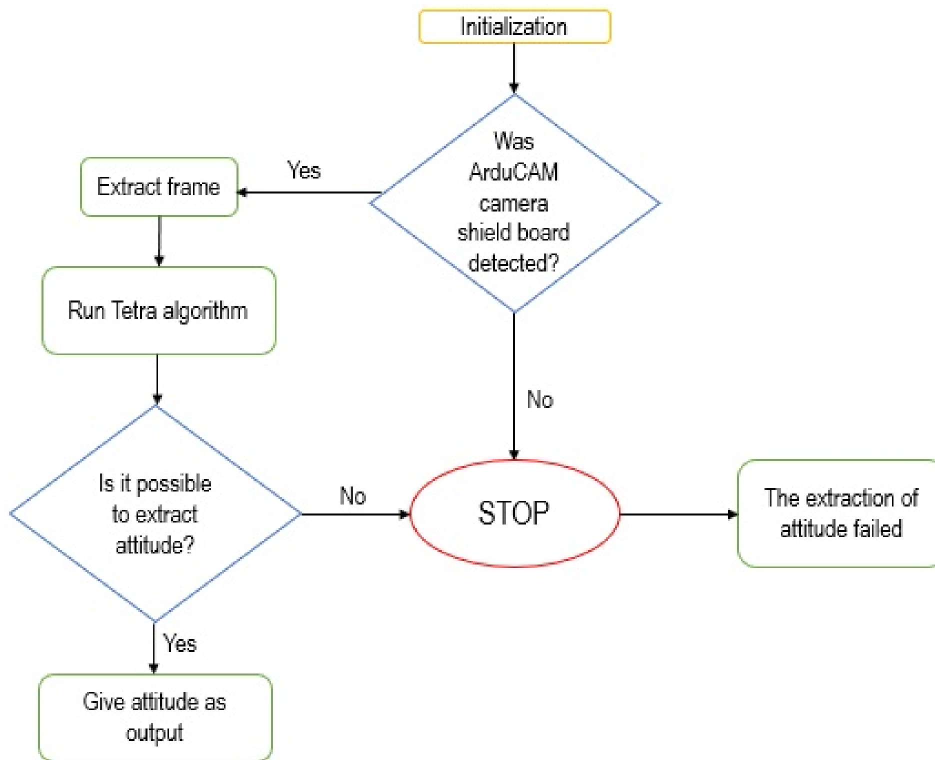


Figure 6.2: Fluxogram for the star tracker's code. This code is in a bashfile and is a fusion between tetra algorithm and ArduCAM camera shield board's code.

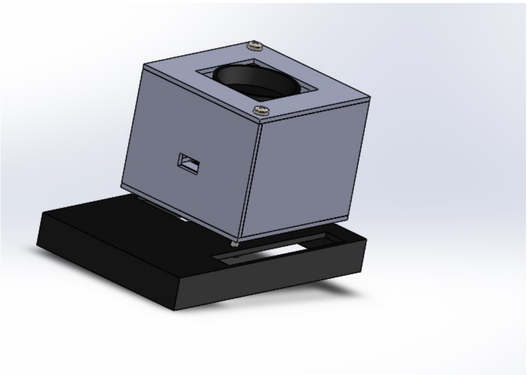
6.2 Mechanical design of the star tracker's structure

Due to the sensibility of the boards it was important to design some structure box to sustain the camera. A structure box can protect the electrical components from unexpected adverse weather conditions during nocturnal testing, like a sudden rain. A default Raspberry Pi case was already acquired with the Raspberry Pi board in order to save resourceful time. This case is shown in figure 6.3.

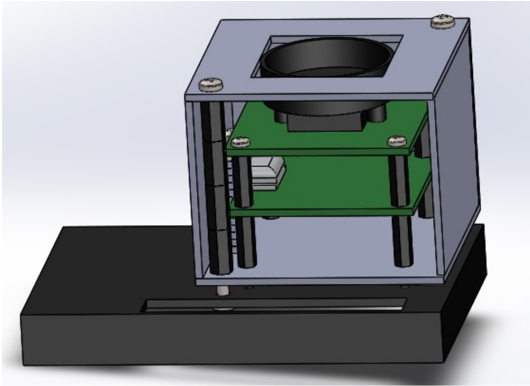
Figure 6.3: Raspberry Pi Model 3 B+ and its official case

Unfortunately, there is no commercial default case for MT9J001 and the ArduCAM USB camera shield. This is why a box case to hold and protect the camera was designed using Solidworks, a CAD software. The designed case has a cubic format and it is designed to hold the image sensor and the

ArduCAM USB camera shield board. The architecture is quite simple and it is presented in figure 6.4. It is composed by 4 lateral pieces, a top piece and a bottom piece. All lateral and bottom pieces are glued to each other. Two M3 spacers are placed in two opposite inner corners of the designed case: they cross the whole cubic case (from top to bottom), and two holes made in the Raspberry Pi's case top, in order to connect both. They are fixed to the Raspberry Pi's case using two M3 nuts. The top part seals the cubic case using two M3 screws.



(a) Main view of the star tracker's case designed using CAD software



(b) Lateral sliced view of the star tracker's case designed using CAD software

Figure 6.4: Case designed in Solidwords. The figure on the left shows the main view of the case and the right figure shows a sliced lateral view. The image sensor, the ArduCAM USB shield board and the Raspberry Pi's case are already represented in this design.

The center hole in the top part of the case was originally designed to have the same round shape as the lens system, in order to surround it. However, the case was developed using a manual milling machine, and it can be quite difficult to achieve a perfect round hole. This is why it was decided to change the hole to be square and not round. The material used in the case is FR4, a glass-reinforced epoxy laminate material. The glue used to connect the lateral pieces and the bottom piece was epoxi.

Figure 6.5: Case designed to protect the image sensor and ArduCAM USB camera shield board with the help of a manual milling machine. After the mounting with epoxi, the box was painted in grey.

The final result is presented in figure 6.5. The box has a lateral hole for the USB cord that connects

the ArduCAM USB camera shield board to the Raspberry Pi.

All components regarding the prototype star tracker developed in this work are now connected and presented in figure 6.6.

Figure 6.6: Star tracker developed in this work.

The star tracker is now mounted and ready for nocturnal tests. Nonetheless, Raspberry Pi needs external power to function. An external battery can be used in order to test the full functional star tracker outside.

Chapter 7

Final tests and results

The only way to know if the image sensor and lens characteristics chosen in this work are ideal to be implemented in a future prototype that fulfills all requirements listed in chapter 3, is to test the performance of the star tracker in the Earth's night sky. This chapter resumes the final tests conducted in order to characterize the star tracker developed in this work.

7.1 Nocturnal testing

The accuracy of a star tracker is an important parameter that can only be measured by testing it several times in determined place. The attitude values resulting from tests can be compared to the ones from Astrometry.net or Stellarium[1].

In chapter 6, it was explained that the Raspberry Pi's board needs an external battery in order to test the star tracker outside. The battery available for this work however, had not enough capacity to sustain the star tracker for the testing time required. For that reason, the star tracker (image sensor board, ArduCAM USB camera shield and lens system), was directly connected to a laptop with an USB cable. The default ArduCAM USB camera shield board software was used to help define register values and extract frames.

7.1.1 Test plan and required test conditions

This master thesis was developed in Lisbon, the capital of Portugal, also known for its light pollution. The star tracker needs to be tested in an environment with low light pollution, in order to use the lowest as possible values for the gain and integration time. Knowing this fact, two places relatively dark in the surroundings of Lisbon were chosen to test the star tracker. One of them is the faculty's roof top and the other Mata da Machada woods. The GPS coordinates of both places were extracted using google maps and are listed in table 7.1.

Test location	Latitude (°)	Longitude (°)
University	38.756916	-9.156896
Woods	38.616757	-9.047124

Table 7.1: GPS locations of the places chosen to test the star tracker.

[1] Stellarium is an open-source free-software planetarium, that uses the Hipparcos catalog.

The nocturnal tests can be divided in two phases.

First phase

In the first phase, at the university, the main goal is to choose the proper registers (e.g. integration time and gain), to be used in Earth testing. The algorithm has a default value of 5.5 for the maximum magnitude to be recognized. If a higher magnitude can be detected by the image sensor, this value should be changed in order for the algorithm to use more stars in hash tables. This will increase accuracy of measurements and decrease mismatch probability. The maximum magnitudes detected in each frame can be checked using Astrometry.net.

The sensitivity of the lens system focus is not known, and the star tracker can easily be unfocused with transportation. This is why the first test is made at the university. The optics laboratory is placed there and if needed, the star tracker can be rushed into the optics lab to be refocused in the interferometer. In this first test, several frames are suppose to be extracted with different integration times and gain, in order to find the perfect trade-off between these two properties for the test location chosen.

Second phase

The combination for both registers of integration time and gain found in the first test phase is going to be finally used in the second and final phase of the test. This test has as a main goal to find the accuracy of the star tracker. Several frames are extracted without moving the star tracker. The results from the algorithm are compared with results from Stellarium. This final test is going to be conducted in the woods.

It is extremely important to extract the GPS coordinates of the tests' location along with the geographic coordinates, date and time a determined frame was taken. These parameters can be inserted in Stellarium in order to know the right ascension and declination of the star tracker at that moment. The right ascension and declination values can be compared with the ones from Astrometry.net, in a similar way it was already made to test both star catalogs.

7.1.2 First test results

The test at the university was made by extracting 17 frames with different values of gain and integration time registers. The maximum magnitude detected by the image sensor was found using Astrometry.net. The results can be found in table 7.2.

Frame number	Gain	Integration Time (ms)	Maximum magnitude detected (Astrometry.net)
1	8	190,5	Failed
2	8	190,5	Failed
3	8	190,5	Failed
4	8	190,5	Failed
5	37	571,4	5,25
6	37	571,4	5,25
7	37	1828,4	5,58
8	80	571,4	5,17
9	37	380,9	5,25
10	37	380,9	5,25
11	80	190,5	4,35
12	80	190,5	4,35
13	80	59,5	Failed
14	80	59,5	Failed
15	80	83,33	Failed
16	80	83,33	Failed
17	80	571,4	5,17

Table 7.2: Maximum magnitude values detected by the star tracker.

By analyzing the table it is possible to see that the first four frames and frames 13 to 16 failed to be analyzed in Astrometry.net. The best combinations where determined to be 9 and 10: gain of 37 and integration time of 380,9 ms. In frames with high gains (80), the quality of image presented a higher signal to noise ratio when compared to the ones with lower gains.

At the end of the test, all frames were analyzed with Tetra algorithm, because it is also important to know if these register values and conditions are ideal for the algorithm to recognize stars.

Figure 7.1: Frame extracted with star tracker after tetra algorithm extracted attitude The algorithm only identified attitude in this frame. At the right of the frame it is possible to see Vega star, occupying a high number of pixels.

One frame (9), represented in figure 7.1, was useful to extract attitude. Attitude values are presented

in table 7.3. The algorithm could not identify stars on the other frames although it was quite clear that much more than five bright stars were present in the frames (the minimum number of stars needed to extract algorithm is five, as said previously on chapter 5).

Right Ascension (º)	Declination (º)	Roll (º)	FOV (rad)
87.3213	69.1767	264.6974	2312.519

Table 7.3: Identified frame results for Tetra algorithm.

Usually the focusing is not problematic for a star tracker, as it was mentioned in chapter 4. Nonetheless, by analyzing the frames it was quite clear that the star tracker is not focused and that the number of pixel a star occupies is too high (more than 25 pixels for some of them), even for lower integration times, which can be the root of the problem. The algorithm may not be prepared to deal with such a large number of pixels for only one star. The proof of that is that Vega star, a star with magnitude approximately 0, was not recognized by the algorithm in frame 9. When analyzing the frames with Thorcam it became clear that the star tracker had a focusing problem. This analysis is shown in figure 7.2.

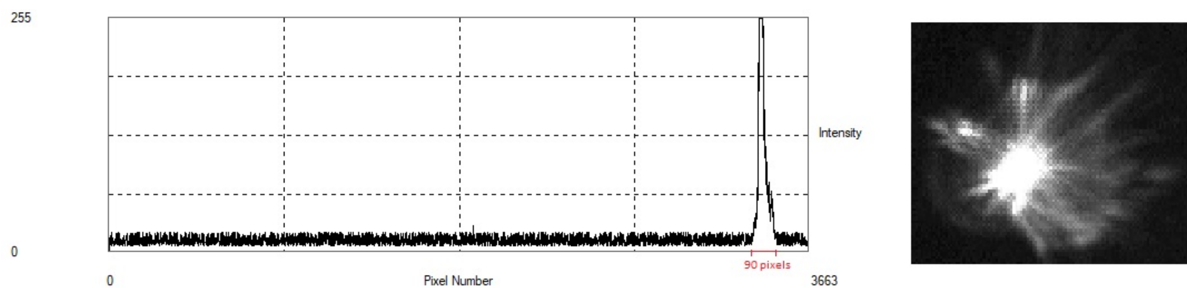


Figure 7.2: Magnification of Vega star and its analysis using Thorcam. A linear profile of a frame 9 at the center of Vega was made in order to analyze the diameter of the star in the frame. In some pictures Vega star occupies 90 pixels. This adds more proof to the theory that the algorithm is not recognizing the stars because the number of pixels a star is occupying is too high.

7.1.3 Second test results

The vibration of the machines from the university's rooftop could have perturbed the system's equilibrium and lead to a poor focus during tests. Although the results from the first test were not promising, it was decided that the star tracker should be tested again for similar parameters of frame 9, but this time in the woods.

In this location the star tracker can be fixed to the ground, and there is no risk of movement since the woods are an isolated place. Hundreds of frames were extracted in the woods during an hour, as predicted at the beginning of this work, for the following combinations of integration times and gain represented in table 7.4.

The algorithm was again tested for all frames and none was recognized by the algorithm.

Set	Gain	Integration Time (ms)
1	15	190,5
2	25	380,9
3	35	190,5
4	35	571,4

Table 7.4: Combinations of gain and integration times used to test the star tracker.

7.2 Discussion

The algorithm was able to extract right ascension and declination values for one frame. By analyzing the frames extracted in the first test, it was evident that the focusing of the lens system could be a problem, because stars were occupying a large number of pixels. The focusing wheel of the lens system is too sensitive, and it could be moving through frames. This may be the reason why the algorithm failed to identify the stars present in the other frames.

If the frames extracted in the second test could have been recognized by the algorithm, the conclusion would be that the vibration of machinery in the university's rooftop was perturbing the focus of the lens system. One explanation is that the lens system could be sensitive to transportation. Despite being focused previously in the interferometer, the transportation from the university to the woods could have compromised the focusing. Therefore, the number of pixels a star occupies is too high to be recognized by the algorithm.

Chapter 8

Conclusions

This chapter presents the conclusions taken from this work. Improvements to be made to the star tracker are also suggested at the end, so that the steps that should be taken to develop a full functional star tracker prototype are understood.

8.1 Achievements and difficulties

The development prototype of the star tracker designed in this work was successfully implemented in Raspberry Pi and right ascension and declination were extracted from one frame. Despite the promising results, the star tracker failed to extract attitude in most of the frames taken. A review of all the work done is presented and the topic on why the star tracker was not able to extract attitude is also discussed.

Image sensor

A characterization of the image sensor's noise was made in the optics laboratory of the University, using a laser beam and an integration sphere. The dark current measured had a value of 1,095 e^{-}/s for 24°C and the readout noise a value of 11,97 e^{-}. Both values were determined by analyzing frames extracted by the image sensor. This experimental work had several problems. The quantum efficiency was supposed to be measured, however the value obtained with the laser beam used made no sense, due to extremely high fluctuations in the measurements. Also, the spectral sensitivity curve could not be measured due to the lack of sources for different wavelengths. That's the reason why this measurements were not included in this work.

The USB input of the ArduCAM USB camera shield board also broke during tests. While trying to weld the USB input in the board, the heat damaged the image sensor. A new sensor and ArduCAM USB camera shield board had to be bought in order to continue this work, which delayed the experimental phase of it. This is the reason why a simple case to protect the image sensor and ArduCAM USB camera shield board was designed and developed in the university's milling machine. Since these electronic components would be exposed to the night sky, it would be important to have them the most protected as possible.

Lens system

The lens system chosen (f-number 1.6 and 35mm of focal length), along with the MT9J001 sensor from Aptina, resulted in a star tracker with a horizontally FOV of 10,5° and a vertical FOV of 7,5°.

A theoretical study of star photometry was also conducted in parallel with the experimental work made to characterize the image sensor. This study tried to answer the question "is this sensor adequate for space?". It was made in order to predict the flux that achieves the image sensor for different magnitudes of A-type stars in space. The flux corresponds to the signal that achieves the image sensor. Through the signal, the shot noise was determined, along with the total noise. Using the signal and the noise, the signal-to-noise ratio was calculated for each magnitude in dB, for different integration times. A signal-to-noise ratio higher than 0 meant that that magnitude could be read by the image sensor. It was determined that the image sensor was capable of reading a magnitude of 6,5 with 50ms, and a magnitude of 7,5 with 100ms. This means MT9J001 is probably adequate for space.

Algorithm and Electronics

The original algorithm was tested for 10 pictures. Some of them were taken from a cellphone and others were random pictures from the internet. After the migration to Hipparcos, a more extended catalog, the code was tested again in order to compare the results with the original catalog and Astrometry.net. The algorithm runtime obtained was higher than expected: 10s per picture for the original catalog and 11s per picture for Hipparcos catalog. At the beggining of this work, the algorithm runtime for both catalogs were less than 1s per picture. The computer used for this work was a ASUS computer with 4GB RAM and processor Inter(R) Core(TM) i3-2365M 1.40GHz. A problem with this computer's disk was detected and with time, the use of the computer to test algorithms and install software damaged the disk even more. This is the most probable reason why the algorithm exceeds 1s per picture. It was not possible to replicate the results from the algorithm authors.

The star tracker was implemented in Raspberry Pi. A bash file containing the code for ArduCAM USB camera shield and MT9J001 and the code for tetra algorithm was created. The star tracker can communicate with an external computer by SSH and work in a similar way it should work in space.

Nocturnal testing

The full system with the Raspberry Pi board was not tested in the night sky, due to the need of a powerful external battery. It was decided that only the star tracker (image sensor, lens system and ArduCAM USB camera shield), would be tested in the night sky using a USB cable to connect an external computer. Hundreds of frames were extracted from two different locations. Despite that, it was possible to extract the attitude from only one frame. Although the algorithm was successful tested in one night sky frame, in the vast majority failed to identify attitude. As discussed in the previous chapter, this could be because of the lens system (focusing wheel too sensitive), or because of star focusing: the number of pixels a star occupies may be too high to be recognized by the algorithm default values.

8.2 Final Outcome

The development prototype proposed and designed in this master thesis is not close to a final prototype to be implemented in a cubeSat mission. Nonetheless, it is quite difficult to design and develop a star tracker for cubeSats, and this document proves that with some iterations and hard work, a full functional star tracker can be achieved even by university students, if they have the proper equipment. This work presents valuable piece of research in order to prevent common mistakes while designing the future star tracker prototype. In the next section, some improvements and future work are suggested in order to achieve a final functional star tracker

8.3 Future work

The main goal of this master thesis was to design and test a development prototype of a star tracker close to a prototype with the following requirements:

1. The FOV shall be 10°.
2. The lens system should have a f-number lower than 2,5.
3. The full star tracker (image sensor, processing board and lens system), should weight less than 250g.
4. The full star tracker volume should not be higher than 50 mm^3.
5. The algorithm runtime shall not be higher than 1s.
6. The star tracker shall resist the temperature from -20°C to 40°C.
7. The magnitude limit shall not be less than 5.
8. The accuracy should be better than 20 arcsec.
9. The star tracker shall resist a radiation test of 10Krad from a proton source.

Although requirements 1 and 2 were fulfilled with the image sensor and lens system chosen, it was not possible to fulfill all requirements. Some improvements should be applied in the future if it is desired to achieve a prototype work.

Radiation and thermal tests

Requirements 6 and 9 implied that thermal and radiation tests had to be made to the development prototype designed in this work. It was not possible to test the star tracker for neither one of them, since the required costs for this type of tests were not justified for this work (the development prototype is not a final prototype, only a proof of concept). It should also be interesting to test other image sensors for radiation, and see how their noise varies in the presence of different doses.

Algorithm

The algorithm could not be tested for a higher magnitude limit. The computer overheated every time the hash tables were being computed for a higher magnitude than 5.5. It would be interesting to test if the attitude would be extracted for the frames obtained if a higher magnitude was defined, although it was already proven the problem might be in the lens system.

The algorithm runtime should be tested with more powerful computers than the one used in this work. The algorithm should also be tested with other configuration beyond the default ones.

Optical system

The lens system chosen is not ideal for a star tracker. This is because first, the lens system is not manufactured by any known brand. It doesn't have a datasheet and its lens characteristics are not known. Secondly, the focus wheel is too sensitive to be used in tests. Since the optical system is probably the reason why the star tracker didn't work, it is mandatory to design a new system to be tested with MT9J001 or other image sensors, like MT9P031 from On-Semiconductor. This sensor has shown to be quite promising to be implemented in star trackers as can be seen in reference [32], due to its natural radiation tolerance.

The f-number gives an important relation between a lens aperture and the focal distance. For that reason, an ideal value of 1,6 to 2,5 f-number was desired for the lens system. Several combinations with

different types of commercial lens were tried in Zemax. A particular one using two commercial lens from Edmund optics was designed and presented in figure 8.1. This lens system has a total length of 17,38 mm, f-number of 2,26 and focal length of 2 mm.

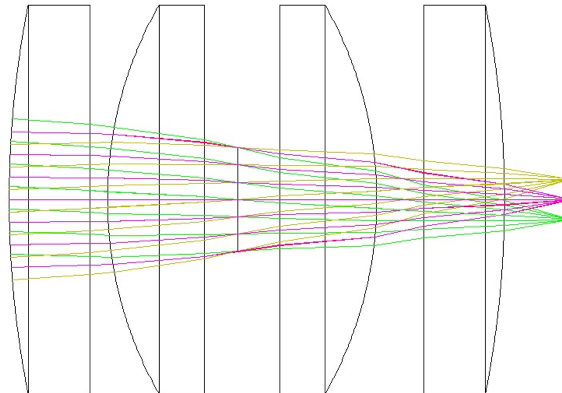


Figure 8.1: Lens system proposed to be tested in the future. This system has an f-number of 2,26

This lens system is not perfect: it has a spherical aberration at the corners of the image as can be seen in figure 8.2, for Blue wavelengths. In a similar way as a small unfocus, spherical aberrations are not problematic if they are not very pronounced. What counts is the final pixel value, as proven in this work.

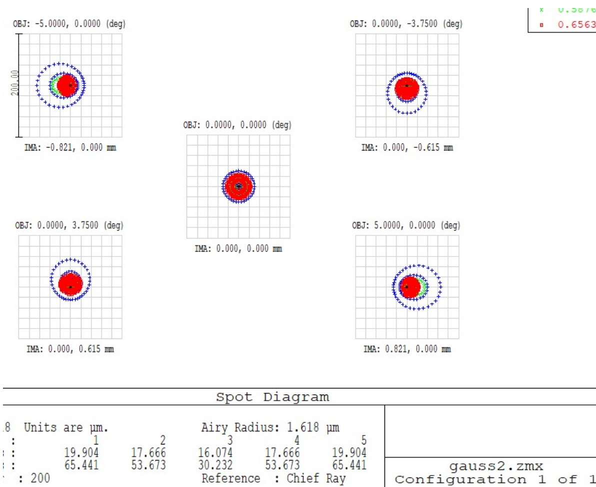


Figure 8.2: Spot diagram of the lens system proposed to be tested in the future.

Electronics system

Raspberry Pi is not an ideal electronic system for a star tracker. The Raspberry Pi's board is too voluminous and heavy to be implemented in a cubeSat, and this is why requirement 3 was not fulfilled.

A processor board needs to be designed to communicate with the image sensor and run the algorithm. The connection between both systems, optical and electronics, should be parallel. The parallel connection guarantees the maximum frame rate for the image sensor. With a runtime of less that 1s for the algorithm, it is possible to have an almost continuous extraction of attitude.

An electronic system is proposed in figure 8.3 to be implemented in MT9P031.

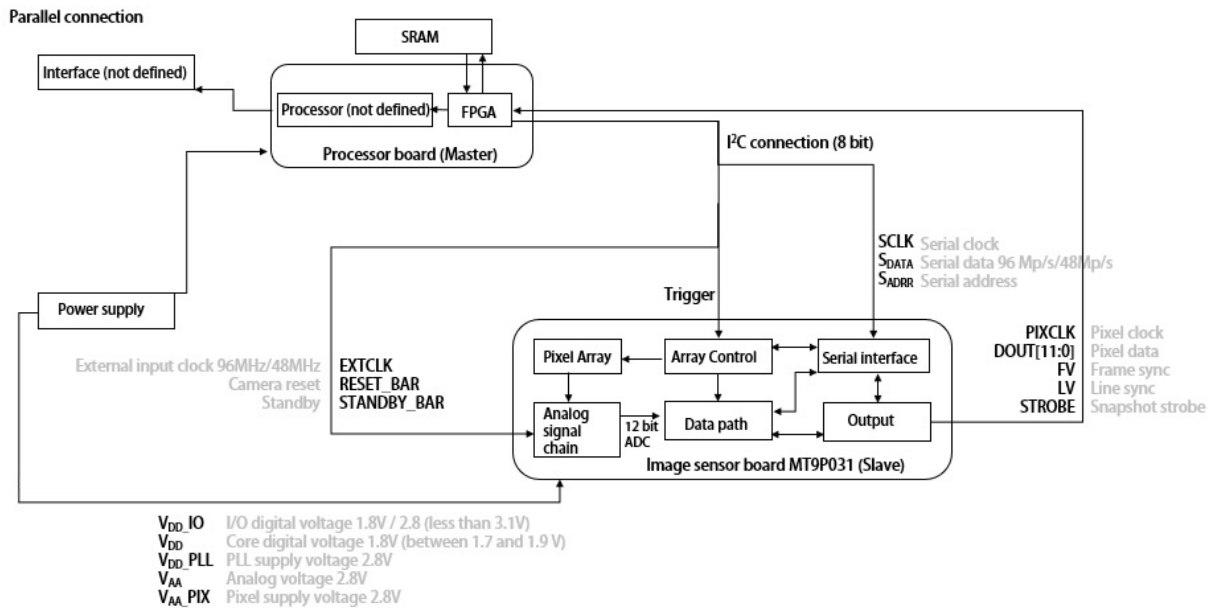


Figure 8.3: Electronic system proposed to be implemented in the future with a promising image sensor, MT9P031.

Bibliography

[1] W. J. Jorden. Soviet fires earth satellite into space. *The New York Times*, 5th October 1957. Consulted on 20th August 2018 online here.

[2] W. Atherton. Miniaturization of eletronics. In *From Compass to Computer*, pages 237–267. Palgrave London, 1984. ISBN 978-0-333-35268-7.

[3] E. Kulu. Nanosatellite & cubesat database. Available online here.

[4] H. Helvajian and S. Janson. *Small Satellites: Past, Present, and Future*. Aerospace Press, 2009. ISBN 978-1-884989-22-3.

[5] NASA CubeSat Launch Initiative. Cubesat 101: Basic concepts and processes for first-time cubesat developers. Technical report, NASA, October 2017. Available online here.

[6] Ancillary Description Writer's Guide. Global change master directory: Ancillary description writer's guide. Available online here, 2013.

[7] C. Henry. Portuguese company embarks on first domestic satellite project. *Space News*, November 10th 2017. Available online here.

[8] The CubeSat Program. Cubesat design specification rev 13. Technical report, Cal Poly, April 2015. Available online here.

[9] B. Yost. Chapter 6: Structures, materials and mechanisms. In *State of the Art of Small Spacecraft Technology*. NASA, 2018. Available online here.

[10] European Space Agency. 3d printing cubesat bodies for cheaper, faster missions. Available online here, May 2017.

[11] B. Yost. Chapter 3: Power. In *State of the Art of Small Spacecraft Technology*. NASA, 2018. Available online here.

[12] B. Yost. Chapter 7: Thermal control. In *State of the Art of Small Spacecraft Technology*. NASA, 2018. Available online here.

[13] B. Yost. Chapter 9: Communications. In *State of the Art of Small Spacecraft Technology*. NASA, 2018. Available online here.

[14] P. Fortescue, G. Swinerd, and J. Stark. *Spacecraft Systems Engineering*. John Wiley and Sons Ltd, 4th edition edition, 2011. ISBN: 978-0-470-75012-4.

[15] F. L. Markley and J. L. Crassidis. *Fundamentals of Spacecraft Attitude Determination and Control*. Springer and Microcosm Press, 2014. ISBN 978-1-4939-0802-8.

[16] C. C. Liebe. Star trackers for attitude determination. *IEEE Aerospace and Electronic Systems Magazine*, 1995. DOI: 10.1109/62.387971.

[17] M. J. Sidi. *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Cambridge University Press, 1997. ISBN: 0-521-55072-6.

[18] B. Schultz, B. Tapley, and G. Born. *Statistical Orbit Determination*. Academic Press, 2004. ISBN 978-0-12683-630-1.

[19] P. D. Groves. Chapter 2: Coordinate frames, kinematics, and the earth. In *Principles of GNSS, Inertial and Multisensor Integrated Navigation Systems*, pages 23–78. Antech House, 2013. ISBN 9978-1-60807-005-3.

[20] J. S. Drilling. Normal stars. In A. N. Cox, editor, *Allen's astrophyisical quantities*. NASA, 4th edition edition, 2002. ISBN: 978-1-4612-1186-0.

[21] W. Romanishin. *An introduction to Astronomical Photometry using CCDs*. University of Oklahoma, 2006.

[22] M. Moser. A simple way to determine charge conversion efficiency for a cmos sensor. Available online here, 2012.

[23] X. Qian. *Wide Dynamic Range CMOS Image Sensor for Star Tracking Applications*. PhD thesis, Nanyang Technological University, 2015.

[24] D. Doyle. Radiation hardness of optical materials, 2010. Available online here.

[25] National Aeronautics and Space Administration (NASA). What is space radiation? Available online here, 2016.

[26] National Aeronautics and Space Administration (NASA). Understanding space radiation. Available online here, 2002.

[27] E. M. Silverman. *Space Environmental Effects on Spacecraft: LEO materials selection guide*. National Aeronautics and Space Administration (NASA), 1995. Available online here.

[28] D. Sinclair. Radiation effects and cots parts in smallsats. *Small Satellite Conference*, 2013. Available online here.

[29] K. Avery, J. Fenchel, J. Mee, W. Kemp, R. Netzer, D. Elkins, B. Zufelt, and D. Alexander. Total dose test results for cubesat electronics. Available online here, 2011.

[30] M. B. B. M. Tom Segert, Steven Engelen. Development of the pico star tracker st-200 - design challenges and road ahead. Berlin Space Technologies, 2011.

[31] Observational Astronomy. Ccd characteristics lab. Available online here, 2007.

[32] N. Shterev. Design and testing of the star trackers for the seam nanosatellite. Master's thesis, Lulea University of Technology, 2015.

[33] K. S. Julian Brown. Tetra: Star identification with hash tables. Available online here, 2017.

[34] B. Wie. *Space Vehicle Dynamics and Control*. American Institute of Aeronautics and Astronautics inc, 2008. ISBN 978-1-56347-953-3.

[35] J. R. Wertz and W. J. Larson. *Space Mission Analysis and Design*. Microcosm Press and Space Technology Library, 3rd edition edition, 1999. ISBN 1-881883-10-8.

Appendix A

Register addresses and default values for MT9J001 and MT9J003

The following pages contain all register addresses and default values for the MT9J001 image sensor. These pages were taken from the MT9J001/3 Register Reference from Aptina.

Register List and Default Values

SMIA Configuration Register List

Table 1: SMIA Configuration
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R0 (R0x0000)	model_id	dddd dddd dddd dddd	11265 (0x2C01)
R2 (R0x0002)	revision_number	dddd dddd	32 (0x20)
R3 (R0x0003)	manufacturer_id	???? ????	6 (0x06)
R4 (R0x0004)	smia_version	???? ????	10 (0x0A)
R5 (R0x0005)	frame_count	???? ????	255 (0xFF)
R6 (R0x0006)	pixel_order	0000 00??	0 (0x00)
R8 (R0x0008)	data_pedestal	0000 dddd dddd dddd	40 (0x0028)
R64 (R0x0040)	frame_format_model_type	???? ????	1 (0x01)
R65 (R0x0041)	frame_format_model_subtype	???? ????	18 (0x12)
R66 (R0x0042)	frame_format_descriptor_0	???? ???? ???? ????	24144 (0x5E50)
R68 (R0x0044)	frame_format_descriptor_1	???? ???? ???? ????	4098 (0x1002)
R70 (R0x0046)	frame_format_descriptor_2	???? ???? ???? ????	23228 (0x5ABC)
R72 (R0x0048)	frame_format_descriptor_3	???? ???? ???? ????	0 (0x0000)
R74 (R0x004A)	frame_format_descriptor_4	???? ???? ???? ????	0 (0x0000)
R76 (R0x004C)	frame_format_descriptor_5	???? ???? ???? ????	0 (0x0000)
R78 (R0x004E)	frame_format_descriptor_6	???? ???? ???? ????	0 (0x0000)
R80 (R0x0050)	frame_format_descriptor_7	???? ???? ???? ????	0 (0x0000)
R82 (R0x0052)	frame_format_descriptor_8	???? ???? ???? ????	0 (0x0000)
R84 (R0x0054)	frame_format_descriptor_9	???? ???? ???? ????	0 (0x0000)
R86 (R0x0056)	frame_format_descriptor_10	???? ???? ???? ????	0 (0x0000)
R88 (R0x0058)	frame_format_descriptor_11	???? ???? ???? ????	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Aptina Confidential and Proprietary

MT9J003: Registers
Register List and Default Values

Table 1: SMIA Configuration (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R90 (R0x005A)	frame_format_descriptor_12	???? ???? ???? ????	0 (0x0000)
R92 (R0x005C)	frame_format_descriptor_13	???? ???? ???? ????	0 (0x0000)
R94 (R0x005E)	frame_format_descriptor_14	???? ???? ???? ????	0 (0x0000)
R128 (R0x0080)	analogue_gain_capability	???? ???? ???? ????	1 (0x0001)
R132 (R0x0084)	analogue_gain_code_min	???? ???? ???? ????	8 (0x0008)
R134 (R0x0086)	analogue_gain_code_max	???? ???? ???? ????	255 (0x00FF)
R136 (R0x0088)	analogue_gain_code_step	???? ???? ???? ????	1 (0x0001)
R138 (R0x008A)	analogue_gain_type	???? ???? ???? ????	0 (0x0000)
R140 (R0x008C)	analogue_gain_m0	???? ???? ???? ????	1 (0x0001)
R142 (R0x008E)	analogue_gain_c0	???? ???? ???? ????	0 (0x0000)
R144 (R0x0090)	analogue_gain_m1	???? ???? ???? ????	0 (0x0000)
R146 (R0x0092)	analogue_gain_c1	???? ???? ???? ????	8 (0x0008)
R192 (R0x00C0)	data_format_model_type	???? ????	1 (0x01)
R193 (R0x00C1)	data_format_model_subtype	???? ????	5 (0x05)
R194 (R0x00C2)	data_format_descriptor_0	???? ???? ???? ????	2570 (0x0A0A)
R196 (R0x00C4)	data_format_descriptor_1	???? ???? ???? ????	2056 (0x0808)
R198 (R0x00C6)	data_format_descriptor_2	???? ???? ???? ????	2568 (0x0A08)
R200 (R0x00C8)	data_format_descriptor_3	???? ???? ???? ????	3084 (0x0C0C)
R202 (R0x00CA)	data_format_descriptor_4	???? ???? ???? ????	3080 (0x0C08)
R204 (R0x00CC)	data_format_descriptor_5	???? ???? ???? ????	0 (0x0000)
R206 (R0x00CE)	data_format_descriptor_6	???? ???? ???? ????	0 (0x0000)
R256 (R0x0100)	mode_select	0000 000d	0 (0x00)
R257 (R0x0101)	image_orientation	0000 00dd	0 (0x00)
R259 (R0x0103)	software_reset	0000 000d	0 (0x00)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 1: SMIA Configuration (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R260 (R0x0104)	grouped_parameter_hold	0000 000d	0 (0x00)
R261 (R0x0105)	mask_corrupted_frames	0000 000d	0 (0x00)
R272 (R0x0110)	ccp2_channel_identifier	0000 0ddd	0 (0x00)
R273 (R0x0111)	ccp2_signalling_mode	0000 000d	1 (0x01)
R274 (R0x0112)	ccp_data_format	dddd dddd dddd dddd	3084 (0x0C0C)
R288 (R0x0120)	gain_mode	0000 000d	0 (0x00)
R512 (R0x0200)	fine_integration_time	dddd dddd dddd ddd0	1010 (0x03F2)
R514 (R0x0202)	coarse_integration_time	dddd dddd dddd dddd	16 (0x0010)
R516 (R0x0204)	analogue_gain_code_global	0000 0000 dddd dddd	8 (0x0008)
R518 (R0x0206)	analogue_gain_code_greenr	0000 0000 dddd dddd	8 (0x0008)
R520 (R0x0208)	analogue_gain_code_red	0000 0000 dddd dddd	8 (0x0008)
R522 (R0x020A)	analogue_gain_code_blue	0000 0000 dddd dddd	8 (0x0008)
R524 (R0x020C)	analogue_gain_code_greenb	0000 0000 dddd dddd	8 (0x0008)
R526 (R0x020E)	digital_gain_greenr	0000 0ddd 0000 0000	256 (0x0100)
R528 (R0x0210)	digital_gain_red	0000 0ddd 0000 0000	256 (0x0100)
R530 (R0x0212)	digital_gain_blue	0000 0ddd 0000 0000	256 (0x0100)
R532 (R0x0214)	digital_gain_greenb	0000 0ddd 0000 0000	256 (0x0100)
R768 (R0x0300)	vt_pix_clk_div	0000 0000 0000 dddd	3 (0x0003)
R770 (R0x0302)	vt_sys_clk_div	0000 0000 000d dddd	1 (0x0001)
R772 (R0x0304)	pre_pll_clk_div	0000 0000 00dd dddd	2 (0x0002)
R774 (R0x0306)	pll_multiplier	0000 0000 dddd dddd	48 (0x0030)
R776 (R0x0308)	op_pix_clk_div	0000 0000 000d dddd	12 (0x000C)
R778 (R0x030A)	op_sys_clk_div	0000 0000 000d dddd	1 (0x0001)
R832 (R0x0340)	frame_length_lines	dddd dddd dddd dddd	2891 (0x0B4B)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Aptina Confidential and Proprietary

MT9J003: Registers
Register List and Default Values

Table 1: SMIA Configuration (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R834 (R0x0342)	line_length_pck	dddd dddd dddd ddd0	7440 (0x1D10)
R836 (R0x0344)	x_addr_start	0000 dddd dddd dddd	112 (0x0070)
R838 (R0x0346)	y_addr_start	0000 dddd dddd dddd	8 (0x0008)
R840 (R0x0348)	x_addr_end	0000 dddd dddd dddd	3775 (0x0EBF)
R842 (R0x034A)	y_addr_end	0000 dddd dddd dddd	2755 (0x0AC3)
R844 (R0x034C)	x_output_size	0000 dddd dddd ddd0	3664 (0x0E50)
R846 (R0x034E)	y_output_size	0000 dddd dddd ddd0	2748 (0x0ABC)
R896 (R0x0380)	x_even_inc	0000 0000 0000 000?	1 (0x0001)
R898 (R0x0382)	x_odd_inc	0000 0000 0000 dddd	1 (0x0001)
R900 (R0x0384)	y_even_inc	0000 0000 0000 000?	1 (0x0001)
R902 (R0x0386)	y_odd_inc	0000 0000 00dd dddd	1 (0x0001)
R1024 (R0x0400)	scaling_mode	0000 0000 0000 00dd	0 (0x0000)
R1026 (R0x0402)	spatial_sampling	0000 0000 0000 000d	0 (0x0000)
R1028 (R0x0404)	scale_m	0000 0000 dddd dddd	16 (0x0010)
R1030 (R0x0406)	scale_n	0000 0000 ???? ????	16 (0x0010)
R1280 (R0x0500)	compression_mode	0000 0000 0000 000?	1 (0x0001)
R1536 (R0x0600)	test_pattern_mode	0000 000d 0000 0ddd	0 (0x0000)
R1538 (R0x0602)	test_data_red	0000 dddd dddd dddd	0 (0x0000)
R1540 (R0x0604)	test_data_greenr	0000 dddd dddd dddd	0 (0x0000)
R1542 (R0x0606)	test_data_blue	0000 dddd dddd dddd	0 (0x0000)
R1544 (R0x0608)	test_data_greenb	0000 dddd dddd dddd	0 (0x0000)
R1546 (R0x060A)	horizontal_cursor_width	0000 dddd dddd dddd	0 (0x0000)
R1548 (R0x060C)	horizontal_cursor_position	0000 dddd dddd dddd	0 (0x0000)
R1550 (R0x060E)	vertical_cursor_width	0000 dddd dddd dddd	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 1: SMIA Configuration (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R1552 (R0x0610)	vertical_cursor_position	0000 dddd dddd dddd	0 (0x0000)

SMIA Parameter Limits Register List

Table 2: SMIA Parameter Limits
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R4096 (R0x1000)	integration_time_capability	0000 0000 0000 000?	1 (0x0001)
R4100 (R0x1004)	coarse_integration_time_min	dddd dddd dddd dddd	0 (0x0000)
R4102 (R0x1006)	coarse_integration_time_max_margin	dddd dddd dddd dddd	1 (0x0001)
R4104 (R0x1008)	fine_integration_time_min	dddd dddd dddd dddd	1010 (0x03F2)
R4106 (R0x100A)	fine_integration_time_max_margin	dddd dddd dddd dddd	638 (0x027E)
R4224 (R0x1080)	digital_gain_capability	0000 0000 0000 000?	1 (0x0001)
R4228 (R0x1084)	digital_gain_min	???? ???? ???? ????	256 (0x0100)
R4230 (R0x1086)	digital_gain_max	???? ???? ???? ????	1792 (0x0700)
R4232 (R0x1088)	digital_gain_step_size	???? ???? ???? ????	256 (0x0100)
R4352 (R0x1100)	min_ext_clk_freq_mhz_1	???? ???? ???? ????	16384 (0x4000)
R4354 (R0x1102)	min_ext_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4356 (R0x1104)	max_ext_clk_freq_mhz_1	???? ???? ???? ????	17024 (0x4280)
R4358 (R0x1106)	max_ext_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4360 (R0x1108)	min_pre_pll_clk_div	???? ???? ???? ????	1 (0x0001)
R4362 (R0x110A)	max_pre_pll_clk_div	???? ???? ???? ????	64 (0x0040)
R4364 (R0x110C)	min_pll_ip_freq_mhz_1	???? ???? ???? ????	16512 (0x4080)
R4366 (R0x110E)	min_pll_ip_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4368 (R0x1110)	max_pll_ip_freq_mhz_1	???? ???? ???? ????	16832 (0x41C0)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 2: SMIA Parameter Limits (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R4370 (R0x1112)	max_pll_ip_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4372 (R0x1114)	min_pll_multiplier	???? ???? ???? ????	32 (0x0020)
R4374 (R0x1116)	max_pll_multiplier	???? ???? ???? ????	128 (0x0080)
R4376 (R0x1118)	min_pll_op_freq_mhz_1	???? ???? ???? ????	17344 (0x43C0)
R4378 (R0x111A)	min_pll_op_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4380 (R0x111C)	max_pll_op_freq_mhz_1	???? ???? ???? ????	17472 (0x4440)
R4382 (R0x111E)	max_pll_op_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4384 (R0x1120)	min_vt_sys_clk_div	???? ???? ???? ????	1 (0x0001)
R4386 (R0x1122)	max_vt_sys_clk_div	???? ???? ???? ????	8 (0x0008)
R4388 (R0x1124)	min_vt_sys_clk_freq_mhz_1	???? ???? ???? ????	16832 (0x41C0)
R4390 (R0x1126)	min_vt_sys_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4392 (R0x1128)	max_vt_sys_clk_freq_mhz_1	???? ???? ???? ????	17472 (0x4440)
R4394 (R0x112A)	max_vt_sys_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4396 (R0x112C)	min_vt_pix_clk_freq_mhz_1	???? ???? ???? ????	16409 (0x4019)
R4398 (R0x112E)	min_vt_pix_clk_freq_mhz_2	???? ???? ???? ????	39322 (0x999A)
R4400 (R0x1130)	max_vt_pix_clk_freq_mhz_1	???? ???? ???? ????	17088 (0x42C0)
R4402 (R0x1132)	max_vt_pix_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4404 (R0x1134)	min_vt_pix_clk_div	???? ???? ???? ????	2 (0x0002)
R4406 (R0x1136)	max_vt_pix_clk_div	???? ???? ???? ????	8 (0x0008)
R4416 (R0x1140)	min_frame_length_lines	dddd dddd dddd dddd	81 (0x0051)
R4418 (R0x1142)	max_frame_length_lines	dddd dddd dddd dddd	65535 (0xFFFF)
R4420 (R0x1144)	min_line_length_pck	dddd dddd dddd dddd	1648 (0x0670)
R4422 (R0x1146)	max_line_length_pck	dddd dddd dddd dddd	65534 (0xFFFE)
R4424 (R0x1148)	min_line_blanking_pck	dddd dddd dddd dddd	1134 (0x046E)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 2: SMIA Parameter Limits (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R4426 (R0x114A)	min_frame_blanking_lines	dddd dddd dddd dddd	79 (0x004F)
R4448 (R0x1160)	min_op_sys_clk_div	???? ???? ???? ????	1 (0x0001)
R4450 (R0x1162)	max_op_sys_clk_div	???? ???? ???? ????	1 (0x0001)
R4452 (R0x1164)	min_op_sys_clk_freq_mhz_1	???? ???? ???? ????	16793 (0x4199)
R4454 (R0x1166)	min_op_sys_clk_freq_mhz_2	???? ???? ???? ????	39322 (0x999A)
R4456 (R0x1168)	max_op_sys_clk_freq_mhz_1	???? ???? ???? ????	17472 (0x4440)
R4458 (R0x116A)	max_op_sys_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4460 (R0x116C)	min_op_pix_clk_div	???? ???? ???? ????	8 (0x0008)
R4462 (R0x116E)	max_op_pix_clk_div	???? ???? ???? ????	12 (0x000C)
R4464 (R0x1170)	min_op_pix_clk_freq_mhz_1	???? ???? ???? ????	16409 (0x4019)
R4466 (R0x1172)	min_op_pix_clk_freq_mhz_2	???? ???? ???? ????	39322 (0x999A)
R4468 (R0x1174)	max_op_pix_clk_freq_mhz_1	???? ???? ???? ????	17088 (0x42C0)
R4470 (R0x1176)	max_op_pix_clk_freq_mhz_2	???? ???? ???? ????	0 (0x0000)
R4480 (R0x1180)	x_addr_min	???? ???? ???? ????	24 (0x0018)
R4482 (R0x1182)	y_addr_min	???? ???? ???? ????	0 (0x0000)
R4484 (R0x1184)	x_addr_max	???? ???? ???? ????	3879 (0x0F27)
R4486 (R0x1186)	y_addr_max	???? ???? ???? ????	2763 (0x0ACB)
R4544 (R0x11C0)	min_even_inc	???? ???? ???? ????	1 (0x0001)
R4546 (R0x11C2)	max_even_inc	???? ???? ???? ????	1 (0x0001)
R4548 (R0x11C4)	min_odd_inc	???? ???? ???? ????	1 (0x0001)
R4550 (R0x11C6)	max_odd_inc	???? ???? ???? ????	63 (0x003F)
R4608 (R0x1200)	scaling_capability	0000 0000 0000 00??	2 (0x0002)
R4612 (R0x1204)	scaler_m_min	???? ???? ???? ????	16 (0x0010)
R4614 (R0x1206)	scaler_m_max	???? ???? ???? ????	128 (0x0080)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Aptina Confidential and Proprietary

Table 2: SMIA Parameter Limits (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R4616 (R0x1208)	scaler_n_min	???? ???? ???? ????	16 (0x0010)
R4618 (R0x120A)	scaler_n_max	???? ???? ???? ????	16 (0x0010)
R4864 (R0x1300)	compression_capability	0000 0000 0000 000?	1 (0x0001)
R5120 (R0x1400)	matrix_element_redinred	dddd dddd dddd dddd	578 (0x0242)
R5122 (R0x1402)	matrix_element_greeninred	dddd dddd dddd dddd	65280 (0xFF00)
R5124 (R0x1404)	matrix_element_blueinred	dddd dddd dddd dddd	65470 (0xFFBE)
R5126 (R0x1406)	matrix_element_redingreen	dddd dddd dddd dddd	65460 (0xFFB4)
R5128 (R0x1408)	matrix_element_greeningreen	dddd dddd dddd dddd	512 (0x0200)
R5130 (R0x140A)	matrix_element_blueingreen	dddd dddd dddd dddd	65357 (0xFF4D)
R5132 (R0x140C)	matrix_element_redinblue	dddd dddd dddd dddd	65521 (0xFFF1)
R5134 (R0x140E)	matrix_element_greeninblue	dddd dddd dddd dddd	65332 (0xFF34)
R5136 (R0x1410)	matrix_element_blueinblue	dddd dddd dddd dddd	476 (0x01DC)

Manufacturer-Specific Register List

Table 3: Manufacturer-Specific
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R12288 (R0x3000)	model_id_	dddd dddd dddd dddd	11265 (0x2C01)
R12290 (R0x3002)	y_addr_start_	0000 dddd dddd dddd	8 (0x0008)
R12292 (R0x3004)	x_addr_start_	0000 dddd dddd dddd	112 (0x0070)
R12294 (R0x3006)	y_addr_end_	0000 dddd dddd dddd	2755 (0x0AC3)
R12296 (R0x3008)	x_addr_end_	0000 dddd dddd dddd	3775 (0x0EBF)
R12298 (R0x300A)	frame_length_lines_	dddd dddd dddd dddd	2891 (0x0B4B)
R12300 (R0x300C)	line_length_pck_	dddd dddd dddd ddd0	7440 (0x1D10)
R12304 (R0x3010)	fine_correction	0ddd dddd dddd dddd	156 (0x009C)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R12306 (R0x3012)	coarse_integration_time_	dddd dddd dddd dddd	16 (0x0010)
R12308 (R0x3014)	fine_integration_time_	dddd dddd dddd ddd0	1010 (0x03F2)
R12310 (R0x3016)	row_speed	0000 0ddd 0ddd 0ddd	289 (0x0121)
R12312 (R0x3018)	extra_delay	dddd dddd dddd ddd0	0 (0x0000)
R12314 (R0x301A)	reset_register	dd0d 0ddd dddd dddd	24 (0x0018)
R12316 (R0x301C)	mode_select_	0000 000d	0 (0x00)
R12317 (R0x301D)	image_orientation_	0000 00dd	0 (0x00)
R12318 (R0x301E)	data_pedestal_	0000 dddd dddd dddd	40 (0x0028)
R12321 (R0x3021)	software_reset_	0000 000d	0 (0x00)
R12322 (R0x3022)	grouped_parameter_hold_	0000 000d	0 (0x00)
R12323 (R0x3023)	mask_corrupted_frames_	0000 000d	0 (0x00)
R12324 (R0x3024)	pixel_order_	0000 00??	0 (0x00)
R12326 (R0x3026)	gpi_status	dddd dddd dddd ????	65535 (0xFFFF)
R12328 (R0x3028)	analogue_gain_code_global_	0000 0000 dddd dddd	8 (0x0008)
R12330 (R0x302A)	analogue_gain_code_greenr_	0000 0000 dddd dddd	8 (0x0008)
R12332 (R0x302C)	analogue_gain_code_red_	0000 0000 dddd dddd	8 (0x0008)
R12334 (R0x302E)	analogue_gain_code_blue_	0000 0000 dddd dddd	8 (0x0008)
R12336 (R0x3030)	analogue_gain_code_greenb_	0000 0000 dddd dddd	8 (0x0008)
R12338 (R0x3032)	digital_gain_greenr_	0000 0ddd 0000 0000	256 (0x0100)
R12340 (R0x3034)	digital_gain_red_	0000 0ddd 0000 0000	256 (0x0100)
R12342 (R0x3036)	digital_gain_blue_	0000 0ddd 0000 0000	256 (0x0100)
R12344 (R0x3038)	digital_gain_greenb_	0000 0ddd 0000 0000	256 (0x0100)
R12346 (R0x303A)	smia_version_	???? ????	10 (0x0A)
R12347 (R0x303B)	frame_count_	???? ????	255 (0xFF)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R12348 (R0x303C)	frame_status	0000 0000 0000 00??	0 (0x0000)
R12352 (R0x3040)	read_mode	dddd dddd dddd dddd	65 (0x0041)
R12358 (R0x3046)	flash	??dd dddd dddd 0000	1536 (0x0600)
R12360 (R0x3048)	flash_count	dddd dddd dddd dddd	8 (0x0008)
R12374 (R0x3056)	green1_gain	0ddd dddd dddd dddd	4160 (0x1040)
R12376 (R0x3058)	blue_gain	0ddd dddd dddd dddd	4160 (0x1040)
R12378 (R0x305A)	red_gain	0ddd dddd dddd dddd	4160 (0x1040)
R12380 (R0x305C)	green2_gain	0ddd dddd dddd dddd	4160 (0x1040)
R12382 (R0x305E)	global_gain	0ddd dddd dddd dddd	4160 (0x1040)
R12394 (R0x306A)	datapath_status	0000 0000 00?d dddd	0 (0x0000)
R12398 (R0x306E)	datapath_select	dddd dddd ?ddd dddd	36992 (0x9080)
R12400 (R0x3070)	test_pattern_mode_	0000 000d 0000 0ddd	0 (0x0000)
R12402 (R0x3072)	test_data_red_	0000 dddd dddd dddd	0 (0x0000)
R12404 (R0x3074)	test_data_greenr_	0000 dddd dddd dddd	0 (0x0000)
R12406 (R0x3076)	test_data_blue_	0000 dddd dddd dddd	0 (0x0000)
R12408 (R0x3078)	test_data_greenb_	0000 dddd dddd dddd	0 (0x0000)
R12410 (R0x307A)	test_raw_mode	0000 0000 0000 00dd	0 (0x0000)
R12448 (R0x30A0)	x_even_inc_	0000 0000 0000 000?	1 (0x0001)
R12450 (R0x30A2)	x_odd_inc_	0000 0000 0000 dddd	1 (0x0001)
R12452 (R0x30A4)	y_even_inc_	0000 0000 0000 000?	1 (0x0001)
R12454 (R0x30A6)	y_odd_inc_	0000 0000 00dd dddd	1 (0x0001)
R12638 (R0x315E)	global_seq_trigger	dddd 0d?? 00dd 0ddd	0 (0x0000)
R12640 (R0x3160)	global_rst_end	dddd dddd dddd dddd	152 (0x0098)
R12642 (R0x3162)	global_shutter_start	dddd dddd dddd dddd	168 (0x00A8)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R12644 (R0x3164)	global_shutter_start2	dddd dddd dddd dddd	0 (0x0000)
R12646 (R0x3166)	global_read_start	dddd dddd dddd dddd	184 (0x00B8)
R12648 (R0x3168)	global_read_start2	dddd dddd dddd dddd	0 (0x0000)
R12704 (R0x31A0)	serial_format_descriptor_0	???? ???? ???? ????	257 (0x0101)
R12706 (R0x31A2)	serial_format_descriptor_1	???? ???? ???? ????	513 (0x0201)
R12708 (R0x31A4)	serial_format_descriptor_2	???? ???? ???? ????	514 (0x0202)
R12710 (R0x31A6)	serial_format_descriptor_3	???? ???? ???? ????	769 (0x0301)
R12712 (R0x31A8)	serial_format_descriptor_4	???? ???? ???? ????	770 (0x0302)
R12714 (R0x31AA)	serial_format_descriptor_5	???? ???? ???? ????	772 (0x0304)
R12716 (R0x31AC)	serial_format_descriptor_6	???? ???? ???? ????	0 (0x0000)
R12720 (R0x31B0)	frame_preamble	0000 0000 dddd dddd	99 (0x0063)
R12722 (R0x31B2)	line_preamble	0000 0000 dddd dddd	57 (0x0039)
R12724 (R0x31B4)	mipi_timing_0	???? dddd dddd dddd	3415 (0x0D57)
R12726 (R0x31B6)	mipi_timing_1	??dd dddd ??dd dddd	2832 (0x0B10)
R12728 (R0x31B8)	mipi_timing_2	??dd dddd ??dd dddd	269 (0x010D)
R12730 (R0x31BA)	mipi_timing_3	??dd dddd ?ddd dddd	1293 (0x050D)
R12732 (R0x31BC)	mipi_timing_4	???? ???? ?ddd dddd	11 (0x000B)
R12736 (R0x31C0)	hispi_timing	0ddd dddd dddd dddd	0 (0x0000)
R12742 (R0x31C6)	hispi_control_status	dddd dddd dddd dddd	32768 (0x8000)
R12776 (R0x31E8)	horizontal_cursor_position_	0000 dddd dddd dddd	0 (0x0000)
R12778 (R0x31EA)	vertical_cursor_position_	0000 dddd dddd dddd	0 (0x0000)
R12780 (R0x31EC)	horizontal_cursor_width_	0000 dddd dddd dddd	0 (0x0000)
R12782 (R0x31EE)	vertical_cursor_width_	0000 dddd dddd dddd	0 (0x0000)
R12786 (R0x31F2)	i2c_ids_mipi_default	dddd dddd dddd dddd	28268 (0x6E6C)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R12796 (R0x31FC)	i2c_ids	dddd dddd dddd dddd	12320 (0x3020)
R13824 (R0x3600)	p_gr_p0q0	dddd dddd dddd dddd	0 (0x0000)
R13826 (R0x3602)	p_gr_p0q1	dddd dddd dddd dddd	0 (0x0000)
R13828 (R0x3604)	p_gr_p0q2	dddd dddd dddd dddd	0 (0x0000)
R13830 (R0x3606)	p_gr_p0q3	dddd dddd dddd dddd	0 (0x0000)
R13832 (R0x3608)	p_gr_p0q4	dddd dddd dddd dddd	0 (0x0000)
R13834 (R0x360A)	p_rd_p0q0	dddd dddd dddd dddd	0 (0x0000)
R13836 (R0x360C)	p_rd_p0q1	dddd dddd dddd dddd	0 (0x0000)
R13838 (R0x360E)	p_rd_p0q2	dddd dddd dddd dddd	0 (0x0000)
R13840 (R0x3610)	p_rd_p0q3	dddd dddd dddd dddd	0 (0x0000)
R13842 (R0x3612)	p_rd_p0q4	dddd dddd dddd dddd	0 (0x0000)
R13844 (R0x3614)	p_bl_p0q0	dddd dddd dddd dddd	0 (0x0000)
R13846 (R0x3616)	p_bl_p0q1	dddd dddd dddd dddd	0 (0x0000)
R13848 (R0x3618)	p_bl_p0q2	dddd dddd dddd dddd	0 (0x0000)
R13850 (R0x361A)	p_bl_p0q3	dddd dddd dddd dddd	0 (0x0000)
R13852 (R0x361C)	p_bl_p0q4	dddd dddd dddd dddd	0 (0x0000)
R13854 (R0x361E)	p_gb_p0q0	dddd dddd dddd dddd	0 (0x0000)
R13856 (R0x3620)	p_gb_p0q1	dddd dddd dddd dddd	0 (0x0000)
R13858 (R0x3622)	p_gb_p0q2	dddd dddd dddd dddd	0 (0x0000)
R13860 (R0x3624)	p_gb_p0q3	dddd dddd dddd dddd	0 (0x0000)
R13862 (R0x3626)	p_gb_p0q4	dddd dddd dddd dddd	0 (0x0000)
R13888 (R0x3640)	p_gr_p1q0	dddd dddd dddd dddd	0 (0x0000)
R13890 (R0x3642)	p_gr_p1q1	dddd dddd dddd dddd	0 (0x0000)
R13892 (R0x3644)	p_gr_p1q2	dddd dddd dddd dddd	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R13894 (R0x3646)	p_gr_p1q3	dddd dddd dddd dddd	0 (0x0000)
R13896 (R0x3648)	p_gr_p1q4	dddd dddd dddd dddd	0 (0x0000)
R13898 (R0x364A)	p_rd_p1q0	dddd dddd dddd dddd	0 (0x0000)
R13900 (R0x364C)	p_rd_p1q1	dddd dddd dddd dddd	0 (0x0000)
R13902 (R0x364E)	p_rd_p1q2	dddd dddd dddd dddd	0 (0x0000)
R13904 (R0x3650)	p_rd_p1q3	dddd dddd dddd dddd	0 (0x0000)
R13906 (R0x3652)	p_rd_p1q4	dddd dddd dddd dddd	0 (0x0000)
R13908 (R0x3654)	p_bl_p1q0	dddd dddd dddd dddd	0 (0x0000)
R13910 (R0x3656)	p_bl_p1q1	dddd dddd dddd dddd	0 (0x0000)
R13912 (R0x3658)	p_bl_p1q2	dddd dddd dddd dddd	0 (0x0000)
R13914 (R0x365A)	p_bl_p1q3	dddd dddd dddd dddd	0 (0x0000)
R13916 (R0x365C)	p_bl_p1q4	dddd dddd dddd dddd	0 (0x0000)
R13918 (R0x365E)	p_gb_p1q0	dddd dddd dddd dddd	0 (0x0000)
R13920 (R0x3660)	p_gb_p1q1	dddd dddd dddd dddd	0 (0x0000)
R13922 (R0x3662)	p_gb_p1q2	dddd dddd dddd dddd	0 (0x0000)
R13924 (R0x3664)	p_gb_p1q3	dddd dddd dddd dddd	0 (0x0000)
R13926 (R0x3666)	p_gb_p1q4	dddd dddd dddd dddd	0 (0x0000)
R13952 (R0x3680)	p_gr_p2q0	dddd dddd dddd dddd	0 (0x0000)
R13954 (R0x3682)	p_gr_p2q1	dddd dddd dddd dddd	0 (0x0000)
R13956 (R0x3684)	p_gr_p2q2	dddd dddd dddd dddd	0 (0x0000)
R13958 (R0x3686)	p_gr_p2q3	dddd dddd dddd dddd	0 (0x0000)
R13960 (R0x3688)	p_gr_p2q4	dddd dddd dddd dddd	0 (0x0000)
R13962 (R0x368A)	p_rd_p2q0	dddd dddd dddd dddd	0 (0x0000)
R13964 (R0x368C)	p_rd_p2q1	dddd dddd dddd dddd	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R13966 (R0x368E)	p_rd_p2q2	dddd dddd dddd dddd	0 (0x0000)
R13968 (R0x3690)	p_rd_p2q3	dddd dddd dddd dddd	0 (0x0000)
R13970 (R0x3692)	p_rd_p2q4	dddd dddd dddd dddd	0 (0x0000)
R13972 (R0x3694)	p_bl_p2q0	dddd dddd dddd dddd	0 (0x0000)
R13974 (R0x3696)	p_bl_p2q1	dddd dddd dddd dddd	0 (0x0000)
R13976 (R0x3698)	p_bl_p2q2	dddd dddd dddd dddd	0 (0x0000)
R13978 (R0x369A)	p_bl_p2q3	dddd dddd dddd dddd	0 (0x0000)
R13980 (R0x369C)	p_bl_p2q4	dddd dddd dddd dddd	0 (0x0000)
R13982 (R0x369E)	p_gb_p2q0	dddd dddd dddd dddd	0 (0x0000)
R13984 (R0x36A0)	p_gb_p2q1	dddd dddd dddd dddd	0 (0x0000)
R13986 (R0x36A2)	p_gb_p2q2	dddd dddd dddd dddd	0 (0x0000)
R13988 (R0x36A4)	p_gb_p2q3	dddd dddd dddd dddd	0 (0x0000)
R13990 (R0x36A6)	p_gb_p2q4	dddd dddd dddd dddd	0 (0x0000)
R14016 (R0x36C0)	p_gr_p3q0	dddd dddd dddd dddd	0 (0x0000)
R14018 (R0x36C2)	p_gr_p3q1	dddd dddd dddd dddd	0 (0x0000)
R14020 (R0x36C4)	p_gr_p3q2	dddd dddd dddd dddd	0 (0x0000)
R14022 (R0x36C6)	p_gr_p3q3	dddd dddd dddd dddd	0 (0x0000)
R14024 (R0x36C8)	p_gr_p3q4	dddd dddd dddd dddd	0 (0x0000)
R14026 (R0x36CA)	p_rd_p3q0	dddd dddd dddd dddd	0 (0x0000)
R14028 (R0x36CC)	p_rd_p3q1	dddd dddd dddd dddd	0 (0x0000)
R14030 (R0x36CE)	p_rd_p3q2	dddd dddd dddd dddd	0 (0x0000)
R14032 (R0x36D0)	p_rd_p3q3	dddd dddd dddd dddd	0 (0x0000)
R14034 (R0x36D2)	p_rd_p3q4	dddd dddd dddd dddd	0 (0x0000)
R14036 (R0x36D4)	p_bl_p3q0	dddd dddd dddd dddd	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R14038 (R0x36D6)	p_bl_p3q1	dddd dddd dddd dddd	0 (0x0000)
R14040 (R0x36D8)	p_bl_p3q2	dddd dddd dddd dddd	0 (0x0000)
R14042 (R0x36DA)	p_bl_p3q3	dddd dddd dddd dddd	0 (0x0000)
R14044 (R0x36DC)	p_bl_p3q4	dddd dddd dddd dddd	0 (0x0000)
R14046 (R0x36DE)	p_gb_p3q0	dddd dddd dddd dddd	0 (0x0000)
R14048 (R0x36E0)	p_gb_p3q1	dddd dddd dddd dddd	0 (0x0000)
R14050 (R0x36E2)	p_gb_p3q2	dddd dddd dddd dddd	0 (0x0000)
R14052 (R0x36E4)	p_gb_p3q3	dddd dddd dddd dddd	0 (0x0000)
R14054 (R0x36E6)	p_gb_p3q4	dddd dddd dddd dddd	0 (0x0000)
R14080 (R0x3700)	p_gr_p4q0	dddd dddd dddd dddd	0 (0x0000)
R14082 (R0x3702)	p_gr_p4q1	dddd dddd dddd dddd	0 (0x0000)
R14084 (R0x3704)	p_gr_p4q2	dddd dddd dddd dddd	0 (0x0000)
R14086 (R0x3706)	p_gr_p4q3	dddd dddd dddd dddd	0 (0x0000)
R14088 (R0x3708)	p_gr_p4q4	dddd dddd dddd dddd	0 (0x0000)
R14090 (R0x370A)	p_rd_p4q0	dddd dddd dddd dddd	0 (0x0000)
R14092 (R0x370C)	p_rd_p4q1	dddd dddd dddd dddd	0 (0x0000)
R14094 (R0x370E)	p_rd_p4q2	dddd dddd dddd dddd	0 (0x0000)
R14096 (R0x3710)	p_rd_p4q3	dddd dddd dddd dddd	0 (0x0000)
R14098 (R0x3712)	p_rd_p4q4	dddd dddd dddd dddd	0 (0x0000)
R14100 (R0x3714)	p_bl_p4q0	dddd dddd dddd dddd	0 (0x0000)
R14102 (R0x3716)	p_bl_p4q1	dddd dddd dddd dddd	0 (0x0000)
R14104 (R0x3718)	p_bl_p4q2	dddd dddd dddd dddd	0 (0x0000)
R14106 (R0x371A)	p_bl_p4q3	dddd dddd dddd dddd	0 (0x0000)
R14108 (R0x371C)	p_bl_p4q4	dddd dddd dddd dddd	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Table 3: Manufacturer-Specific (continued)
1 = read-only, always 1; 0 = read-only, always 0; d = programmable;? = read-only, dynamic

Register Dec(Hex)	Name	Data Format (Binary)	Default Value Dec(Hex)
R14110 (R0x371E)	p_gb_p4q0	dddd dddd dddd dddd	0 (0x0000)
R14112 (R0x3720)	p_gb_p4q1	dddd dddd dddd dddd	0 (0x0000)
R14114 (R0x3722)	p_gb_p4q2	dddd dddd dddd dddd	0 (0x0000)
R14116 (R0x3724)	p_gb_p4q3	dddd dddd dddd dddd	0 (0x0000)
R14118 (R0x3726)	p_gb_p4q4	dddd dddd dddd dddd	0 (0x0000)
R14208 (R0x3780)	poly_sc_enable	d000 0000 0000 0000	0 (0x0000)
R14210 (R0x3782)	poly_origin_c	0000 dddd dddd dddd	0 (0x0000)
R14212 (R0x3784)	poly_origin_r	0000 dddd dddd dddd	0 (0x0000)

Aptina reserves the right to change products or specifications without notice.
©2010 Aptina Imaging Corporation. All rights reserved.

Appendix B

Algorithm codes

B.1 C code to filter information from Hipparcos catalog

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define _USE_MATH_DEFINES
#define STARN 118218

//This file only needs to be compiled once
//Commands to compile this code:
//"gcc-6 -Wall hip.c -o hip.x" and then "./hip.x"

//Function to extract the parameters needed for tetra from the file hip_main.dat
void getParameters(int item_number, char string [], FILE *ofp){

  char *str;
  str = string;
  const char s[78] = "|";
  char *token[450] = {NULL};
  int i = 0;

  int XNO; //Hipparcos number
  long double SRA0; //Right Ascension in degrees
  long double SDEC0; //Declination in degrees
  char *IS; //Spectral type;
  float MAG; //Magnitude (in Johnson)
  long double XRPM; //RA proper motion in millisecond of arc
  long double XDPM; //DEC proper motion in millisecond of arc

  //Get the first token
  token[0] = strtok(str, s);

  //Walk through other tokens
  i=1;
  while( i>0 && i<78 ) {
```

```
39      token[i] = strtok(NULL, s);
40      i++;
41    }
42
43    XNO = atoi(token [1]);
44    SRA0 = atof(token [8])*M_PI/180; //Right Ascension in radians
45    SDEC0 = atof(token [9])*M_PI/180; //Right Ascension in radians
46    IS = token [76];
47    MAG = atof(token [5]);
48    XRPM = atof(token [12])*0.001*M_PI/648000; //RA proper motion in radians
49    XDPM = atof(token [13])*0.001*M_PI/648000; //DEC proper motion in radians
50
51    //3 - Writing star parameters in a new file
52    fprintf(ofp, "%d %.13Lf %.13LF %f %.13LF %.13LF\n", XNO, SRA0, SDEC0, MAG, XRPM,
      XDPM);
53
54  }
55
56  //Star of the main function
57  enum { MAX_LINES = (STARN+1), MAX_COLUMNS = 450 };
58  int main (int argc, char **argv) {
59
60    //Open hip_main.dat file
61    char (*lines)[MAX_COLUMNS] = NULL;
62    int n = 0;
63    FILE *hipparcos_catalog = fopen ("hip_main.dat", "r");
64
65    /*Valdiate file open for reading */
66    if (!hipparcos_catalog) {
67      fprintf (stderr, "Error: File was not oppened. \n");
68      return 1;
69    }
70
71    /* Allocate MAX_LINES arrays */
72    if (!(lines = malloc (MAX_LINES * sizeof *lines))) {
73      fprintf (stderr, "Error: Virtual memory exhausted lines. \n");
74      return 1;
75    }
76
77    //Cicle to read each line of the file
78    while (n < MAX_LINES && fgets (lines[n], MAX_COLUMNS, hipparcos_catalog)) {
79      char *p = lines[n];
80
81      //Cicle to find first '\n'
82      for (; *p && *p != '\n'; p++) {}
83
84      //Analyse line char by char
85      if (*p != '\n') {
86        int c;
87        while ((c = fgetc (hipparcos_catalog)) != '\n' && c != EOF) { }
88      }
89      *p = 0, n++;
90    }
91
92    //Close file
```

```
  if (hipparcos_catalog != stdin) fclose (hipparcos_catalog);

  //Create new file to write
  FILE *hip;
  hip = fopen("hip.txt", "wb");

  //Write in the new file the filtered information about the stars
  int counter = 0;
  while (counter < 118322){
    if (counter >= MAX_LINES-1){
     fprintf(hip, "%d %.13Lf %.13LF %f %.13LF %.13LF\n", 0, 0.0000000, 0.00000000,
    0.0000000, 0.0000000, 0.0000000000000);
    }
    else getParameters( counter, lines[counter], hip);
    counter ++;
  }

  //Close file and free allocated memory
  fclose(hip);
  free (lines);

  return 0;
}
```

B.2 Tetra algorithm with alterations from this work

```
"""
Copyright (c) 2016 Julian Brown

Permission is hereby granted, free of charge, to any person obtaining a copy of this
    software and associated documentation files (the "Software"), to deal in the
    Software without restriction, including without limitation the rights to use,
    copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
    Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
    INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
import numpy as np
import itertools
from PIL import Image
import scipy.ndimage
import scipy.optimize
import scipy.stats
import glob

# directory containing input images
```

```
19 image_directory = './pics'
20
21 # boolean for whether or not to display an annotated version
22 # of the image with identified stars circled in green and
23 # unmatched catalog stars circled in red
24 show_solution = True
25
26 # maximum fields of view of catalog patterns in degrees
27 # determines approximately what image fields of view
28 # can be identified by the algorithm
29 max_fovs = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60]
30
31 # radius around image stars to search for matching catalog stars
32 # as a fraction of image field of view in x dimension
33 match_radius = .01
34
35 # maximum error in edge ratios
36 max_error = .01
37
38 # image downsampling factor for median filtering
39 downsample_factor = 2
40
41 # median filter window radius in down-sampled image pixels
42 filter_radius = 2
43 filter_width = filter_radius * 2 + 1
44
45 # percentage of pattern catalog that stores values
46 catalog_fill_factor = .5
47
48 # number of divisions along each dimension of the pattern catalog
49 num_catalog_bins = 25
50
51 # maximum number of pattern catalog stars within
52 # the maximum fov centered on any given pattern catalog star
53 max_stars_per_fov = 10
54
55 # minimum star brightness magnitude
56 magnitude_minimum = 5.5
57
58 # minimum allowable angle between star vectors in radians
59 # used to remove double stars
60 min_angle = .002
61
62 # number of stars to use in each pattern
63 pattern_size = 5
64
65 # minimum number of pixels in a group of bright pixels
66 # needed to classify the group as a star
67 min_pixels_in_group = 3
68
69 # centroiding window radius around a star's center pixel
70 # does not count the center pixel
71 window_radius = 2
72
73 # maximum number of bright stars to check against pattern catalog
```

```
74  max_pattern_checking_stars = 8
75
76  # maximum probability of mismatch for verifying an attitude determination
77  max_mismatch_probability = 1e-20
78
79  # percentage of fine sky map that stores values
80  fine_sky_map_fill_factor = .5
81
82  # number of divisions to break a single radius of
83  # the celestial sphere into for rapid star lookup
84  num_fine_sky_map_bins = 100
85
86  # percentage of course sky map that stores values
87  course_sky_map_fill_factor = .5
88
89  # number of divisions to break a single radius of
90  # the celestial sphere into for rapid star lookup
91  num_course_sky_map_bins = 4
92
93  # constant used for randomizing hash functions
94  avalanche_constant = 2654435761
95
96  # converts a hash_code into an index in the hash table
97  def hash_code_to_index(hash_code, bins_per_dimension, hash_table_size):
98    # convert hashcode to python integers
99    hash_code = [int(value) for value in hash_code]
100   # represent hash code as a single integer
101   integer_hash_code = sum(hash_code[i] * bins_per_dimension ** i for i in range(len(
      hash_code)))
102   # randomize the hash code by multiplying by the avalanching constant
103   # take the result modulo the table size to give a random index
104   index = (integer_hash_code * avalanche_constant) % hash_table_size
105   return index
106
107 # find all stars within a radius centered on the given vector using the compressed
      course sky map
108 def get_nearby_stars_compressed_course(vector, radius):
109   # create list of nearby stars
110   nearby_star_ids = []
111   # given error of at most radius in each dimension, compute the space of hash codes
       to lookup in the sky map
112   hash_code_space = [range(max(low,0), min(high+1,2*num_course_sky_map_bins)) for (
      low, high) in zip(((vector + 1 - radius) * num_course_sky_map_bins).astype(np.
      int),
113
                          ((vector + 1 + radius) * num_course_sky_map_bins).astype(np.int
      ))]
114   # iterate over hash code space, looking up partitions of the sky map that are
      within range of the given vector
115   for hash_code in itertools.product(*hash_code_space):
116     hash_index = hash_code_to_index(hash_code, 2*num_course_sky_map_bins,
      compressed_course_sky_map_hash_table_size)
117     # iterate over the star lists in the given partition, adding them to
118     # the nearby stars list if they're in the correct bin and within range of the
      vector
```

```
119     for index in ((2 * (hash_index + offset ** 2)) %
    compressed_course_sky_map_hash_table_size for offset in itertools.count()):
120       # if the current slot is empty, the bin does not exist
121       if not compressed_course_sky_map[index]:
122         break
123       # otherwise, check if the indices correspond to the correct bin
124       indices = compressed_course_sky_map[index:index+2]
125       # extract the sublist of star ids given by the indices
126       star_id_list = compressed_course_sky_map[slice(*indices)]
127       # check if the hash code for the first star matches the bin's hash code
128       first_star_vector = star_table[star_id_list[0]]
129       first_star_hash_code = tuple(((first_star_vector+1)*num_course_sky_map_bins).
    astype(np.int))
130       if first_star_hash_code == hash_code:
131         # iterate over the stars in the sublist, adding them to
132         # the nearby stars list if they're within range of the vector
133         for star_id in star_id_list:
134           if np.dot(vector, star_table[star_id]) > np.cos(radius):
135             nearby_star_ids.append(star_id)
136   return nearby_star_ids
137
138 # open the pattern catalog and fine sky map and test whether they are fully
139 # generated with the following parameters and if not, regenerate them
140 parameters = (max_fovs,
141               num_catalog_bins,
142               max_stars_per_fov,
143               magnitude_minimum,
144               min_angle,
145               pattern_size,
146               fine_sky_map_fill_factor,
147               num_fine_sky_map_bins,
148               course_sky_map_fill_factor,
149               num_course_sky_map_bins)
150 # try opening the database files
151 try:
152   pattern_catalog = np.load('pattern_catalog.npy')
153   fine_sky_map = np.load('fine_sky_map.npy')
154   compressed_course_sky_map = np.load('compressed_course_sky_map.npy')
155   compressed_course_sky_map_hash_table_size = compressed_course_sky_map[-1]
156   star_table = np.load('star_table.npy')
157   stored_parameters = open('params.txt', 'r').read()
158   # if it got this far, the reads didn't fail
159   read_failed = 0
160 except:
161   # loading from the files failed, so they either didn't existing or weren't
    complete
162   read_failed = 1
163 # there are two cases in which the catalog needs to be regenerated:
164 # reading the stored files failed or the stored parameters
165 # are different than those specified above
166 if read_failed or str(parameters) != stored_parameters:
167   #------------------------------------------------------------------------------
168   # Original code with BSC5 catalog:
169
170   # number of stars in BSC5 catalog
```

```
# STARN = 9110
# BSC5 data storage format
# bsc5_data_type = [("XNO", np.float32),
#                   ("SRA0", np.float64),
#                   ("SDEC0", np.float64),
#                   ("IS", np.int16),
#                   ("MAG", np.int16),
#                   ("XRPM", np.float32),
#                   ("XDPM", np.float32)
#                   ]

# open BSC5 catalog file for reading
# bsc5_file = open('BSC5', 'rb')
# skip first 28 header bytes
# bsc5_file.seek(28)
# read BSC5 catalog into array
# bsc5 = np.fromfile(bsc5_file, dtype=bsc5_data_type, count=STARN)

# year to calculate catalog for
# should be relatively close to 1950
# year = 2016


# retrieve star positions, magnitudes and ids from BSC5 catalog
# stars = []
# for star_num in range(STARN):
  # only use stars brighter (i.e. lower magnitude)
  # than the minimum allowable magnitude
  # mag = bsc5[star_num][4] / 100.0
  # if mag <= magnitude_minimum:
    # retrieve RA in 1950
    # ra = bsc5[star_num][1]
    # correct RA to modern day
    # ra += bsc5[star_num][5] * (year - 1950)
    # retrieve DEC in 1950
    # dec = bsc5[star_num][2]
    # correct DEC to modern day
    # dec += bsc5[star_num][6] * (year - 1950)
    # skip blank star entries
    # if ra == 0.0 and dec == 0.0:
      # continue
    # convert RA, DEC to (x,y,z)
    # vector = np.array([np.cos(ra)*np.cos(dec), np.sin(ra)*np.cos(dec), np.sin(
  dec)])
    # retrieve star ID number in BSC5
    # star_id = int(bsc5[star_num][0])
    # add vector, magnitude pair to list of stars
    # stars.append((vector, mag, star_id))
#----------------------------------------------------------------------
# Migration to Hipparcos:
#----------------------------------------------------------------------
# number of lines in Hipparcos catalog
STARN = 118322

with open('hip.txt', 'rb') as f:
```

```
225     bsc5 = ([line.split() for line in f])
226 
227   # year to calculate catalog for
228   # Using epoch J1991.25
229   year = 2018
230 
231   # retrieve star positions, magnitudes and ids from BSC5 catalog
232   stars = []
233   for star_num in range(STARN):
234     # only use stars brighter (i.e. lower magnitude)
235     # than the minimum allowable magnitude
236     mag = float(bsc5[star_num][3])
237     if mag <= magnitude_minimum:
238       # retrieve RA in 1950
239       ra = float(bsc5[star_num][1])
240       # correct RA to modern day
241       ra += float(bsc5[star_num][4]) * (year - 1991.25)
242       # retrieve DEC in 1950
243       dec = float(bsc5[star_num][2])
244       # correct DEC to modern day
245       dec += float(bsc5[star_num][5]) * (year - 1991.25)
246       # skip blank star entries
247       if ra == 0.0 and dec == 0.0:
248         continue
249       # convert RA, DEC to (x,y,z)
250       vector = np.array([np.cos(ra)*np.cos(dec), np.sin(ra)*np.cos(dec), np.sin(dec)
    ])
251       # retrieve star ID number in BSC5
252       star_id = int(bsc5[star_num][0])
253       #star_id = star_num
254       # add vector, magnitude pair to list of stars
255       stars.append((vector, mag, star_id))
256   #------------------------------------------------------------------------------
257 
258   # fast method for removing double stars using sort
259   # sort list by x component of star vectors
260   stars.sort(key=lambda star: star[0][0])
261   # create boolean indicator list for which star vectors are double stars
262   doubles = [0] * len(stars)
263   for star_num1 in range(len(stars)):
264     for star_num2 in range(star_num1 + 1, len(stars)):
265       # skip checking star pairs with x component differences which
266       # are already larger than the minimum allowable angle
267       if stars[star_num2][0][0] - stars[star_num1][0][0] >= min_angle:
268         break
269       # if the star pair forms a double star, add both to the indicator list
270       if np.dot(stars[star_num1][0], stars[star_num2][0]) > np.cos(min_angle):
271         doubles[star_num1] = 1
272         doubles[star_num2] = 1
273         break
274   # use the double star indicator list to create a new star vectors list without
    double stars
275   stars_no_doubles = [stars[i] for i in range(len(stars)) if not doubles[i]]
276   # output how many stars will be stored in the star table and the fine and course
    sky maps
```

```
277 print("number of stars in star table and sky maps: " + str(len(stars)))
278
279 # add all non-double stars brighter than magnitude_minimum to the star table and
    sky maps
280 star_table = np.zeros((STARN+1, 3), dtype=np.float32)
281 # create fine sky map hash table, which maps vectors to star ids
282
283 # In this line original code was changed: fine_sky_map = np.zeros(len(stars) /
    fine_sky_map_fill_factor, dtype=np.uint16)
284 fine_sky_map = np.zeros(int(len(stars) / fine_sky_map_fill_factor), dtype=np.
    uint32)
285 # create course sky map hash table, which maps vectors to star ids
286 course_sky_map = {}
287 for (vector, mag, star_id) in stars_no_doubles:
288   # add star vector to star table in position corresponding to its id
289   star_table[star_id] = vector
290   # find which partition the star occupies in the fine sky map hash table
291   hash_code = ((vector+1)*num_fine_sky_map_bins).astype(np.int)
292   hash_index = hash_code_to_index(hash_code, 2*num_fine_sky_map_bins, fine_sky_map
    .size)
293   # use quadratic probing to find an open space in the fine sky map to insert the
    star in
294   for index in ((hash_index + offset ** 2) % fine_sky_map.size for offset in
    itertools.count()):
295     # if the current slot is empty, add the star
296     # otherwise, move on to the next slot
297     if not fine_sky_map[index]:
298       fine_sky_map[index] = star_id
299       break
300   # find which partition the star occupies in the course sky map hash table
301   hash_code = tuple(((vector+1)*num_course_sky_map_bins).astype(np.int))
302   # if the partition is empty, create a new list to hold the star
303   # if the partition already contains stars, add the star to the list
304   course_sky_map[hash_code] = course_sky_map.pop(hash_code, []) + [star_id]
305
306 # create compressed version of course sky map by indirectly mapping vectors to
    star ids
307 # the map consists of a hash table, a superlist of stars, and a number
    representing the size of the hash table
308 # the hash table consists of pairs of indices which slice the superlist into the
    output star id lists
309 compressed_course_sky_map_hash_table_size = 2 * len(course_sky_map.keys()) /
    course_sky_map_fill_factor
310
311 # In this line original code was changed: compressed_course_sky_map = np.zeros(
    compressed_course_sky_map_hash_table_size + len(stars_no_doubles) + 1, dtype=np.
    uint16)
312 compressed_course_sky_map = np.zeros(int(compressed_course_sky_map_hash_table_size
     + len(stars_no_doubles) + 1), dtype=np.uint32)
313 compressed_course_sky_map[-1] = compressed_course_sky_map_hash_table_size
314 # add the items of the course sky map to the compressed course sky map one at a
    time
315 first_open_slot_in_superlist = compressed_course_sky_map_hash_table_size
316 for (hash_code, star_id_list) in course_sky_map.items():
317   # compute the indices for the slice of the superlist the star id list will
```

```
      occupy
318
319     # In this line original code was changed: slice_indices = (
      first_open_slot_in_superlist, first_open_slot_in_superlist + len(star_id_list))
320     slice_indices = (int(first_open_slot_in_superlist), int(
      first_open_slot_in_superlist + len(star_id_list)))
321     # add the star id list to the superlist
322     compressed_course_sky_map[slice(*slice_indices)] = star_id_list
323     # increment the counter for the first open slot in the superlist
324     first_open_slot_in_superlist += len(star_id_list)
325     hash_index = hash_code_to_index(hash_code, 2*num_course_sky_map_bins,
      compressed_course_sky_map_hash_table_size)
326     # use quadratic probing to find an open space in the hash table to insert the
      star in
327     for index in ((2 * (hash_index + offset ** 2)) %
      compressed_course_sky_map_hash_table_size for offset in itertools.count()):
328       # if the current slot is empty, add the slice indices to the hash table
329       # otherwise, move on to the next slot
330
331       # In this segment the original code was changed:
332       # if not compressed_course_sky_map[index]:
333       #   compressed_course_sky_map[index:(index+2)] = slice_indices
334       if not compressed_course_sky_map[int(index)]:
335         compressed_course_sky_map[int(index):int(index+2)] = slice_indices
336         break
337
338   # sort list by star magnitude, from brightest to dimmest
339   stars_no_doubles.sort(key=lambda star: star[1])
340
341
342   # find all stars within a radius centered on the given vector using the pruned
      course sky map
343   def get_nearby_stars_pruned_course(vector, radius):
344     # create list of nearby stars
345     nearby_star_ids = []
346     # given error of at most radius in each dimension, compute the space of hash
      codes to lookup in the sky map
347     hash_code_space = [range(max(low,0), min(high+1,2*num_course_sky_map_bins)) for
      (low, high) in zip(((vector + 1 - radius) * num_course_sky_map_bins).astype(np.
      int),
348
                              ((vector + 1 + radius) * num_course_sky_map_bins).astype(np.
      int))]
349     # iterate over hash code space, looking up partitions of the sky map that are
      within range of the given vector
350     for hash_code in itertools.product(*hash_code_space):
351       # iterate over the stars in the given partition, adding them to
352       # the nearby stars list if they're within range of the vector
353       for star_id in pruned_course_sky_map.get(hash_code, []):
354         if np.dot(vector, star_table[star_id]) > np.cos(radius):
355           nearby_star_ids.append(star_id)
356     return nearby_star_ids
357
358   # generate pattern catalog
359   print("generating catalog, this may take an hour...")
```

```
360  # create temporary list to store the patterns
361  pattern_list = np.zeros((100000000, pattern_size), dtype=np.uint32)
362  # create counter, which records how many patterns have been created
363  num_patterns_found = 0
364  # generate a piece of the catalog for each fov specified
365  for max_fov in max_fovs:
366    print("computing " + str(max_fov) + " degree fov patterns...")
367
368    # change field of view from degrees to radians
369    max_fov_rad = max_fov * np.pi / 180
370
371    # fast method for pruning high density areas of the sky
372    # create a hash table of the sky, which divides the
373    # unit cube around the celestial sphere
374    # up into (2*num_course_sky_map_bins)^3 partitions
375    pruned_course_sky_map = {}
376    # insert the stars into the hash table
377    for (vector, mag, star_id) in stars_no_doubles:
378      # skip stars that have too many closeby, brighter stars
379      if len(get_nearby_stars_pruned_course(vector, max_fov_rad / 2)) >=
     max_stars_per_fov:
380        continue
381      # find which partition the star occupies in the hash table
382      hash_code = tuple(((vector+1)*num_course_sky_map_bins).astype(np.int))
383      # if the partition is empty, create a new list to hold the star
384      # if the partition already contains stars, add the star to the list
385      pruned_course_sky_map[hash_code] = pruned_course_sky_map.pop(hash_code, []) +
     [star_id]
386    # create a list of stars without high density areas of the sky
387    star_ids_pruned = [star_id for sublist in pruned_course_sky_map.values() for
     star_id in sublist]
388
389    # initialize pattern, which will contain pattern_size star ids
390    pattern = [None] * pattern_size
391    for pattern[0] in star_ids_pruned:
392      # find which partition the star occupies in the sky hash table
393      hash_code = tuple(np.floor((star_table[pattern[0]]+1)*num_course_sky_map_bins)
     .astype(np.int))
394      # remove the star from the sky hash table
395      pruned_course_sky_map[hash_code].remove(pattern[0])
396      # iterate over all possible patterns containing the removed star
397      for pattern[1:] in itertools.combinations(get_nearby_stars_pruned_course(
     star_table[pattern[0]], max_fov_rad), pattern_size-1):
398        # retrieve the vectors of the stars in the pattern
399        vectors = [star_table[star_id] for star_id in pattern]
400        # verify that the pattern fits within the maximum field-of-view
401        # by checking the distances between every pair of stars in the pattern
402        if all(np.dot(*star_pair) > np.cos(max_fov_rad) for star_pair in itertools.
     combinations(vectors[1:], 2)):
403          pattern_list[num_patterns_found] = pattern
404          num_patterns_found += 1
405
406  # truncate pattern list to only contain valid values
407  pattern_list = pattern_list[:num_patterns_found]
408
```

```
409    # insert star patterns into pattern catalog hash table
410    print("inserting patterns into catalog...")
411    # In this line original code was changed: pattern_catalog = np.zeros((
         num_patterns_found / catalog_fill_factor, pattern_size), dtype=np.uint16)
412    pattern_catalog = np.zeros(( int(num_patterns_found / catalog_fill_factor),
         pattern_size), dtype=np.uint32)
413    for pattern in pattern_list:
414      # retrieve the vectors of the stars in the pattern
415      vectors = np.array([star_table[star_id] for star_id in pattern])
416      # calculate and sort the edges of the star pattern, which are the distances
         between its stars
417      edges = np.sort([np.linalg.norm(np.subtract(*star_pair)) for star_pair in
         itertools.combinations(vectors, 2)])
418      # extract the largest edge
419      largest_edge = edges[-1]
420      # divide the edges by the largest edge to create dimensionless ratios
421      edge_ratios = edges[:-1] / float(largest_edge)
422      # convert edge ratio float to hash code by binning
423      hash_code = tuple((edge_ratios * num_catalog_bins).astype(np.int))
424      hash_index = hash_code_to_index(hash_code, num_catalog_bins, pattern_catalog.
         shape[0])
425      # use quadratic probing to find an open space in the pattern catalog to insert
         the pattern in
426      for index in ((hash_index + offset ** 2) % pattern_catalog.shape[0] for offset
         in itertools.count()):
427        # if the current slot is empty, add the pattern
428        if not pattern_catalog[index][0]:
429          pattern_catalog[index] = pattern
430          break
431        # if the current slot contains a previously inserted
432        # copy of the same pattern, don't add the pattern
433        elif sorted(pattern_catalog[index]) == sorted(pattern):
434          break
435        # otherwise, continue the search by moving on to the next slot
436        else:
437          continue
438
439    # save star table, sky maps, pattern catalog, and parameters to disk
440    np.save('star_table.npy', star_table)
441    np.save('fine_sky_map.npy', fine_sky_map)
442    np.save('compressed_course_sky_map.npy', compressed_course_sky_map)
443    np.save('pattern_catalog.npy', pattern_catalog)
444    parameters = open('params.txt', 'w').write(str(parameters))
445
446  # run the tetra star tracking algorithm on the given image
447  def tetra(image_file_name):
448    # read image from file and convert to black and white
449    image = np.array(Image.open(image_file_name).convert('L'))
450    # extract height (y) and width (x) of image
451    height, width = image.shape
452
453    # flatten image by subtracting median filtered image
454    # clip image so size is a multiple of downsample_factor
455    # note that this may shift the center of the image
456    height = height - height % downsample_factor
```

```
457    width = width - width % downsample_factor
458    image = image[:height, :width]
459    # downsample image for median filtering
460    downsampled_image = image.reshape((height//downsample_factor,downsample_factor,
         width//downsample_factor,downsample_factor)).mean(axis=3).mean(axis=1)
461    # apply median filter to downsampled image
462    median_filtered_image = scipy.ndimage.filters.median_filter(downsampled_image,
         size=filter_width, output=image.dtype)
463    # upsample median filtered image back to original image size
464    upsampled_median_filtered_image = median_filtered_image.repeat(downsample_factor,
         axis=0).repeat(downsample_factor, axis=1)
465    # subtract the minimum of the image pixel and the local median to prevent values
         less than 0
466    normalized_image = image - np.minimum.reduce([upsampled_median_filtered_image,
         image])
467
468    # find all groups of pixels brighter than 5 sigma
469    bright_pixels = zip(*np.where(normalized_image > 5 * np.std(normalized_image)))
470    # group adjacent bright pixels together
471    # create a dictionary mapping pixels to their group
472    pixel_to_group = {}
473    # iterate over the pixels from upper left to lower right
474    for pixel in bright_pixels:
475      # check whether the pixels above or to the left are part of
476      # an existing group, which the current pixel will be added to
477      left_pixel = (pixel[0]   , pixel[1]-1)
478      up_pixel   = (pixel[0]-1, pixel[1]   )
479      in_left_group = left_pixel in pixel_to_group
480      in_up_group = up_pixel in pixel_to_group
481      # if both are part of existing, disjoint groups, add the current pixel and
         combine the groups
482      if in_left_group and in_up_group and id(pixel_to_group[left_pixel]) != id(
         pixel_to_group[up_pixel]):
483        # add the current pixel to the upper pixel's group
484        pixel_to_group[up_pixel].append(pixel)
485        # append the upper pixel group onto the left pixel group
486        pixel_to_group[left_pixel].extend(pixel_to_group[up_pixel])
487        # replace all of the upper pixel group's dictionary entries
488        # with references to the left pixel group
489        for up_group_pixel in pixel_to_group[up_pixel]:
490          pixel_to_group[up_group_pixel] = pixel_to_group[left_pixel]
491      # if exactly one of the left pixel or upper pixels is part of an existing group,
492      # add the current pixel to that group and add the current pixel to the
         dictionary
493      elif in_left_group:
494        pixel_to_group[left_pixel].append(pixel)
495        pixel_to_group[pixel] = pixel_to_group[left_pixel]
496      elif in_up_group:
497        pixel_to_group[up_pixel].append(pixel)
498        pixel_to_group[pixel] = pixel_to_group[up_pixel]
499      # if neither of the left pixel or upper pixel are in an existing group,
500      # add the current pixel to its own group and store it in the dictionary
501      else:
502        pixel_to_group[pixel] = [pixel]
503    # iterate over the dictionary to extract all of the unique groups
```

```
504   seen = set()
505   groups = [seen.add(id(group)) or group for group in pixel_to_group.values() if id(
        group) not in seen]
506
507   # find the brightest pixel for each group containing at least
508   # the minimum number of pixels required to be classified as a star
509   star_center_pixels = [max(group, key=lambda pixel: normalized_image[pixel]) for
        group in groups if len(group) > min_pixels_in_group]
510   # find the centroid, or center of mass, of each star
511   window_size = window_radius * 2 + 1
512   # pixel values are weighted by their distances from the left (x) and top (y) of
        the window
513   x_weights = np.fromfunction(lambda y,x:x+.5,(window_size, window_size))
514   y_weights = np.fromfunction(lambda y,x:y+.5,(window_size, window_size))
515   star_centroids = []
516   for (y,x) in star_center_pixels:
517     # throw out star if it's too close to the edge of the image
518     if y < window_radius or y >= height - window_radius or \
519        x < window_radius or x >= width  - window_radius:
520       continue
521     # extract the window around the star center from the image
522     star_window = normalized_image[y-window_radius:y+window_radius+1, x-
        window_radius:x+window_radius+1]
523     # find the total mass, or brightness, of the window
524     mass = np.sum(star_window)
525     # calculate the center of mass of the window in the x and y dimensions
        separately
526     x_center = np.sum(star_window * x_weights) / mass - window_radius
527     y_center = np.sum(star_window * y_weights) / mass - window_radius
528     # correct the star center position using the calculated center of mass to create
         a centroid
529     star_centroids.append((y + y_center, x + x_center))
530   # sort star centroids from brightest to dimmest by comparing the total masses of
        their window pixels
531
532     # In this line original code was changed: star_centroids.sort(key=lambda yx:-np.
        sum(normalized_image[yx[0]-window_radius:yx[0]+window_radius+1, yx[1]-
        window_radius:yx[1]+window_radius+1]))
533     star_centroids.sort(key=lambda yx:-np.sum(normalized_image[int(yx[0]-
        window_radius):int(yx[0]+window_radius+1), int(yx[1]-window_radius):int(yx[1]+
        window_radius+1)]))
534
535   # compute list of (i,j,k) vectors given list of (y,x) star centroids and
536   # an estimate of the image's field-of-view in the x dimension
537   # by applying the pinhole camera equations
538   def compute_vectors(star_centroids, fov):
539     center_x = width / 2.
540     center_y = height / 2.
541     fov_rad = fov * np.pi / 180
542     scale_factor = np.tan(fov_rad / 2) / center_x
543     star_vectors = []
544     for (star_y, star_x) in star_centroids:
545       j_over_i = (center_x - star_x) * scale_factor
546       k_over_i = (center_y - star_y) * scale_factor
547       i = 1. / np.sqrt(1 + j_over_i**2 + k_over_i**2)
```

```
548        j = j_over_i * i
549        k = k_over_i * i
550        star_vectors.append(np.array([i,j,k]))
551      return star_vectors
552
553    # generate star patterns in order of brightness
554    def centroid_pattern_generator(star_centroids, pattern_size):
555      # break if there aren't enough centroids to make even one pattern
556      if len(star_centroids) < pattern_size:
557        return
558      star_centroids = np.array(star_centroids)
559      # create a list of the pattern's centroid indices
560      # add the lower and upper index bounds as the first
561      # and last elements, respectively
562      pattern_indices = [-1] + list(range(pattern_size)) + [len(star_centroids)]
563      # output the very brightest centroids before doing anything else
564      yield star_centroids[pattern_indices[1:-1]]
565      # iterate until the very dimmest centroids have been output
566      # which occurs when the first pattern index has reached its maximum value
567      while pattern_indices[1] < len(star_centroids) - pattern_size:
568        # increment the pattern indices in order
569        for index_to_change in range(1, pattern_size + 1):
570          pattern_indices[index_to_change] += 1
571          # if the current set of pattern indices is valid, use them
572          if pattern_indices[index_to_change] < pattern_indices[index_to_change + 1]:
573            break
574          # otherwise, incrementing caused a conflict with the next pattern index
575          # resolve the conflict by resetting the current pattern index and moving on
576          else:
577            pattern_indices[index_to_change] = pattern_indices[index_to_change - 1] +
      1
578        # output the centroids corresponding to the current set of pattern indices
579        yield star_centroids[pattern_indices[1:-1]]
580
581    # iterate over every combination of size pattern_size of the brightest
      max_pattern_checking_stars stars in the image
582    for pattern_star_centroids in centroid_pattern_generator(star_centroids[:
      max_pattern_checking_stars], pattern_size):
583      # iterate over possible fields-of-view
584      for fov_estimate in max_fovs:
585        # compute star vectors using an estimate for the field-of-view in the x
      dimension
586        pattern_star_vectors = compute_vectors(pattern_star_centroids, fov_estimate)
587        # calculate and sort the edges of the star pattern, which are the Euclidean
      distances between its stars' vectors
588        pattern_edges = np.sort([np.linalg.norm(np.subtract(*star_pair)) for star_pair
       in itertools.combinations(pattern_star_vectors, 2)])
589        # extract the largest edge
590        pattern_largest_edge = pattern_edges[-1]
591        # divide the pattern's edges by the largest edge to create dimensionless
      ratios for lookup in the catalog
592        pattern_edge_ratios = pattern_edges[:-1] / pattern_largest_edge
593        # given error of at most max_error in the edge_ratios, compute the space of
      hash codes to lookup in the catalog
594        hash_code_space = [range(max(low,0), min(high+1,num_catalog_bins)) for (low,
```

```
    high) in zip(((pattern_edge_ratios - max_error) * num_catalog_bins).astype(np.int),
595                          ((pattern_edge_ratios + max_error) * num_catalog_bins).astype(np.int))]
596     # iterate over hash code space, only looking up non-duplicate codes that are in sorted order
597     for hash_code in set([tuple(sorted(code)) for code in itertools.product(*hash_code_space)]):
598         hash_code = tuple(hash_code)
599         hash_index = hash_code_to_index(hash_code, num_catalog_bins, pattern_catalog.shape[0])
600         # use quadratic probing to find all slots that patterns with the given hash code could appear in
601         for index in ((hash_index + offset ** 2) % pattern_catalog.shape[0] for offset in itertools.count()):
602             # if the current slot is empty, we've already
603             # seen all patterns that match the given hash code
604             if not pattern_catalog[index][0]:
605                 break
606             # retrieve the star ids of possible match from pattern catalog
607             catalog_pattern = pattern_catalog[index]
608             # retrieve the vectors of the stars in the catalog pattern
609             catalog_vectors = np.array([star_table[star_id] for star_id in catalog_pattern])
610             # find the centroid, or average position, of the star pattern
611             centroid = np.mean(catalog_vectors, axis=0)
612             # calculate each star's radius, or Euclidean distance from the centroid
613             radii = [np.linalg.norm(vector - centroid) for vector in catalog_vectors]
614             # use the radii to uniquely order the catalog vectors
615             catalog_sorted_vectors = catalog_vectors[np.argsort(radii)]
616             # calculate and sort the edges of the star pattern, which are the distances between its stars
617             catalog_edges = np.sort([np.linalg.norm(np.subtract(*star_pair)) for star_pair in itertools.combinations(catalog_vectors, 2)])
618             # extract the largest edge
619             catalog_largest_edge = catalog_edges[-1]
620             # divide the edges by the largest edge to create dimensionless ratios
621             catalog_edge_ratios = catalog_edges[:-1] / catalog_largest_edge
622             # verify star patterns match to within the given maximum allowable error
623             # note that this also filters out star patterns from colliding bins
624             if any([abs(val) > max_error for val in (catalog_edge_ratios - pattern_edge_ratios)]):
625                 continue
626             # compute the actual field-of-view using least squares optimization
627             # compute the catalog pattern's edges for error estimation
628             catalog_edges = np.append(catalog_edge_ratios * catalog_largest_edge, catalog_largest_edge)
629             # helper function that calculates a list of errors in pattern edge lengths
630             # with the catalog edge lengths for a given fov
631             def fov_to_error(fov):
632                 # recalculate the pattern's star vectors given the new fov
633                 pattern_star_vectors = compute_vectors(pattern_star_centroids, fov)
634                 # recalculate the pattern's edge lengths
635                 pattern_edges = np.sort([np.linalg.norm(np.subtract(*star_pair)) for
```

```
star_pair in itertools.combinations(pattern_star_vectors, 2)])
636             # return a list of errors, one for each edge
637             return catalog_edges - pattern_edges
638         # find the fov that minimizes the squared error, starting with the given
estimate
639         fov = scipy.optimize.leastsq(fov_to_error, fov_estimate)[0][0]
640         # convert newly computed fov to radians
641         fov_rad = fov * np.pi / 180
642         # find half diagonal fov of image in radians
643         fov_half_diagonal_rad = fov_rad * np.sqrt(width ** 2 + height ** 2) / (2 *
 width)
644         # recalculate star vectors using the new field-of-view
645         pattern_star_vectors = compute_vectors(pattern_star_centroids, fov)
646         # find the centroid, or average position, of the star pattern
647         pattern_centroid = np.mean(pattern_star_vectors, axis=0)
648         # calculate each star's radius, or Euclidean distance from the centroid
649         pattern_radii = [np.linalg.norm(star_vector - pattern_centroid) for
star_vector in pattern_star_vectors]
650         # use the radii to uniquely order the pattern's star vectors so they can
be matched with the catalog vectors
651         pattern_sorted_vectors = np.array(pattern_star_vectors)[np.argsort(
pattern_radii)]
652 
653         # calculate the least-squares rotation matrix from the catalog frame to
the image frame
654         def find_rotation_matrix(image_vectors, catalog_vectors):
655             # find the covariance matrix H between the image vectors and catalog
vectors
656             H = np.sum([np.dot(image_vectors[i].reshape((3,1)), catalog_vectors[i].
reshape((1,3))) for i in range(len(image_vectors))], axis=0)
657             # use singular value decomposition to find the rotation matrix
658             U, S, V = np.linalg.svd(H)
659             rotation_matrix = np.dot(U, V)
660             # correct reflection matrix if determinant is -1 instead of 1
661             # by flipping the sign of the third column of the rotation matrix
662             rotation_matrix[:,2] *= np.linalg.det(rotation_matrix)
663             return rotation_matrix
664 
665         # use the pattern match to find an estimate for the image's rotation
matrix
666         rotation_matrix = find_rotation_matrix(pattern_sorted_vectors,
catalog_sorted_vectors)
667         # calculate all star vectors using the new field-of-view
668         all_star_vectors = compute_vectors(star_centroids, fov)
669 
670         def find_matches(all_star_vectors, rotation_matrix):
671             # rotate each of the star vectors into the catalog frame by
672             # using the inverse (transpose) of the tentative rotation matrix
673             rotated_star_vectors = [np.dot(rotation_matrix.T, star_vector) for
star_vector in all_star_vectors]
674             # retrieve matching catalog vectors for each image vector
675             catalog_vectors = []
676             for rotated_star_vector in rotated_star_vectors:
677                 hash_code_space = [range(max(low,0), min(high+1,2*
num_fine_sky_map_bins)) for (low, high) in zip(((rotated_star_vector + 1 -
```

```
match_radius) * num_fine_sky_map_bins).astype(np.int),
678
                              ((rotated_star_vector + 1 + match_radius) *
num_fine_sky_map_bins).astype(np.int))]
679          # iterate over hash code space, only looking up non-duplicate codes
that are in sorted order
680          matching_stars = []
681          for hash_code in [code for code in itertools.product(*hash_code_space)
]:
682            hash_index = hash_code_to_index(hash_code, 2*num_fine_sky_map_bins,
fine_sky_map.size)
683            # use quadratic probing to find an open space in the fine sky map to
 insert the star in
684            for index in ((hash_index + offset ** 2) % fine_sky_map.size for
offset in itertools.count()):
685              # if the current slot is empty, all of the matching stars have
been found
686              # otherwise, move on to the next slot
687              if not fine_sky_map[index]:
688                break
689              # only accept stars within the match radius
690              elif np.dot(star_table[fine_sky_map[index]], rotated_star_vector)
> np.cos(match_radius * fov_rad):
691                matching_stars.append(star_table[fine_sky_map[index]])
692          catalog_vectors.append(matching_stars)
693        # stars must uniquely match a catalog star brighter than
magnitude_minimum
694        matches = [(image_vector, catalog_star[0]) for (image_vector,
catalog_star) in zip(all_star_vectors, catalog_vectors) if len(catalog_star) ==
1]
695        # catalog stars must uniquely match image stars
696        matches_hash = {}
697        # add the matches to the hash one at a time
698        for (image_vector, catalog_vector) in matches:
699          # exactly one image vector must match
700          if tuple(catalog_vector) in matches_hash:
701            matches_hash[tuple(catalog_vector)] = "multiple matches"
702          else:
703            matches_hash[tuple(catalog_vector)] = image_vector
704        # reverse order so that image vector is first in each pair
705        matches = []
706        for (catalog_vector, image_vector) in matches_hash.items():
707          # filter out catalog stars with multiple image star matches
708          # In this line origial code was changed: if image_vector == "multiple
matches":
709          if (type(image_vector) is str):
710            continue
711          matches.append((image_vector, np.array(catalog_vector)))
712        return matches
713
714
715      matches = find_matches(all_star_vectors, rotation_matrix)
716      # calculate loose upper bound on probability of mismatch assuming random
star distribution
717      # find number of catalog stars appear in a circumscribed circle around the
```

```
    image
718         image_center_vector = np.dot(rotation_matrix.T, np.array((1,0,0)))
719         num_nearby_catalog_stars = len(get_nearby_stars_compressed_course(
    image_center_vector, fov_half_diagonal_rad))
720         # calculate probability of a single random image centroid mismatching to a
     catalog star
721         single_star_mismatch_probability = 1 - num_nearby_catalog_stars *
    match_radius ** 2 * width / height
722         # apply binomial theorem to calculate probability upper bound on this many
     mismatches
723         # three of the matches account for the dimensions of freedom: position,
    rotation, and scale
724         mismatch_probability_upper_bound = scipy.stats.binom.cdf(len(
    star_centroids) - (len(matches) - 3), len(star_centroids) - 3,
    single_star_mismatch_probability)
725         # if a high probability match has been found, recompute the attitude using
     all matching stars
726         if mismatch_probability_upper_bound < max_mismatch_probability:
727           # diplay mismatch probability in scientific notation
728           print ("mismatch probability: %.4g" % mismatch_probability_upper_bound)
729           # recalculate the rotation matrix using the newly identified stars
730           rotation_matrix = find_rotation_matrix(*zip(*matches))
731           # recalculate matched stars given new rotation matrix
732           matches = find_matches(all_star_vectors, rotation_matrix)
733           # extract right ascension, declination, and roll from rotation matrix
    and convert to degrees
734           ra = (np.arctan2(rotation_matrix[0][1], rotation_matrix[0][0]) % (2 * np
    .pi)) * 180 / np.pi
735           dec = np.arctan2(rotation_matrix[0][2], np.sqrt(rotation_matrix[1][2]**2
     + rotation_matrix[2][2]**2)) * 180 / np.pi
736           roll = (np.arctan2(rotation_matrix[1][2], rotation_matrix[2][2]) % (2 *
    np.pi)) * 180 / np.pi
737           # print out attitude and field-of-view to 4 decimal places
738           print("RA:   %.4f" % ra)
739           print("DEC:  %.4f" % dec)
740           print("ROLL: %.4f" % roll)
741           print("FOV:  %.4f" % fov)
742           # display input image with green circles around matched catalog stars
743           # and red circles around unmatched catalog stars
744           if show_solution:
745             # draws circles around where the given vectors appear in an image
746             def draw_circles(image, vectors, color, circle_fidelity, circle_radius
    ):
747               # calculate the pixel position of the center of the image
748               image_center_x = width / 2.
749               image_center_y = height / 2.
750               # calculate conversion ratio between pixels and distance in the unit
     celestial sphere
751               scale_factor = image_center_x / np.tan(fov_rad / 2)
752               # iterate over the vectors, adding a circle for each one that
    appears in the image frame
753               for (i, j, k) in vectors:
754                 # find the center pixel for the vector's circle
755                 circle_center_x = np.floor(image_center_x - (j / i) * scale_factor
    )
```

```
756                   circle_center_y = np.floor(image_center_y - (k / i) * scale_factor
    )
757                   # draw a circle of the given color with the given fidelity
758                   for angle in np.array(range(circle_fidelity)) * 2 * np.pi /
    circle_fidelity:
759                     # find the x and y coordinates for the pixel that will be drawn
760                     pixel_x = int(circle_center_x + circle_radius * np.sin(angle))
761                     pixel_y = int(circle_center_y + circle_radius * np.cos(angle))
762                     # verify the pixel is within the image bounds
763                     if pixel_x < 0 or pixel_x >= width or pixel_y < 0 or pixel_y >=
    height:
764                       continue
765                     # draw the pixel
766                     image.putpixel((pixel_x, pixel_y), color)
767               # plot the image with green circles around matched stars
768               # and red circles around stars that weren't matched
769               rgb_image = Image.fromarray(image).convert('RGB')
770               # the circle is drawn using the corners of an n-gon, where the circle
    fidelity is n
771               circle_fidelity = 100
772               # star centroids that appear within the circle radius would match with
     the circle's corresponding catalog vector
773               circle_radius = match_radius * width + 1
774               # find which catalog stars could appear in the image
775               image_center_vector = np.dot(rotation_matrix.T, np.array((1,0,0)))
776               nearby_catalog_stars = get_nearby_stars_compressed_course(
    image_center_vector, fov_half_diagonal_rad)
777               # rotate the vectors of all of the nearby catalog stars into the image
     frame
778               rotated_nearby_catalog_vectors = [np.dot(rotation_matrix, star_table[
    star_id]) for star_id in nearby_catalog_stars]
779               # color all of the circles red by default
780               color_all = (255, 0, 0)
781               draw_circles(rgb_image, rotated_nearby_catalog_vectors, color_all,
    circle_fidelity, circle_radius)
782               # rotate the matched catalog stars into the image frame
783               matched_rotated_catalog_vectors = [np.dot(rotation_matrix,
    catalog_vector) for (image_vector, catalog_vector) in matches]
784               # recolor matched circles green
785               color_matched = (0, 255, 0)
786               draw_circles(rgb_image, matched_rotated_catalog_vectors, color_matched
    , circle_fidelity, circle_radius)
787               rgb_image.show()
788             return
789
790   # print failure message
791   print("failed to determine attitude")
792
793 for image_file_name in glob.glob(image_directory + '/*'):
794   print(image_file_name)
795   tetra(image_file_name)
```

Appendix C

Instruction manual for Raspberry Pi

The following pages contain the instructions manual regarding Raspberry Pi written to Omnidea, in order to leave them all documented.

Star Tracker Manual

How to build a Star Tracker with a Raspberry Pi by Susana Lopes

This document presents a tutorial to configure a Raspberry Pi to be used as a star tracker.

Hardware and software needed

- Raspberry Pi Model 3 B+
- Raspberry Pi 5.1V 2.5A Power Supply
- Memory card 16GB or superior (class 8 or 10 is a plus)
- Wi-Fi connection
- Ethernet cable (Not essential)
- Computer
- USB cable
- External mouse, keyboard and screen (these items provide an easier installation process, but they are not essential)
- Image sensor MT9J001 with Arducam Shield Module or similar

For your convenience, the software needed is provided in this document's directory. However, the hyperlinks to the downloads are also available in this document.

The Arducam files and TETRA algorithm files are available in this document's directory.

1 Configuring Raspberry Pi

These steps are only needed if your Raspberry Pi was just bought and/or if your SD card doesn't have an operating system installed. If Raspberry Pi is already configured and ready to be used, please skip to section 2: Configuring Computer Remote Access.

In order to use Raspberry Pi, an operating system needs to be installed on the SD card. The recommended operating system to be installed is Raspbian, the official Raspberry Pi operating system.

1. **Operating System** - First of all, you need to download Raspbian Stretch with desktop (download here). If you are going to use the program recommended in the next step to write Raspian Stretch image, you wont need to unzip the file.
2. **Writing image to the SD card** - Insert the SD card in the computer slot using an SD card adapter and download Etcher (download here). Open Etcher. In 'Open image', select the Raspbian zip file previously downloaded. Select your SD card in 'Select drive' and press 'Flash!'. The image will start to be written in the SD card (**Be careful**: after pressing start, all the previous files in the SD card will be deleted).

If you have an external keyboard and a monitor, connect them to your Raspberry Pi and follow **A** -Installing Raspbian with an external keyboard and screen. Otherwise, skip to **B** - Installing Raspbian without an external keyboard and screen.

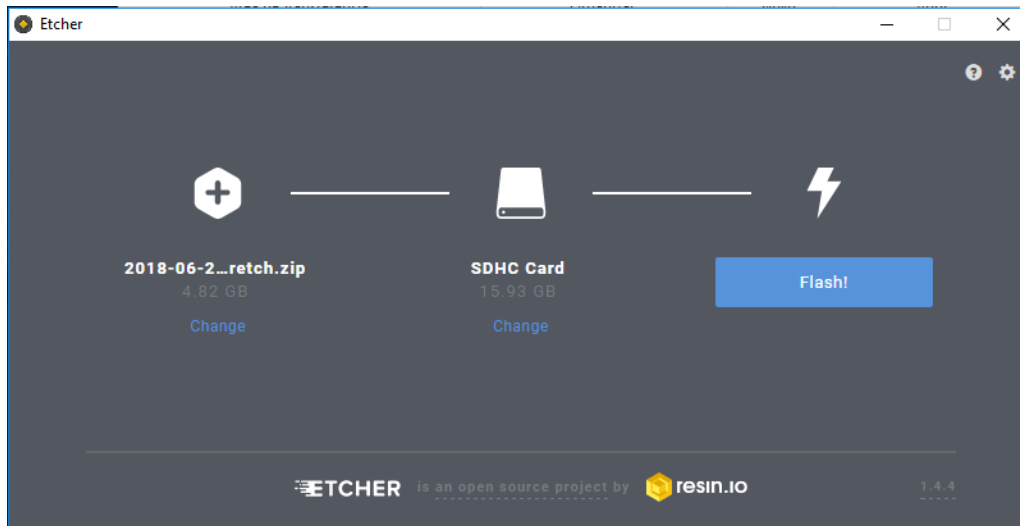


Figure 1: Writing image of Raspbian into the SD card using Etcher.

A - Installing Raspbian with an external keyboard and screen

1. **Installing Raspbian** - Connect your external screen, keyboard and mouse (if you have one), to your Raspberry Pi. Eject the SD card and insert it into the Raspberry Pi. Turn the Raspberry Pi on by connecting the power supply. The LED will start blinking. Raspian will be installed and it may reboot after installation is completed. The default login for Raspbian is *pi* and the password *raspberry*.

2. **Enable SSH and VNC connections** - Raspi-config is the Raspberry Pi configuration tool. This tool will be shown on first booting into Raspbian (but it can be acceded anytime by typing *sudo raspi-config* in the terminal). In the raspi-config menu, go to Interfacing Options. Enable VNC and SSH connections. This will allow you to access Raspberry Pi with your computer without the need of an external keyboard or a screen. You can also change other configurations in the raspi-config menu for your convenience. Once you are done, click tab on your keyboard and select 'finish'. Raspberry Pi will reboot.

3. **Configure connection to internet** - Once you get to Raspbian's desktop you will be able to set up a new Wi-Fi connection in the top right part of the screen. Insert your network credentials.

4. **Raspberry Pi's Wi-Fi IP address** - In order to connect the Raspberry Pi to a computer via SSH and/or VNC you will need to find out its IP address. Open the Raspbian's terminal and type *hostname -I*. Write down The IP address and don't loose it. You will need it to set remote access by Wi-Fi.

5. **Raspberry Pi's Ethernet IP address** - Connect the Ethernet cable to your computer and to your Raspberry Pi. Write again *hostname -I* in Raspbian's terminal. If you are also connected to Wi-Fi you will see two IP adresses. One of them is the one you find previously. The new one corresponds to the Ethernet connection. Write down The IP address and don't loose it. You will need it to set remote access by Ethernet.

B - Installing Raspbian without a keyboard and a screen

This installation process is easier if there is a keyboard and a screen available to be connected to the Raspberry Pi. However it is not impossible to install Raspbian without them.

1. **Enable SSH connection** - In Windows desktop, go to 'My computer' and open the the SD card directory. SSH can be enabled by placing a file named ssh, without any extension, onto the boot partition of the SD card. To create this file, open the notepad. The contents of this file

don't matter so you don't need to write anything. Click on *save as* and write in the file's name *"ssh"*, with the quotation marks. The quotation marks are important because they will allow the file to be saved without an extension.

2. **Configure connection to internet** - In the same boot directory, create another file with the help of notepad and call it "*wpa_supplicant.conf*". Again, the quotation marks are important, because the file should not have the .txt extension. The file must contains the following lines:

   $country = pt$
   $update_config = 1$
   $ctrl_interface = /var/run/wpa_supplicant$

   $network = \{$
   $scan_ssid = 1$
   $ssid = "RouterName"$
   $psk = "Security"$
   $\}$

   In 'country' replace the two letters by the letters that represent your country. In 'ssid' replace RouterName by your network's name and in 'psk' replace Security by your network's password. The network should be the same as your computer, otherwise this tutorial wont work.

3. **Turn Raspberry Pi on** - Eject the SD card from your computer and insert it in the Raspberry Pi slot. Connect the power supply cable to Raspberry Pi. The red and green leds in Raspberry Pi will turn on. Wait until the green light fades and move to the next step.

2 Configuring Computer Remote Access

From now on you have to choose which section is suited for you:

- If you just completed section 1, please follow subsection A - Configuring Computer Remote Access for the first time.
- If your Raspberry Pi was already configured to remote access but you want to use a new computer and/or a different network, please skip to subsection B - Re-configuring Remote Access for a new computer and/or network.
- If your Raspberry Pi is already configured for the network and computer you want to use and you just want to start configuring your Raspbian desktop, please skip to section 3: Configuring Raspbian Desktop.
- If your Raspberry Pi is already configured for the network and computer you want to use and if you already have Raspbian ready to star tracking, please skip to section 4: Running the star tracker.

The following instructions are meant to be used on a Windows computer. In order to configure remote access in other operative systems, you should follow the instructions here: Linux or MAC OS, IOS and Android

A - Configuring Computer Remote Access for the first time

The remote access can be made using Wi-Fi or an Ethernet cable. For first configuration, we will use Wi-Fi.

1. **Finding out Raspberry Pi's IP** - If your still don't know your Raspberry Pi's IP address, it is time to reach it now. You will need to download Advanced IP Scanner (download here). You don't need to install it, because Advanced IP Scanner can also be used as an executable. Run the program and click on the green arrow in the top left of the window. Wait some time and if you did all the steps right, Raspberry Pi's IP address will be shown. Raspberry Pi is by default identified as 'raspberrypi'.

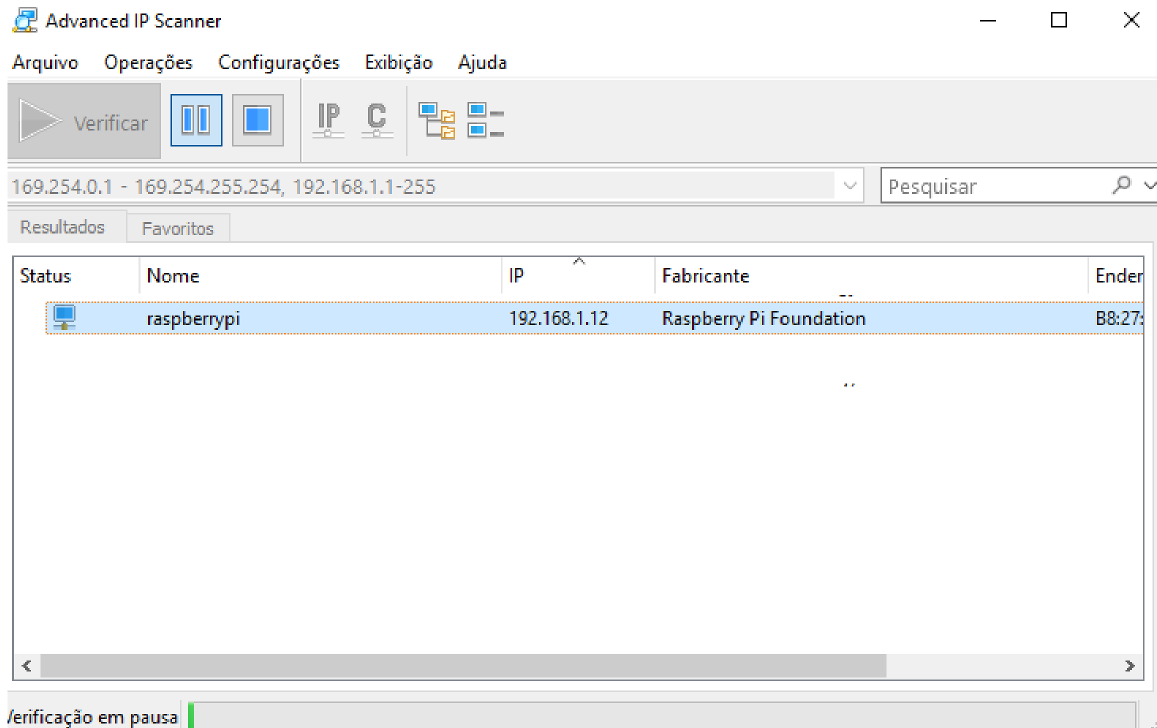


Figure 2: Advanced IP scanner window. If Raspberry Pi is connected to the same network as the computer, this program will show its IP address.

SSH - Secure Shell

The SSH only allows the user to have access to Raspberry Pi's terminal.

1. **Download an SSH client** - One of the most used ones is PuTTY (download here).

2. **Session Logging** - Run PuTTY. Type the IP address of the Raspberry Pi into the Host Name field and click 'open'. You can save the session so you wont have to write the IP again for this session/network connection.

   If the connection succeed, a terminal will open. This terminal is the Raspbian one. Log in with the same username and password you would use on the Raspberry Pi's desktop (user 'pi' and password 'raspberry'). Password doesn't show while it is being written.

VNC - Virtual Network Computing

The VNC connection allows the user to have access to the full Raspbian desktop.

1. **Enable VNC** - If you didn't use an external keyboard and screen, you still need to enable VNC to access Desktop. You can enable VNC with Raspi-config. Raspi-config is the Raspberry Pi configuration tool. This tool can be acceded by typing *sudo raspi-config* in the terminal you just opened with PuTTY. In the raspi-config menu, go to Interfacing Options. Enable VNC connection. You can also change other configurations in the raspi-config menu for your convenience. Once you are done, press tab on your keyboard and select finish.

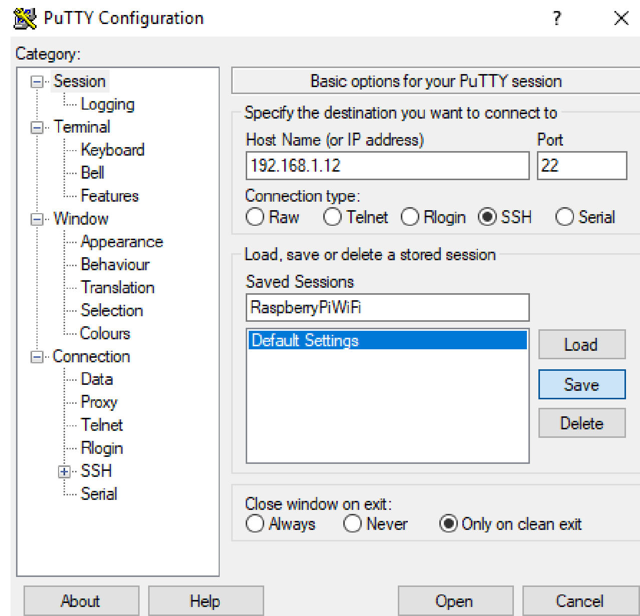


Figure 3: Putty session window. The session can be saved by clicking 'save' before opening the session.

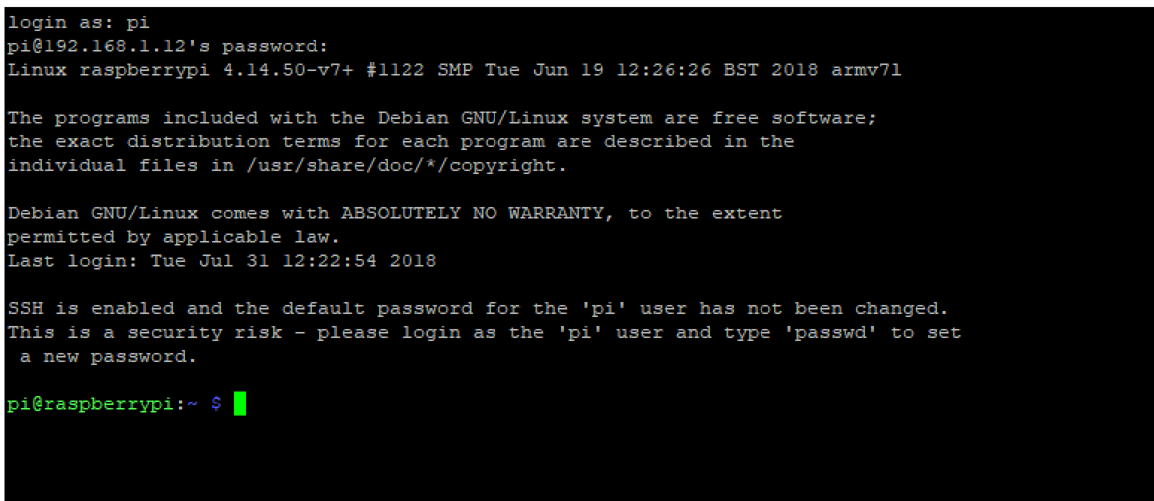


Figure 4: Raspberry Pi's terminal will pop up. Log in with the same username and password you would use on the Raspberry Pi's desktop.

2. **VNC Viewer** - Download VNC Viewer for your computer and install it (download here). You may need to create an Real VNC account.

3. **New connection** - Open VNC viewer. In the top left of the window click on 'File' and create a new connection. A new window will open. Enter your Raspberry Pi's private IP address into 'VNC Server' field. You can also give a name to this session, since it will be saved in VNC Viewer's main window. Click ok and the session will be created.

4. **Raspberry Pi's login** - Now the session is saved in VNC Viewer's main window. Click it twice. A new window will pop up. Enter your Raspberry Pi's credentials. Remember: the default login for Raspbian is *pi* and the password *raspberry*.

You may need to change the resolution of your Raspberry Pi's Desktop. Click on the raspberry in the top left of the desktop. Go to 'Raspberry Pi Configuration', and in 'System' click on Resolution. Change this parameter to the one you want. You need to reboot to see the changes.

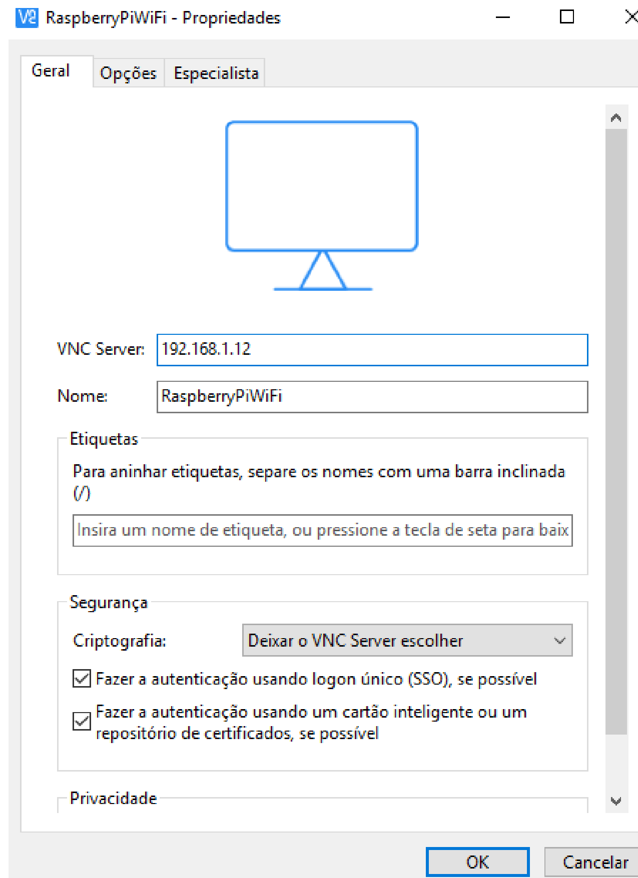


Figure 5: VNC Viewer: new connection window. Enter your Raspberry Pi's private IP address into 'VNC Server' field. You can also give a name to this session, since it will be saved in VNC Viewer's main window.

Ethernet

This step is not essential because you already achieved a Wi-Fi connection, however your connection may not be strong or can fail sometimes.

1. **Setting up Ethernet connection** - Connect the Ethernet cable to the computer and to the Raspberry Pi. Open the terminal and type *hostname -I*. You have now two IP addresses. The new IP is the Ethernet one. Now you can repeat the previous steps and configure PuTTY and VNC Viewer. You can now work with Raspberry Pi without being connected to network simply by connecting an Ethernet cable.

Skip the next subsection and go directly to section 3: Configuring Raspbian Desktop.

B - Re-configuring Remote Access for a new computer and/or network

This steps are only needed if you changed your work computer and/or are using a new Wi-Fi network.

Configuring a new Wi-Fi network with the usual computer

This steps are only needed if you want to configure a new Wi-Fi network to remote access.

1. **Remote access by Ethernet** -Connect your Ethernet cable to your Raspberry Pi and your computer. Turn your Raspberry Pi on by connecting the power supply cable.

2. **Access Raspbian's working environment** - Go to VNC and select your usual Ethernet session. In Raspbian's working environment, go to the top right of the screen and configure manually your new Wi-Fi network.

3. **Finding out the new Wi-Fi IP address** - Use the Raspbian's terminal to find out the new IP address by typing *hostname -I*. Two IP's will show: one is your Ethernet IP and the other one your Wi-Fi IP.

4. **Create new login sessions** - Use the new IP address to configure new sessions in PuTTY and VNC. (Check A - Configuring Computer Remote Access for the first time if needed)

Configuring a new remote access with a new computer

Take the SD card from Raspberry Pi and insert it in the proper computer slot using an SD card adapter.

1. **Configure connection to internet** - Go to 'My computer' in Windows desktop. In the boot directory of your SD card, create a new file with the help of notepad and call it "*wpa_supplicant.conf*". The quotation marks are important, because the file should not have the .txt extension. The file must contains the following lines:

   $country = pt$
   $update_config = 1$
   $ctrl_interface = /var/run/wpa_supplicant$

   $network = \{$
   $scan_ssid = 1$
   $ssid = "RouterName"$
   $psk = "Security"$
   $\}$

   In 'country' replace the two letters by the letters that represent your country. In 'ssid' replace RouterName by your network's name and in 'psk' replace Security by your network's password. The network should be the same as your computer, otherwise this tutorial wont work.

2. **Turn Raspberry Pi on** - Eject the SD card from your computer and insert it in the Raspberry Pi slot. Connect the power supply cable to Raspberry Pi.

3. **Finding out Raspberry Pi's IP** - Raspberry Pi's IP address is dynamic, so you'll need to find it out. You will need Advanced IP Scanner (download here). You don't need to install it, because Advanced IP Scanner can also be used as an executable. Run the program and click on the green arrow in the top left of the window. Wait some time and Raspberry Pi's IP address will be shown. Raspberry Pi is by default identified as 'raspberrypi' (check figure 2).

4. **Installation of PuTTY and VNC Viewer** - You will need to install PuTTY (download here) and VNC Viewer (download here) in your computer to remote access.

   Check subsection A - Configuring Computer Remote Access for the first time, if you need help with SSH and VNC.

5. **Create new logging sessions** - Insert the new IP into Putty and VNC and you will have new remote access sessions.

6. **Setting up Ethernet connection** - Connect the Ethernet cable to the computer and to the Raspberry Pi. Open the terminal and type *hostname -I*. You have now two IP addresses. The new IP is the Ethernet one. Now you can repeat the previous steps and configure PuTTY and VNC Viewer.

3 Configuring Raspbian Desktop

If it is not the first time your Raspberry Pi is used as a star tracker, you can skip this section and go directly to section 4: Running the star tracker.

In this section it is easier to use VNC viewer than PuTTY to work with Raspberry Pi, specially if you are not used to work with Linux terminal. Connect Raspberry Pi to power supply and open Raspbian desktop in VNC Viewer. Open Raspbian's terminal window. Make sure Raspberry Pi is connected to the internet.

1. **Installing Python 3 and pip** - Type *sudo apt-get install python3.* Once the installation is completed, type *sudo apt-get install python3-pip.* This process will install pip, a python installer that will make the process of the libraries installation much easier.

2. **Installing the libraries** - The libraries needed to be installed are pillow, numpy and scipy. In order to install this three libraries you only have to write *pip3 install name*, replacing the word 'name' by the library name you want to install. For instance, to install numpy you write *pip3 install numpy.*

3. **Install libusb** - Install libusb (download here). Open terminal in the download's directory and unzip the downloaded file. You can unzip it by typing *tar -jxvf libusb-1.0.22.tar.bz2* in the terminal. Create a new directory in your desktop called *Star Tracker.* Cut and paste the unziped files (not the folder) to this new directory and open the terminal there. Type *./configure –disable-udev* and then install libusb library with the command *sudo make install.*

4. **Install Python 2 and pip** - The arducam libraries use python 2.7, so you will need to install also python 2.7. Type *sudo apt-get install python2.7-dev* in the terminal in order to install it. Now you will need to install python dependencies libraries: *sudo apt-get install python-pip*, then *sudo apt-get install python-opencv*, *sudo apt-get install python-imaging*, *sudo pip install evdev* and finally *sudo apt-get install libopencv-dev.*

5. **Algorithm and Arducam files** - The algorithm files are available in the folder *other* of the zip where this manual is. You should paste the folder available in Raspbian's desktop, with the help of VNC or a pen drive. Insert the ArduCAM library files (step 3) inside this folder also.

You are now able to use your Raspberry Pi as a star tracker!

4 Running the star tracker

1. **Connection** - Connect Arducam Shield Module to Raspberry Pi with an USB cable. Turn Raspberry Pi on. By default, Raspberry Pi will connect to your main network. Make sure your computer is also connected to your main network.

2. **Session Logging** - Run PuTTY in your computer. Type the IP address of the Raspberry Pi into the Host Name field and click open. Log yourself in using Raspbian credentials.

3. **Run file** - Using the terminal go to the directory where the Arducam files are: type *cd Desktop/other*. The run the bashfile by typing *ṡtartracker.sh*