POLITECNICO DI TORINO

III Faculty of Information Engineering
Master's Degree in Electronics Engineering

Master's Thesis

Design and Simulation of a Star Tracker for the Aramis Small Satellite

Advisors:
prof. Leonardo Reyneri
prof. Alberto Vallan

Candidate:
Diego Urbina

Academic Year 2007-2008

Summary

Star Trackers are devices that provide higher accuracy than other attitude sensors with the added benefits of 3-axis attitude determination. Nevertheless, Star Trackers are frequently heavy, complex and costly systems that can not be adopted by small satellites such as the `Aramis` from Politecnico di Torino, which needs high-accuracy attitude determination to cover the requirements of certain types of payload.

In this thesis, the state of the art in attitude sensing is described, specially that of Star Trackers. Then, a preliminary design of a low-mass, low-cost, low-power and coarse accuracy Star Tracker is proposed to satisfy the requirements of the `Aramis` spacecraft.

Different available algorithms for identifying the presence of single stars on the imager plane are analyzed, as well as those for pattern recognition necessary to ultimately measure the spacecraft attitude. One set of such image processing and pattern recognition algorithms are chosen for use on board `Aramis`. Subsequently, they are tested with the experimental use of the 3D open source planetarium Celestia, while a parallel test of the image processing algorithms is performed on real star field imagery to confirm their capabilities with real-world data.

A scheme is proposed to reduce the amount of false results thanks to the use of attitude approximations coming from other sensors, through the homogeneous segmentation of the celestial sphere.

Commonly used methods to produce the desired quaternion output are described, and finally, an assessment of the performance of the tested algorithms is made.

Acknowledgements

I thank the following persons who were consulted during the development of this thesis: prof. Leonardo Reyneri, prof. Alberto Vallan, prof. Nikolai Tolyarenko, Dr. Angie Buckley, Dr. Jeff Hoffman, Danilo Roascio, Fabio Maggioni.

I also want to thank all my friends that made me feel at home during all these years away from home, Elsa Marie for putting a smile on my face when I most needed it, and my lovely family, specially my mother who has been the Northern Star that helps me not get lost in space.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the now increasing trend of the use of microsatellites, the cost of production and launch has decreased as much as their size, thanks to the use of miniature technologies, new and cheaper launch vectors, and piggyback launches.

Microsatellites provide great advantages besides their convenient price tags. Their characteristics are perfect for the creation of low-data rate communications constellations, their potential for formation flying is quite promising for multiple-point information gathering and finally, their relative simplicity is adequate for educational purposes. This last characteristic was observed by the faculty of Standford University, and in 2001, a new standard called CUBESAT was created to facilitate the development of this sort of spacecraft.

This standard was broadly adopted in many Universities around the globe, among many other institutions with a need to study and develop satellites within shorter amounts of time and with less resources, as opposed to multimillion-euro satellites with weights that can achieve a few tons.

One of the adopters of this technology was Politecnico di Torino, which in 2006 finished its CUBESAT Satellite, PicPot.

After the loss of the PicPot during launch, a new small satellite system was envisioned, built on the grounds left by PicPot, the `Aramis` Small Satellite.

This chapter will provide an overview of the PicPot, the `Aramis`, the space environment in which they operate, a brief description of the ADCS subsystem, its sensors, and more specifically, of Star Trackers.

1.1 Aramis' predecessor: PicPot

PiCPoT, an italian acronym for Piccolo Cubo del Politecnico di Torino (Small Cube of Politecnico di Torino) was the first nanosatellite developed by the University, a

project undertaken with the collaboration of different departments, namely the Electronics Engineering Department, Aerospace Department, Energetics Department and Physics Department. This, with the objective of making it a multidisciplinary project with a homogeneous representation from different pertaining fields.

PiCpoT was a CUBESAT, equipped with five solar panels, three cameras, two antennae, one kill switch, and a test connector.

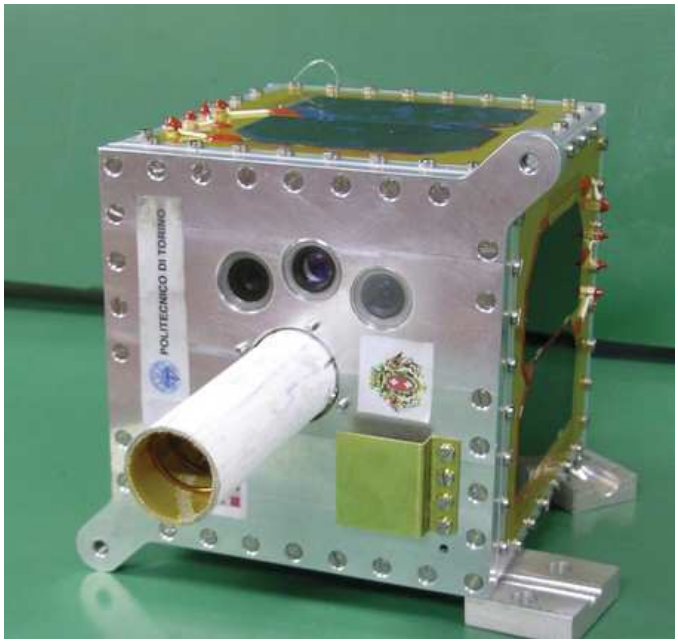


Figure 1.1: PicPot, Politecnico di Torino CUBESAT

Other than showing the possibility of a multi-department collaboration in the Politecnico, it had the objective of verifying the reliability of the Commercial Off-The-Shelf components in a space environment, and obtaining direct knowledge of the environment in Low Earth Orbit and Medium Earth Orbit.

PiCPot was composed by a set of basic subsystems.

- The Power Switch was the one in charge of generating the necessary voltages to keep all the other subsystems in function. It selected the adequate battery and solved Latch-Up events.

- The Power Supply watched the state of the batteries and the solar panels. ProcA and ProcB were the two on board processors, and executed the same operations; they adquired data from the telemetry sensors, administered the payload and sent/received data from the groundstation.

- The Payload was a board designed to acquire the images from one of the three on board cameras, and converted the raw images to JPG format, so they could be transmitted to earth.
- The TxRx subsystem was in charge of sending and receiving commands to and from the ground via 437 MHz for the uplink and 2.4 GHz for the downlink.

The launch was performed through a Kosmotras-operated Dnepr rocket, but a failure in the hydraulic system led to unexpected instability, which forced the emission of an emergency command in order to shut down the motor. [2]

Figure 1.2: Dnepr launch. Source: Christian Hessmann

Although the launch was a failure, the project was highly successful in establishing an environment of space research in the Politecnico, which would lead to a second generation of satellites made in the University.

1.2 The Aramis Micro Satellite

`Aramis` is a new type of satellite, evolved from PiCPoT, although its objectives are now much broader.

The intention is to create a highly flexible satellite, which won't have to be redesigned almost from scratch for each different mission, and will only have a different configuration thanks to the possibilities given by a modular architecture.

The `Aramis` project foresees the use of three basic modular blocks. These are:

Figure 1.3: Aramis satellite in a 2x2x1 configuration

- **Power Supply Block:** it allocates the energetic resources and has inside the atitude control systems, by comparison, it comprises the Solar Panel, Power Supply, Battery and Power Switch subsystem used in the PiCPoT setup.
- **TxRx Block:** it is in charge of communicating with the ground station in a bi-directional manner, sensing incoming and outputting outgoing data.
- **Payload Block:** This block can be used for anything the mission requires, from Synthetic Aperture Radar sensing to image acquisition, or many other different possible experiments on the particular conditions of microgravity, vacuum or radiation present in the earth orbit.

These blocks are coordinated by an On Board Computer (OBC), controlling the overall functioning of the satellite.

Finally, a ground station that adheres to the ESA GENSO network will be developed. This will dramatically increase coverage, compared to a single ground station, allowing the uploading/downloading of data at more than one single point of the Earth.

1.3 Considerations on the Space Environment

A spacecraft operating in space is exposed to an environment significantly different from that of the earth. This has a great deal of influence on the design of the

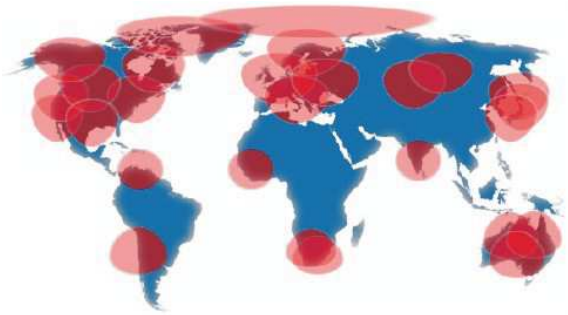


Figure 1.4: GENSO educational network coverage

different parts, specially those of electronic nature.

In the environment close to the Earth, there are multiple possible sources of ionizing radiation that can hit a spacecraft.

The Van Allen radiation belts have a high density of energetic protons and electrons that come from the sun and get caught in Earth's magnetic field. They mostly follow the path of the magnetic field lines. There may be also be considerable fluxes of He, N, and O in the form of heavy ions.

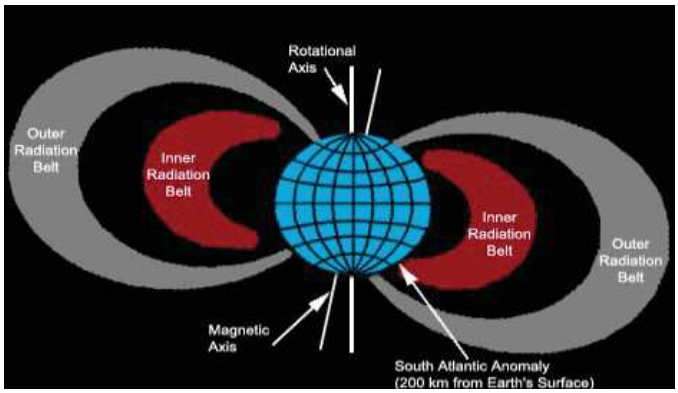


Figure 1.5: South Atlantic Anomaly and Van Allen Belts. Source: NASA.

Proton energies can go from 0.01 to 400 MeV with fluxes of about 10^8 to $600/cm^2s$, respectively, and electrons can have between 0.4 to 4.5 MeV and fluxes between $4 * 10^8$ to $100/cm^2s$, respectively.

When orbiting low altitudes, with a low inclination, the most important radiation feature is the South Atlantic Anomaly (frequently abbreviated to SAA). Because of the offset of the magnetic field of the Earth with respect to the axis of rotation, this is the region where a part of the Van Allen belt is brought to low altitudes ($\sim$ 500 to 1000 km above the earth).

Primary cosmic rays are also a source of particles with great kinetic energy. Despite their name, they should not be confused with electromagnetic radiation. Primary cosmic radiation in LEO consists of about 83% protons, 13% alpha particles, 1% nuclei with an atomic number greater than 2, and 3% electrons. The intensity of cosmic rays is dependent on the solar cycle, and it diminishes as the sunspot number cycle escalates.

Sun activity also plays an important role in spacecraft design: solar particle events are another ingredient of the mix of radiation in Earth orbit. They are random waves of solar activity, these are, Solar Flares, in which the sun emits strong electromagnetic radiation along with electrons and protons that stream out of the Sun through a "corridor" established by the interplanetary magnetic field. The rate of this emissions oscillates in a cycle that lasts eleven years.

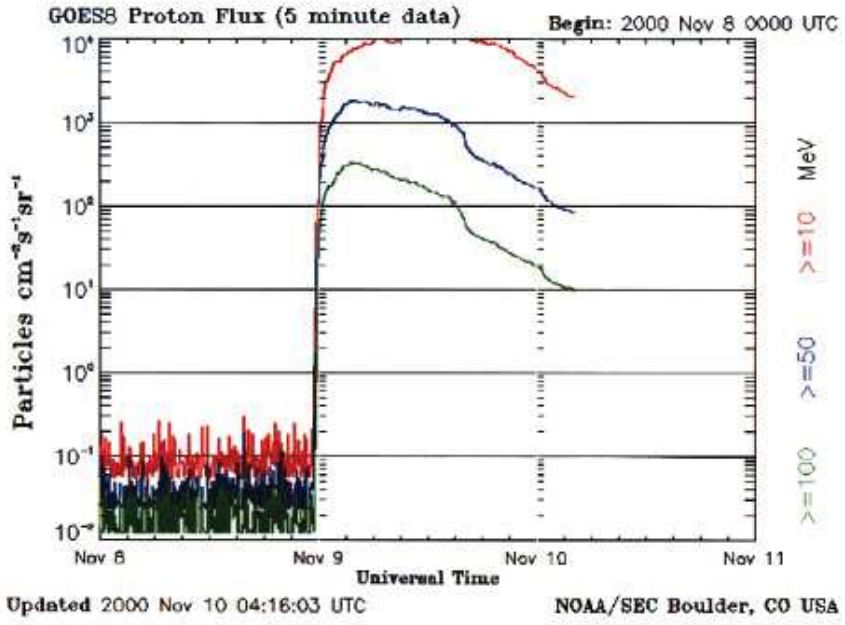


Figure 1.6: STARDUST: GOES-8 graph describing the solar wind event. Source: NASA.

Another source of problems for space hardware are Coronal Mass Ejections or CMEs. They are massive expulsions of gas and charged particles at hundreds of kilometers per second.

A third effect, is the continuous loss of the Sun mass into space, a permanent flow of a million tonnes per second in every possible direction. This is called Solar Wind, and it is composed by electrons and protons that travel in the range between 300 and 700 kilometers per second, and with a density near the earth of about 6

particles per cubic centimeter. It can, though, vary strongly in a matter of hours or minutes, as well as in the long 11 year Sun activity cycle.

All this factors should be taken into account when choosing the components that will constitute any device that flies into LEO, for this matter, the `Aramis` Star Tracker.

Phenomena, known as Single Event Effects (SEEs) are originated by a single particle carrying great energy. They are divided in different classes, depending on their consequences in the operation of electronic circuitry. The types of SEE include: [3][4]

Single Event Upsets (SEU) are soft errors, and do not imply the destruction of the device. They usually manifest themselves as transient pulses or bitflips, and can affect digital, analog and optical components, as well as interface circuitry. They are called *soft* errors since a reset or rewriting makes the device return to its normal state and behave in a correct manner afterwards. **Multiple Bit Upsets (MBU)** are the particular case of SEUs in which more than one bit flip is produced in different points due to one or more striking particles.

Single Hard Errors (SHE) cause permanent change in the operation of a component, for example, a stuck bit in a memory.

Single Event Latchups (SEL) are the case in which the device functionality is lost, caused by a high current state produced by a single event. It is possible for a SEL to cause permanent damage. It requires power strobing of the device in order to regain stability.

Single Event Burnout (SEB) is the condition in which device destruction is caused due to a high current state in a power transistor.

Single Event Gate Rupture (SEGR) is a condition caused by a single ion in power MOSFETs and can produce a conducting path in the gate oxide.

In order to measure the energy deposited per unit length when an energetic particle travels through a material, a value called **Linear Energy Transfer (LET)** is used. The most commonly used LET unit is $\frac{MeV \cdot cm2}{mg}$ of material. Furthermore, a threshold of LET can be established, it is called indeed, the **Threshold LET (LETth)**, and it is the minimum LET that can cause an effect at a particle fluence of $1 \cdot 10^7 \frac{ions}{cm^2}$.

A Star Tracker, as it will be seen in more detail in the next chapters, works by acquiring images of the starfield. These images can be corrupted when radiation events occur. A particular case of SEU can take place, affecting the imager with a relatively long duration.

One example of solar flare radiation affecting the functioning of a Star Tracker was an incident involving NASA's STARDUST. The spacecraft was at 1.4 AU from the sun, heading towards the earth, when a solar wind passed through STARDUST, and a wave of protons impacted the CCD camera, producing hundreds of star-like

Figure 1.7: STARDUST: Last acquired image under the effect of a solar flare. Source: NASA.

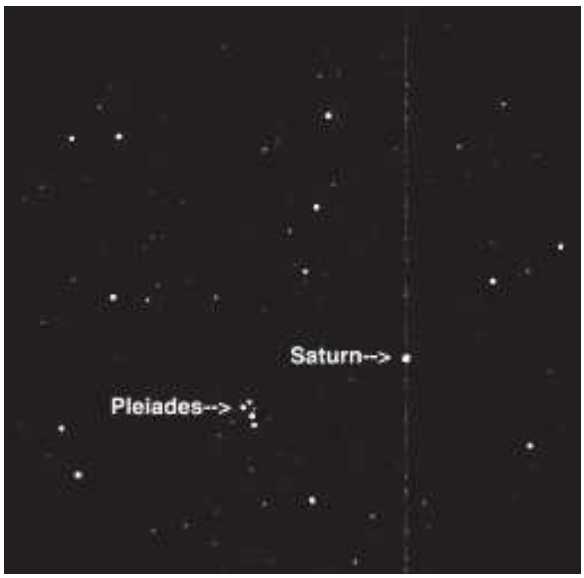


Figure 1.8: STARDUST: Image from the Star Tracker some days after, showing stars and a planet. Source: NASA.

dots, as seen in figure 1.7.

The STARDUST Star Tracker uses the 12 brightest stars for pattern matching (more detailed descriptions of this process will be given in successive chapters as

well), so when its two Star Trackers had a great quantity of false stars and it couldn't get a read of the real star field, the spacecraft entered safe mode, with its panels pointing to the sun, spinning and awaiting instructions from Earth.

The recovery was possible only a few days later, when the effect of protons was gone and one of the Star Trackers could acquire actual stars, as can be seen in figure 1.8. The magnitude of this solar wind can be visually recognized in figure 1.6.

SELs are handled by the power management subsystem, avoiding the complete loss of the Star Tracker. SEUs are mitigated by the use of less radiation-susceptible components, specially memories; and in the case of the Imager, by invalidating any attitude reading that comes from SEU-affected images.

1.4 Attitude Determination and Control

Attitude Determination and Control Systems (ADCS) are those in charge of establishing the orientation of the satellite with respect to an inertial reference frame. The attitude of a spacecraft at any time can be described by three values, a roll angle, a pitch angle and a yaw angle. The system then controls the body axes in a way that errors in roll, pitch and yaw are within determined error margins.

Bodies in space that are not controlled, such as asteroids or space debris, will tumble. For example, the Sputnik I Satellite tumbled this way [5]. Needless to say, natural tumbling is not allowed in modern spacecraft, due to the fact that the solar panels must be pointed to obtain solar energy in order to work.

The orientation that the satellite needs will be a characteristic of each mission. The bus or structure is the mounting base of one or more payloads, and depending on the characteristics of each of these and of the satellite subsystems, they will have to be pointed in specific directions. For example, solar panels will have to be pointed at the Sun and antennae to their target.

It is common that the orientation has to be established with respect to a frame of reference based on the Earth, for example, it may be required that the face with a camera will need to point to the center of the Earth (nadir).

Satellites have different configurations, such as spinners, or dual spinners, that take advantage of the inertia of the satellite to keep it pointing at a determined location without having to intervene, or doing so very little. `Aramis` though, is a three-axis stabilized satellite, which allows for relatively quick changes in attitude.

As of the moment of development of this thesis, `Aramis` was not foreseen to have *pointing mechanisms*. These are mechanisms in which parts of the spacecraft move independently to orient themselves towards a desired location. In general, this mechanisms shall be avoided when not necessary [6].

The accuracy needed by a satellite is generally determined by the payload. When using Earth Observation sensors or telescopes, an accuracy of the order of the arc

seconds is quite common. Accuracy of control is less than the accuracy of measurement.

Attitude, the quantity our sensor will measure, is a straight forward concept. A frame of reference must be adopted (the concept of frame of reference will be elaborated in a later section), once this has been done, the attitude will be the deviation with respect to this reference.

Attitude quantification can be effectuated in various ways, such as Euler angles, direction cosine matrixes, or quaternions. These will be explained at a later point, along with the frames of reference.

1.5 Attitude Sensors

Every spacecraft requires attitude sensors to obtain data that can be used to calculate a reference of its position and pointing its antennae, cameras or other sensing and communication devices during a mission.

Attitude sensors come in a wide variety of types. The sensor used in any specific mission depends largely on the characteristics of the mission, since, as previously stated, the accuracy required by each mission may vary. The sensing device must achieve a balance between accuracy, size, computational power and Field of View (FOV) constraints (as we will see, some sensors may work only in the day, while some others may work only if the sun is not pointing directly towards them, and so on).

To further elaborate on this, and justify the use of a star tracker in the `Aramis` setup, we will analyze the most used sensor systems in modern spacecraft:

1.5.1 Horizon Sensors

Earth horizon sensors are used mainly for navigation and weather reports. They consist in a camera working in the infrared zone of the spectrum that detects the contrast between the low radiation of the cold space and the heat of earth's atmosphere. The system is able to determine attitude through an image processing algorithm that can effectively determine the horizon respect to the spacecraft and establish the vector to the earth (nadir) [7].

This system has the advantage of being able to work during the night as well as during the day, and is the method in which light reflected form the spacecraft has the lowest effect on the image, as opposed to attitude sensing systems that use the visible part of the spectrum. Its accuracy is not very good though, as it is typically 0.1 to 0.25 degrees. For higher accuracy, it is necessary to correct data for Earth oblateness and horizon variations that occur at different seasons.

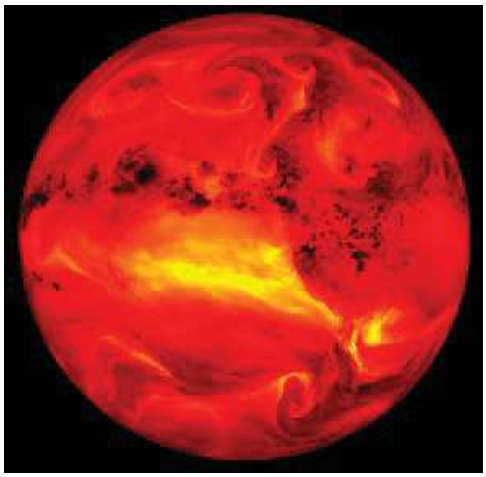


Figure 1.9: Planet Earth as seen in the infrared spectrum by the GOES-N Geostationary Satellites. Source: NASA.

1.5.2 Radiofrecuency beacons

These systems can be used as a pointing reference. The drawback is that if high accuracies are needed, a directional antenna is needed. This method is only able to determine a pointing direction but it can achieve accuracies of up to 1 arcminute. [8]

1.5.3 Solar sensors

Solar sensors detect the position of the sun with respect to the spacecraft, and are commonly used to hide sensitive systems from heat or intense light, and also to position solar panels.

Usually, a solar sensor consists in an optical sensitive chip, with a MEMS optics structure on top (a single pinhole or an array of them) that projects the sun on chip and its image is processed by an external logic.

One of the advantages of using Solar Sensors is that the sun, as far as the spacecraft is concerned, keeps a constant radius, so a point approximation can be made. It also has an adequate accuracy for many applications. Further accuracy improvements can make use of the diffraction rings that form on the sensor plane.

Other implementations take solar panels used to recharge the battery as an attitude sensor, albeit a very rough one.

The main disadvantage of this type of sensors is that no matter what, it simply will not work at night. This is a great disadvantage if the mission should want, for example, to engage communication or take photographs on the dark side of the earth.

Figure 1.10: A sun sensor for microsatellites from Jet Propulsion Lab, Caltech

1.5.4 Magnetometers

Magnetometers are simple, reliable and lightweight sensors that consist in a set of three coils that measure the intensity of the magnetic field in up to three axis [9], and with a model of Earth's magnetic field, is able to determine the attitude.

Magnetometers have the advantages of being a sensor and actuator at the same time, and of being relatively simple. The disadvantage is that the information provided is limited to two axes, a great margin of uncertainty is due to the not complete knowledge of the earth's magnetic field, its shifting, and the fact that they can only work under about 1000 km above the surface of the earth. The latter though, is not a limitation for the `Aramis` since it will operate in LEO.

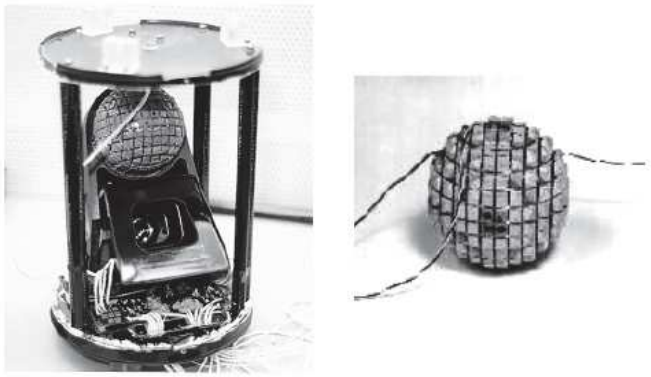


Figure 1.11: The Oersted satellite magnetometer (right) mounted on the same platform with a Star Tracker (left). Source: Danish Meteorological Institute.

1.6 Star Trackers

A Star Tracker, whose design is the aim of this thesis, is an apparatus essentially composed by an optics system, a sensor and an electronics assemble which does the image processing and carries on with the attitude estimate algorithms, aided by a memory bank.

The sensor can be a Charged Coupled Device (CCD) sensor (heavier and more expensive but less sensitive to dark noise -a feature that would be well appreciated in our device-) and CMOS (less expensive and performing, but some CMOS sensors are sensitive enough for a Star Tracker application).

Star Trackers identify stellar patterns and compare them with previous images or a database of stars stored on board. This results in an attitude measurement of the star tracker with respect to the celestial sphere, which can then be translated to the attitude of the spacecraft with respect to an inertial reference frame.

The position of the stars is extremely reliable and numerous catalogs exist. There are cases of bright stars disappearing, such as a star the Galileo Probe saw disappear while it was using this celestial body to stabilize its attitude [10], we should keep in mind nevertheless, that these disappearing stars are extremely rare.

A Star Tracker, not depending on the earth or the sun, is a very flexible subsystem and also gives the best accuracies between all the common attitude sensors. The most complex (heavy , expensive and *power hungry*) Star Trackers are able to provide attitude accuracies of the order of some arc-seconds.

In the recent past, given the high precision that Star Trackers can achieve, multi million euro ventures and international space projects used heavily Attitude Control Systems based on very complex and expensive Star Trackers. We will try to obtain most likely an accuracy better than a Sun Sensor, but certainly lower than these state-of-the art Star Trackers. Using this relatively new niche of star trackers will allow us to build a sensor that can achieve a good balance between cost, simplicity and accuracy.

The Star Tracker is the ADCS sensor of choice for the `Aramis` satellites that require fine attitude determination.

The study and development of Star Trackers for nanosatellites is currently a hot topic in many research institutions around the globe, due to the great mass and price of this type of attitude sensor, which, as stated before, makes it prohibitive for its use in small satellites. An example can be seen in table 1.1.

The use of COTS and the fact that coarse results, as opposed to high precision measurements, will be obtained, allow the price reduction of the `Aramis` Star Tracker. For a comparison with the current state of the art, figures 2.14, 2.15 and 2.16 are provided.

Star trackers are divided into Star Scanners, which scan the whole celestial sphere using the attitude correction system of the spacecraft, gimbaled Star Trackers, which

Figure 1.12: A-Star Tracker from Galileo Avionica

Sensor Type	CCD
Tracked Stars	Up to 10
Operation modes	LIS (Lost in Space) and Tracking Mode
FOV	16.4
Roll rate	0.5 deg/sec @ full accuracy, 2 deg/sec @ reduced acc.
Bias error	<10 arcsec
Low frequency error	<7 arcsec (pitch and yaw) <25 arcsec (roll)
Random error at 0.5 deg/sec	<9 (arcsec pitch and yaw) <95arsec (roll)
Random error at 2deg/sec	<20 arcsec (pitch and yaw) <210 (arcec roll)
Operation temperature	-30 to +60 deg C
Power consumption	8.9 W +
Mass	**3 kg** with baffle
Size	**195(L)x175(W)x288(H) mm**

Table 1.1: Galileo Avionica A-Star Tracker Specifications

have a moving head to scan a considerable portion of the stars available, and fixed-head Star Trackers, which stay still and take images of a limited FOV. The latter will be the model we will center our efforts in for this project, for the sake of the relative "simplicity" required by the `Aramis` Subsystems.

Type	$\frac{Size \cdot Mass \cdot Power}{Savings}$	Operating angle	Axes	Accuracy
Magnetometer	Low	Full sphere, but needs no mag. interference	3	1 arcminute
Radiofrequency beacon	Medium	Narrow: Nadir$\pm \sim 10°$	2	1 arcminute
Horizon sensor	Medium	Narrow: Nadir$\pm \sim 10°$	2	5 arcminutes
Sun Sensor	Low	Narrow: Sun$\pm \sim 30°$	2	1 arcminute
Solar Panel	Low	Narrow: Sun$\pm \sim 30°$	2	1 degree
Star Tracker	High	Wide: Full sphere except earth or sun	3	25 arcseconds
`Aramis` Satellite Star Tracker	Low	Wide: Full sphere except earth or sun	3	200 arcseconds (goal)

Table 1.2: Current sensor types comparison

Chapter 2

Architecture

This chapter describes the requirements of the `Aramis` Star Tracker, defines its functional composition and proposes different components based on their performance and on their capacity of operating in the space environment described in chapter 1.

A Star Tracker is basically composed by three sections. The optics, an imager and a processing unit. The optics collect light with different schemes, they can be pin-hole-based or lens-based. The imager detects the impinging photons and translates them into electrons. This signal is then transmitted to the processing unit that is in charge of interpreting the provided image to obtain relevant attitude data. It then sends the data over the Satellite bus and also receives commands over the latter.

As an example on figure 2.1 we can see what the Oersted Advanced Stellar Compass [11] scheme looks like. The lens collects light from the stars and the CCD converts it to an analog signal. This signal is amplified and converted to a useful digital signal by the frame grabber. The CPU, which has already loaded star data from the Flash PROM into the DRAM, processes the image data and sends it back to the interface circuit, that will send it to the bus. Commands from a central computer give the interface circuit necessary instructions of operation.

2.1 Requirements

The requirements for the `Aramis` satellite Star Tracker are:

- It shall provide an indication of the current attitude of the satellite.
- It shall have an accuracy of the order of the hundreds of arc-seconds.
- It shall have a low mass: `Aramis` is a small satellite. The size and mass of each component is critical. Typical commercial Star Trackers have a mass

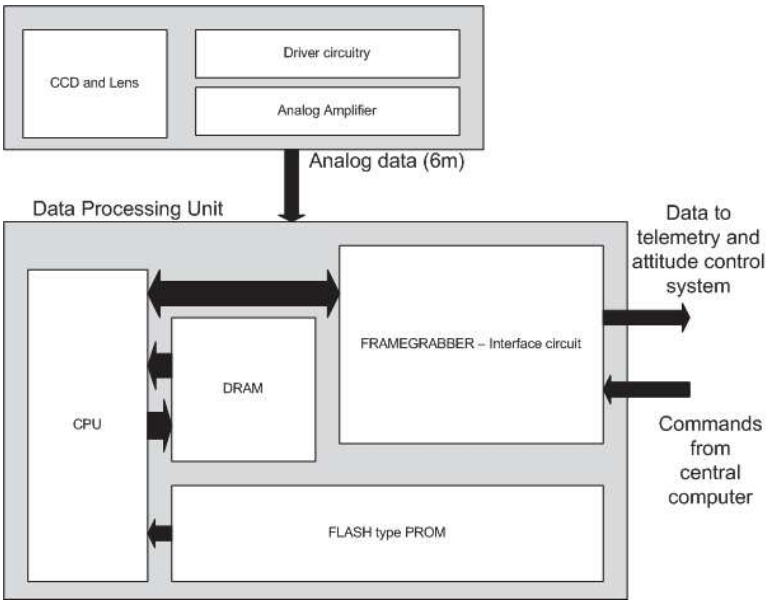


Figure 2.1: Oersted μASC

comparable to that of a whole `Aramis` Satellite, and a size greater than one of its tiles.

- It shall have a low consumption: It shall use low consumption components where possible.
- Full autonomy preferred, although it could use the help of other sensors (sun sensor, magnetic sensor) for coarse approximations.
- It shall be able to determine attitude with Roll rates of up to $10^o/sec$
- It shall use COTS (Commercial Off-the-Shelf) components which are not easily affected by cosmic rays.

Additional considerations for the Hardware/Software

- Stellar database should be encoded with a TBD error correction code in Flash Memory to protect it from SEU
- Periodic reloading of Stellar data into RAM
- SEL protection done by Power Subsystem

The `Aramis` Star Tracker will be divided in two parts, one will contain the optics and the sensor (the Star Tracker Camera Unit - SCU) and another one (Star Tracker Processing Unit - SPU), all the elements required by the CPU to process data (memory, energy management, oscillators, etcetera).

This will allow more functionality in physically mounting the setup, because the camera unit will be able to stand in places that would not be so accessible or practical if the sensor was mounted on the same bulky board as the CPU.

Figure 2.2 shows the components of the Stellar Compass.

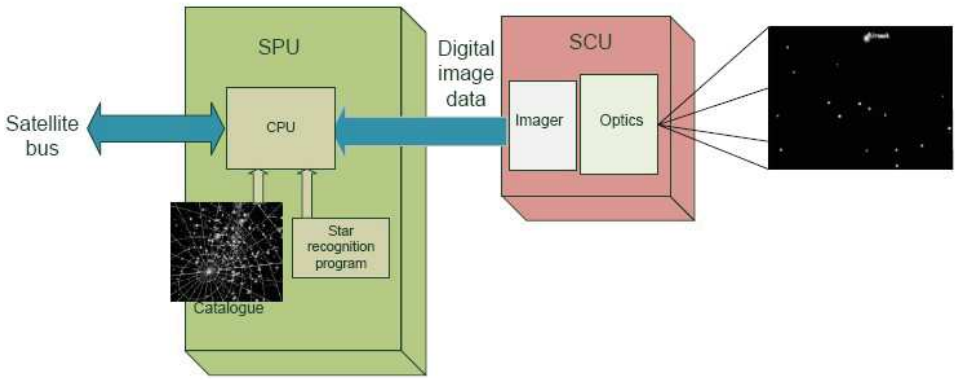


Figure 2.2: Star Tracker block diagram

2.2 UML Model

The Unified Modeling Language, or UML is a graphical language used in order to depict, specify and build a system. UML models are widely used for systems that require software development, as they can be transformed into software and facilitate the process of collaboration between the different individuals and entities involved in a project.

UML has 13 types of diagram, three of the most important ones are the following:

- **Class Diagram** a Class Diagram is a kind of structure diagram that describes the system composition through the use of classes. The attributes for each class are specified, as well as the relationships between classes.

- **Sequence Diagram** a Sequence Diagram is a graphical representation of the processes in a system, and the way they operate between them, and their correspondent order. It shows the processes that coexist at the same time, as vertical lines, and the messages exchanged between them, as horizontal arrows.

- **Use Case Diagram** a Use Case Diagram has as a purpose, to show a condensed vision of the functionality of the system, through the use of actors, that are entities external to the system and can interact with the system. Every actor has a goal. This goal is shown, as well as eventual dependencies between the use cases.

The Use Case Diagram for the Star Tracker is provided to facilitate the interaction with other entities inside the satellite, to illustrate the intended functioning of the Star Tracker, and to provide a base for the development of the code.

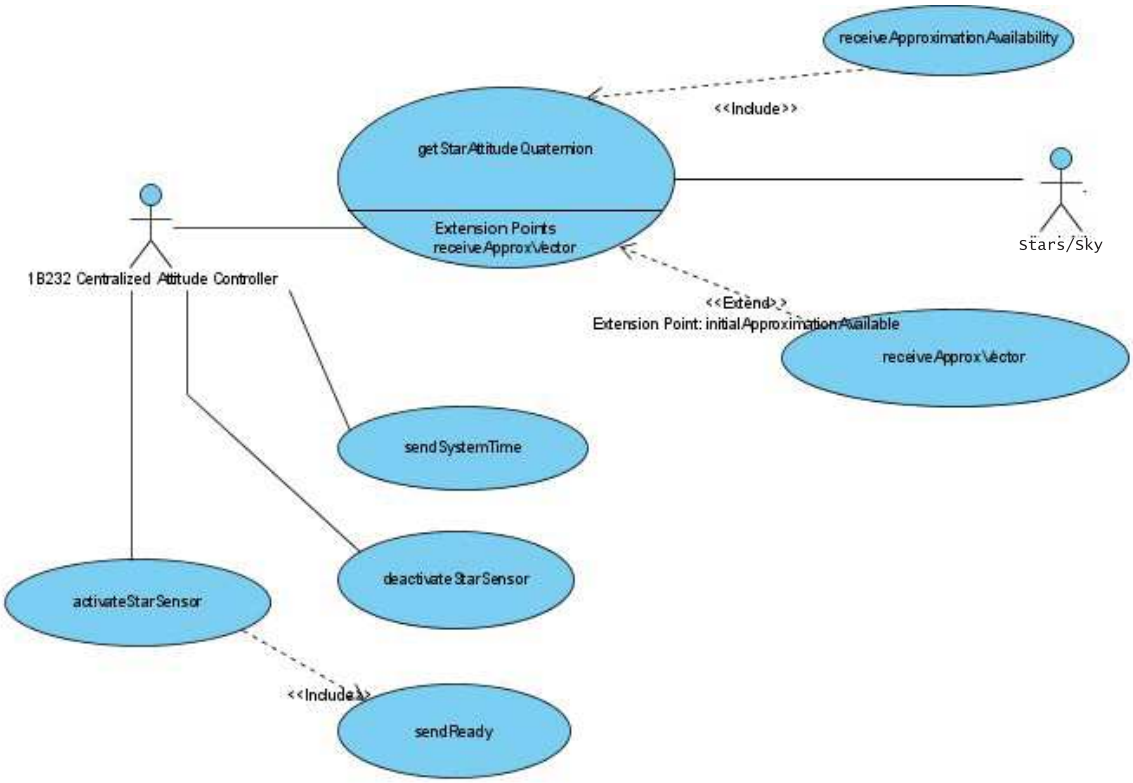


Figure 2.3: Use cases of the Star Tracker with 1B232 and Stars/Sky as actors

Five actors can be identified:

- **1B232 Centralized Attitude Controller** It is the AOCS computer in charge of coordinating the inputs from the sensors, the desired attitude as an input from the ground or a satellite routin, and the signals sent to the actuators to correct the orientation.
- **1B1 Power Management Subsystem** It is the subsystem that powers every other subsystem in the satellite.

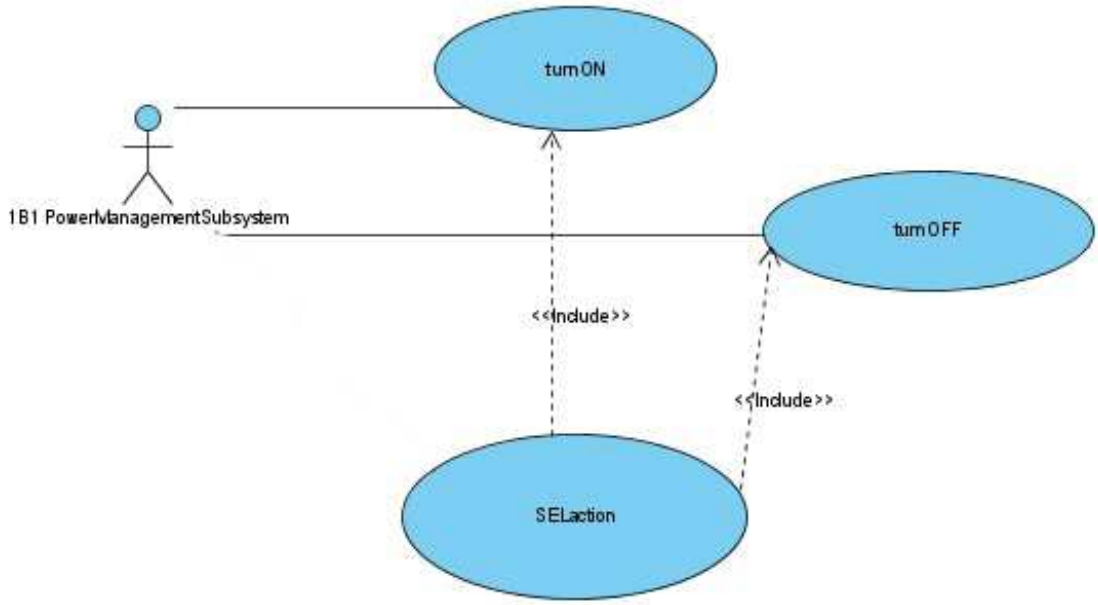


Figure 2.4: Use cases of the Star Tracker with 1B1 Power Management Subsystem as an actor

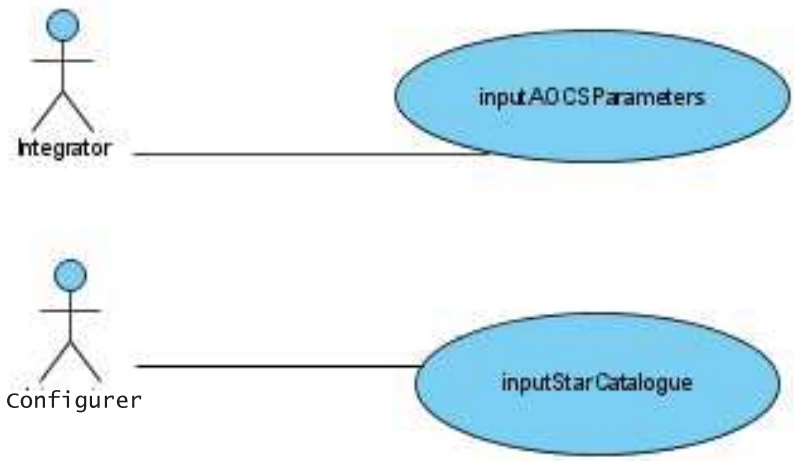


Figure 2.5: Use cases of the Star Tracker with Integrator and Configurer as actors

- **Stars/Sky** The celestial sphere from which the Star Tracker obtains a reference to produce an attitude output.

- **Integrator** It is the person on the ground in charge of setting up the Star Tracker and putting it in operational conditions in the spacecraft assembly.
- **Configurer** The person in charge of providing the required catalogues and files.

The event sequence of the Use Cases are as follows:

activateStarSensor

CAC sends a ACTIVATE_STAR_TRACKER command

ST activates the Star Tracker, loads libraries, initializes the Star Tracker, sends a ACK indication to CAC when the Star Tracker is ready to receive requests.

deactivateStarSensor

CAC deactivates the Star Tracker.

sendSystemTime

CAC sends the system time in a TBD format, most likely Julian Date to the Star Tracker.

ST Confirms its internal clock calibration

getStarAttitudeQuaternion

CAC requests the current attitude quaternion and indicates if it needs a LIS measurement, or a measurement with an initial approximation, indicating its availability.

ST confirms that it is ready to make a measurement.

CAC sends the initial approximation in a TBD format that can be an unitary vector or RA/DEC in the celestial sphere frame of reference.

ST sends the resulting attitude quaternion along with the time at which the image was acquired, or an indication of the lack of an attitude quaternion, along with the failure reason.

SELaction

CAC detects a Single Event Latchup due to an energized particle hitting a device

- calls the turn OFF use case
- calls the turn ON use case

inputStarcatalogue

Integrator inputs all the databases required for the functioning of the device, including the triangle database and the starfield database.

inputAOCSparameters

Integrator inputs the quaternion rotations necessary to convert the startracker local reference frame into the spacecraft local reference frame.

The class diagram of the hardware, on the other hand, can be seen in figure 2.6.

2.3 Star Tracker Processing Unit (SPU)

The Star Tracker Processing Unit is the section in charge of analyzing the star pattern and produce an attitude indication output.

2.3.1 CPU

The Star Tracker processing unit should use a CPU that has a good balance between high performance and low consumption, as the image processing algorithms must be executed fast enough. The processor should be able to handle data from high resolution, most likely megapixel images. The pipeline execution should be quick enough to process the image in near-real time to ready the star tracker for the next frame capture; great delays are not acceptable.

Image processing will be not the only duty for the CPU, it will also have to search through star data and perform the operations to calculate the attitude quaternion, each function shall not compromise the other ones, given that the real time constraint for the star tracker is evident.

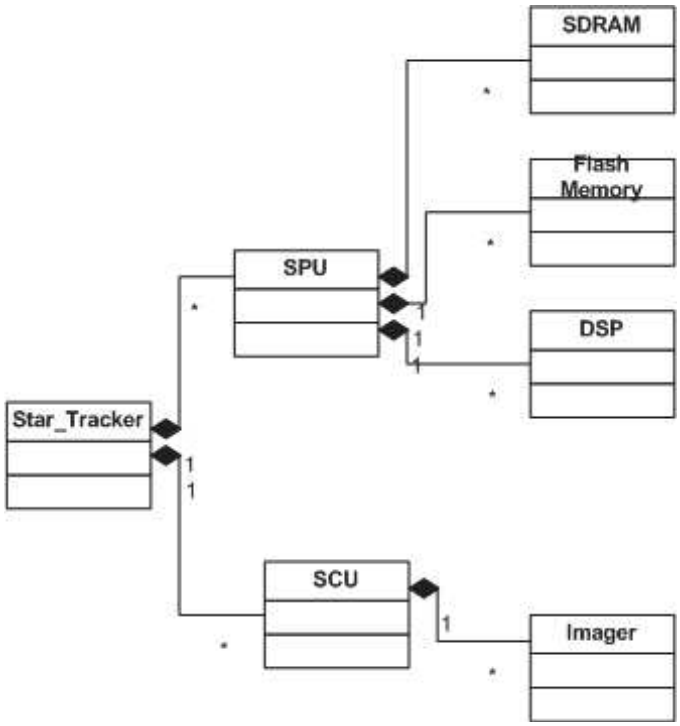


Figure 2.6: Star Tracker Hardware Class Diagram

Power management functions such as a programmable voltage regulator and low power state would be a plus since a waste of energy in the satellite is not affordable, with a power unit struggling to remain charged while payload, AOCS and communication modules take energy away from it.

The CPU unit should be as immune as possible to radiation-generated SEL effects, as the latchup of the device would result in a single point failure.

2.3.2 Flash Memory

The flash memory has the purpose of storing the boot data for the CPU and the stellar data with the coordinates of each star on the celestial sphere that we intend to use.

Enough space for the planar triangle catalogue or the two-star angle catalogue should be left. The planar triangle catalogue, through the chosen algorithm, among those described in chapter 5, resulted in 21.245 Megabytes of memory in an ASCII file of double-precision numbers. This, though, is an upper bound: we used more than 21^o of FOV and allowed the 6 brightest stars per FOV to stay, and the most likely configuration will use optics with a 18^o FOV, which produces much less triangles than the former.

On the other hand, the alternative two-star angle catalogue occupies much less memory than database for the planar triangle method. Additional space for the overhead of the Hamming code would be needed.

The Flash memory should also have space for the code that runs the CPU. The exact size of the memory required for this function will be determined upon completion of the translation of the MATLAB code into C code, and cross-compilation for the Blackfin DSP.

2.3.3 RAM Memory

RAM memory should be able to hold a grayscale 8-bit image. The estimated filesize for an image generated with the chosen sensor is 1.25 MB. If more complex algorithms should be formulated, more images could be needed in the Random Access Memory, therefore, more RAM would be a plus.

It should also be able to store a redundant or Hamming-coded full catalogue of the planar triangles and the star couples. Also the individual star data must be stored in the dynamic memory, as keeping it in the Flash memory would have prohibitive effects on the real time functioning of the device.

2.4 Camera Unit (SCU)

The Star Tracker Camera Unit is the section that is in charge of obtaining images of the star field and sending them to the SPU. The camera unit is a separated daughter board with the imager and the optics.

2.4.1 Imager

The imager or sensor, is the device located on the focal plane, which translates photons into electrons, and then into a codified signal that contains the raw image in grayscale.

The imager can be CCD or CMOS. **CCDs** (which stand for Charged Coupled Device) are composed by an array of photosensitive coupled capacitors that transport analog signals and are controlled by a clock.

There is a photoactive region and a transmission region. The image is projected by the optics onto the photoactive region, making each capacitor accumulate a charge proportional to the intensity of light that impinges it. Once the array has been exposed to light, a control circuit makes each capacitor transfer its charge to its neighbor. The last capacitor puts its charge onto a charge amplifier, which converts the charge into voltage. Through a repetition of this process, the circuit converts

the charges into a sequence of voltages which it samples, converts to digital data and then stores into a register. [12]

The CCD is commonly fabricated in three architectures: full frame, frame transfer and interline; which address differently the problem of shuttering.

A full-frame device has a full photoactive area and has no shutter. A mechanical shutter must be added or the image will smear as the image is being read.

In a frame transfer CCD, half of the silicon area is covered by an opaque mask. The image can be transferred quickly from the image area to the storage area. This image can be read slowly from the opaque area while another image is being projected onto the photoactive zone. This architecture requires twice the silicon of a full-frame CCD and it costs twice as much.

The interline architecture masks every-other column of the imager with an opaque mask, so it only takes a one-pixel move for a pixel to reach the storage and readout region.

Generally, applications that require a high level of light capture, should use full-frame CCDs, and those that require a relatively low cost and low power consumption, should use interline CCDs.

CMOS sensors, on the other hand, are more flexible regarding the reading process, as every pixel can be individually read. This allows in some systems to have a "windowing" capability useful for target tracking.

CMOS imagers can have a passive pixel or an active pixel configuration. Passive pixels are the oldest ones. They generate a high amount of background noise, so active pixel sensors were designed to reduce this defect. This is accomplished by means of an active circuitry that determines and cancels the noise levels at each pixel. Active Pixel Sensors (APSs) allow to have a larger pixel array and have higher resolution, while still consuming far less power than CCDs.

Current APS technologies bring them almost up to par to CCD sensors. The balance between performance and energy consumption (the latter being critical for the `Aramis` micro satellite) make APS the sensor of choice for our star tracker.

2.4.2 Optics assembly

The optics section has the collection of light as a main function. A lens or a pinhole can potentially be used in an optic system.

The pinhole

A pinhole is a small hole on a thin surface that allows light through this single point and produces an image on the sensor, located in a dark box. The smaller the hole, the sharper the image will be, but longer exposure times will be required.

If the perforation is large, the diverging set of rays causes the blurred image, as each infinitesimal point of the star field will be projected as a circular patch of light on the sensor. If the hole becomes smaller, though, the patch becomes small, but the effect of diffraction increases, and a blurred picture is obtained again. Three effects must be balanced, the amount of light that is allowed onto the imager, the divergence of the rays, and the diffraction. An optimum hole size is that for which the divergence and diffraction have equal influence on the resulting image. This happens when:

$$d = \sqrt{2l \cdot f} \tag{2.1}$$

where d is the diameter of the hole, l is the wavelength of the light and f is the focal length or distance from the pinhole to the imager plane.

However, the use of a pinhole means that there will be a dramatically high integration time. With a commercial camera from the ground, integration times for a night sky picture where stars appear clear, can range from 5 up to 30 seconds [13]. It is licit to believe that a pinhole, having an aperture hundreds of times smaller than one of these cameras, will make it take too long to integrate an image.

Single lens

A single lens is a piece of refractive material that has two opposite surfaces, normally both curved, or one curved and the other one, a plane. because of the curvature of the surfaces, the different rays of an incoming beam are refracted at different angles in a way that they converge on a single point. This point is called the focal point of the lens. Every lens has a focal length, which is the distance from the center of the lens to the point in which the image is formed. A lens with a long focal length, forms a larger image and have a narrow FOV, whereas a lens with a short focal length will form a smaller image and have a wider FOV.

Images, in the case of the SCU, are real (as opposed to virtual images that can be only seen through the lens) and project onto the sensor which is located on the focal plane.

The focal plane for a single lens, is actually not a flat plane. The so-called curvature field must be taken into account [14]. This optical aberration can be better observed in figure 2.7. Object PQ generates a curved image P'Q'. This is caused by the outer ray having a nearer focus point, when compared to the inner ray. This makes the rays that go through the center of the lens, intersect the foci in this way. If all the possible rays are projected and the plane is constructed between P and Q, curvature can be observed.

An example of the distortion generated by field curvature on stars that are projected close to the border of an imager can be seen in figure 2.8. Distortion in these stars is significant, and can lead to a misinterpretation of their real brightness

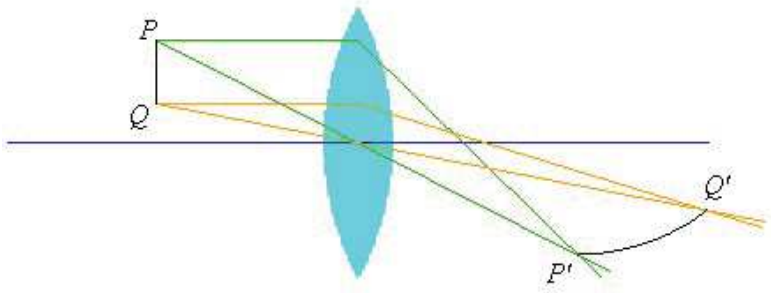


Figure 2.7: Field Curvature. Source: University of British Columbia

value, as a single star would appear smaller, and therefore dimmer, in the center of the image than in the zones closer to the border.

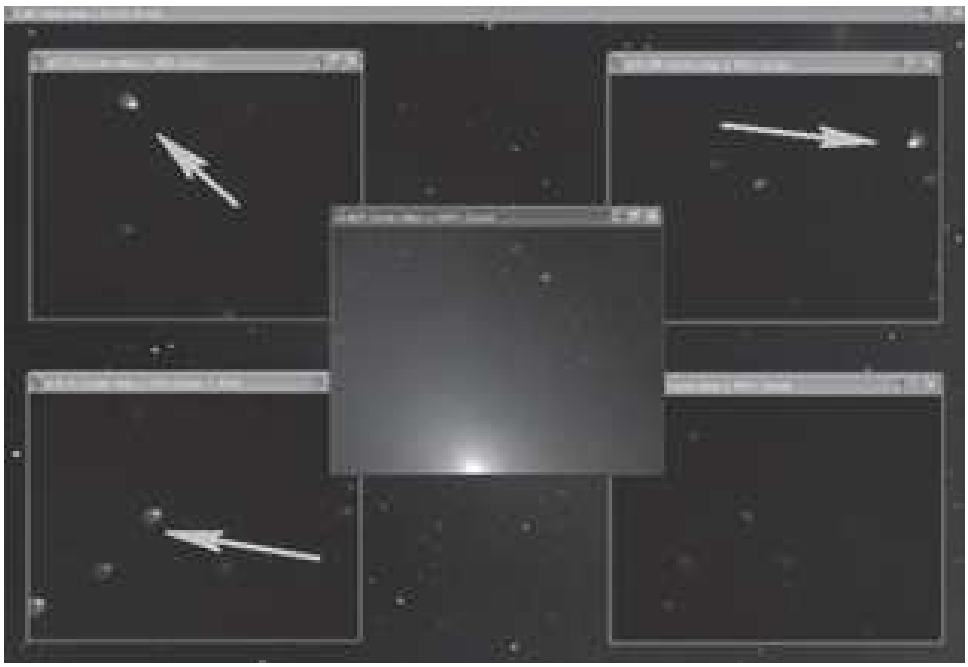


Figure 2.8: Field Curvature effect on a Star Field. Side images: wide Field of View, center image: Detail of a distorted star.

The aperture of a lens is an opening through which light is admitted. The greater it is, the larger the lens is, and the better are its capability of collecting light. The downside of having a large aperture is the resulting mass, since the aperture is a physical lens, as opposed to the pinhole solution. For a Star tracker, the normal approach is to maximize the aperture while maintaining a certain mass limitation.

Single lenses are the simplest form of lenses (besides a pinhole) and therefore, are lighter than the lens that will be explained next.

Double Gauss Lens

The Double Gauss lens is a set of two positive meniscus lenses with two negative meniscus lenses in the middle. This configuration reduces optical aberrations, such as chromatic aberration or, the one that interests us the most, field curvature [15]. An illustration of this type of lens can be seen in figure 2.9, the focal plane lies effectively on a flat surface, that will correspond to that of the imager, and the projected image will present no distortion.

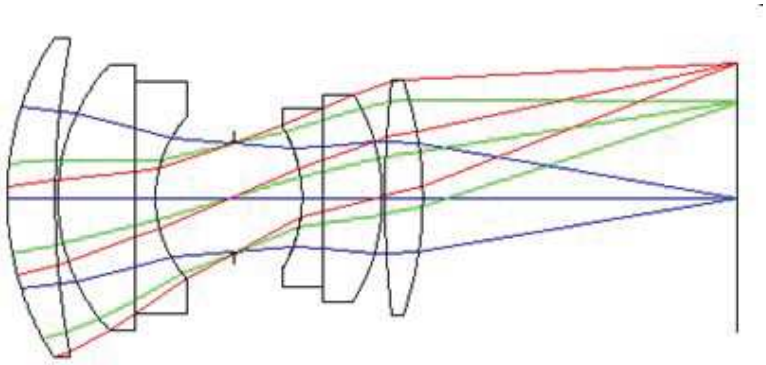


Figure 2.9: Double Gauss lens. Source: Zemax software

The use of this type of lens comes with two drawbacks, one is the increased cost of the device, and the other is the greater mass of this type of lenses.

2.5 Components selection and Preliminary Schematics

The selection of the parts takes into account radiation data publicly available to avoid SEE, availability from known part resellers (RS components, Farnell), ease of integration with the other components, quality of the documentation and software available for the part from the hardware vendor and from third parties.

As the requirements for the SPU have changed, and may change again dramatically during the development of the project by the incorporation of new techniques, or after the cross compilation of the software for the DSP; the schematic capture and components selection of the SPU is out of the scope of this thesis. However, the latest iteration of its design is shown in the appendix. This is the last proposal before

it was decided not to continue its development and it does not fulfill the memory requirements for the Star Tracker, but it is shown for information purposes.

A block scheme of this first design can be seen in figure: 2.10. However, in this chapter, critical components that can be useful for the SPU wil be chosen, namely the microprocessor, the sensor and the optics.

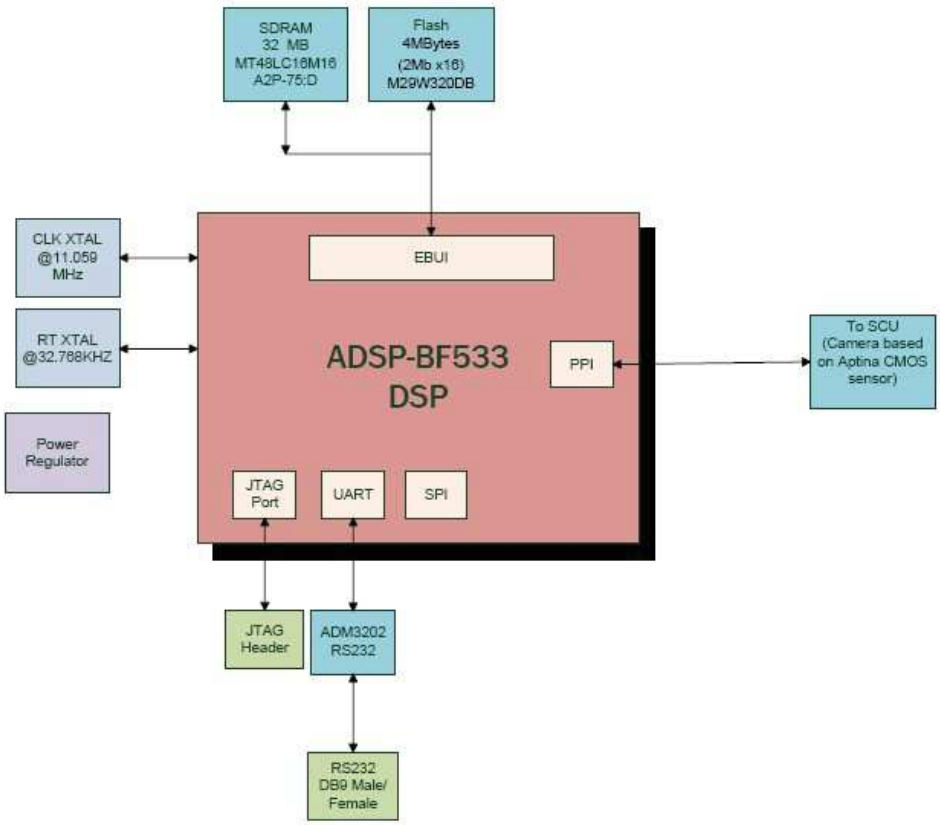


Figure 2.10: `Aramis` Star Tracker HW Block Scheme

2.5.1 Star Tracker Processing Unit

The microprocessor will be an analog devices Blackfin DSP. It was found being used in micro star trackers such as the FAR-MST and LIST from Aeroastro (VA, USA), in the payload of NASA's New Millenium ST8 Project, and in the Payload of Politecnico di Torino's PicPot, a camera that was to take pictures of the Earth from LEO.

The ADSPBF533 is a powerful DSP, capable of performing intensive image processing while maintaining power efficiency. In the case the drawn power should

be too high for the energy efficiency of the satellite, a downgrade to the fully pin-compatible ADSPBF532 with lower consumption can be made with relative ease. [16]

In tests performed by JPL to characterize SELs in the BF533 for the payload of the ST8, the BF533 presented a LET threshold of $\sim 15MeV \cdot cm2/mg$, a saturated cross-section of $\sim 1E-06\ cm^2$ and a SEL probability of 0.008% for a 6 month mission and 14 hours of operation in polar LEO. [17] Radiation in a polar LEO orbit is known for its higher radiation conditions, compared with other LEO orbits. Current plans for `Aramis` don't include a launch in polar orbit, this leaves a broad safety margin for operation, also considering the non-continuous operation of the device, as it will only be used for precise pointing of special instrumentation.

2.5.2 Star Tracker Camera Unit

Sensor

For the reasons stated in section 2.4.1, the imager will be an Active Pixel Sensor (CMOS), Aptina (previously Micron) MT9M001C12STM Monochrome sensor. This is a 1280H x 1024V digital image sensor. This family of sensors offers the advantage of having a straightforward method of interfacing the sensor and the processor through its Parallel Peripheral Interface without the need of any extra "glue logic" [1], reducing complexity and SEE probabilities.

Optics

The only easily available Double Gauss lenses are TECHSPEC lenses sold by Edmund Optics. They are manually focusable and their focus should be set to *infinity*. They can be installed on C-MOUNT standard interfaces, so the manufacturing of a C-MOUNT dark case to be installed on the PCB is necessary.

2.6 Comparison with State of the Art

As the requirements stated, the Star Tracker needs to have a convenient price for a University project; a low mass, adequate for a satellite with the characteristics of `Aramis`; and finally, a low power consumption.

These requirements can be achieved by using COTS components with moderated consumption and with standby capabilities. Some of them have been introduced in section 2.5; a preliminary schematic, located in the appendix, has been developed. It is possible now to produce and a raw estimate of the maximum consumption, the mass and the price of the satellite. These values can be better put in context with the state of the art in figures 2.14, 2.15 and 2.16. It is important to note that

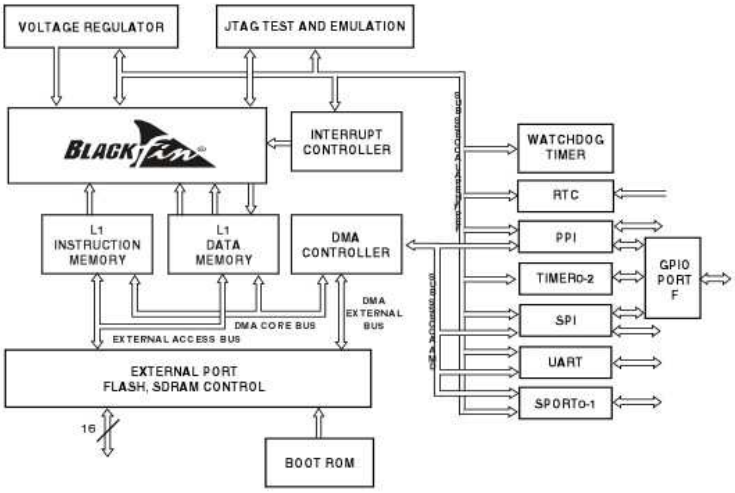


Figure 2.11: ADSPBF533 Functional Diagram. Analog Devices.

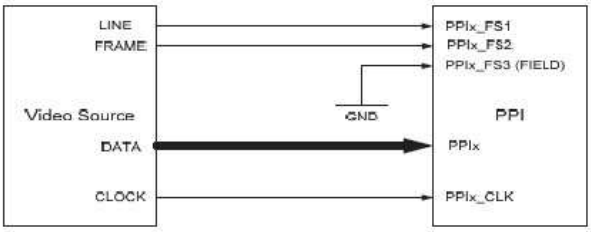


Figure 2.12: Aptina sensor-dsp interface. On the left, imager video source. On the right, PPI of Blackfin. Source: Analog Devices [1]

the manpower cost is not taken into account for the `Aramis` systems, being it an educational project; and also that these figures consider masses and consumption of the memory devices too, devices that are surely subject to change, however, they should provide a good approximation of what the project is seeking. Data obtained and calculated from: [18, 16, 19, 20].

2.7 Star Tracker mounting

The optics should be covered by a light shade assembly, in order to keep the light that is not in the FOV from entering the optics and filling it with light that would invalidate the image. Possible parasite light may include that of the moon, the sun, Earth albedo and reflections from the spacecraft itself. This light shade should have an aperture angle equal or slightly higher than the defined FOV. It shall also be

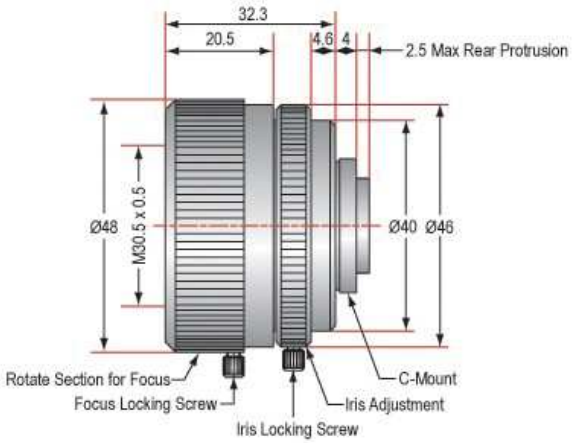


Figure 2.13: TECHSPEC Double Gauss Focusable Lens. [1]

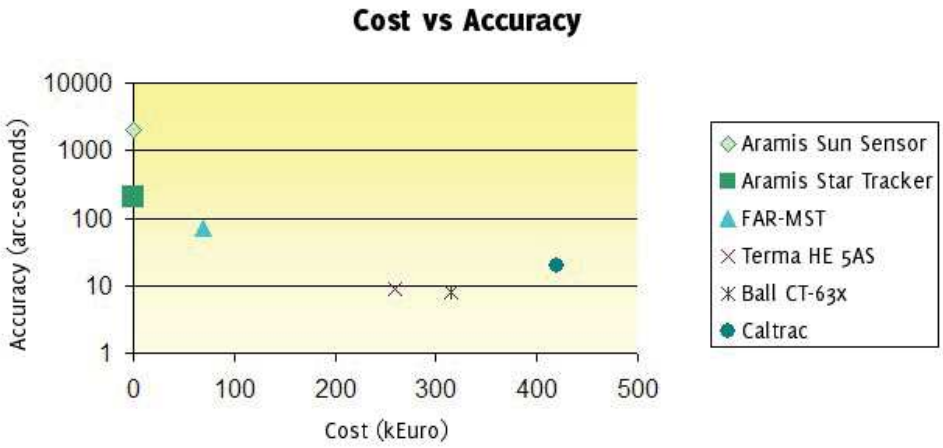


Figure 2.14: Cost versus Accuracy for various Star Trackers. Figures for `Aramis` are estimates.

painted with matte black paint in the inside to minimize reflections.

The `Aramis` satellite operates with solar panels located on the different faces of the cubic modules, therefore an optimum inclination angle for the light shade would be 45 degrees, so no sunlight would enter during the charging of the batteries with the panels of the side the light shade stands on, or with those of the side adjacent to it when the sun light vector is normal to the panel surfaces.

Based on the physical architecture of `Aramis` [21], the setup shown in figures 2.17 and 2.18 is proposed.

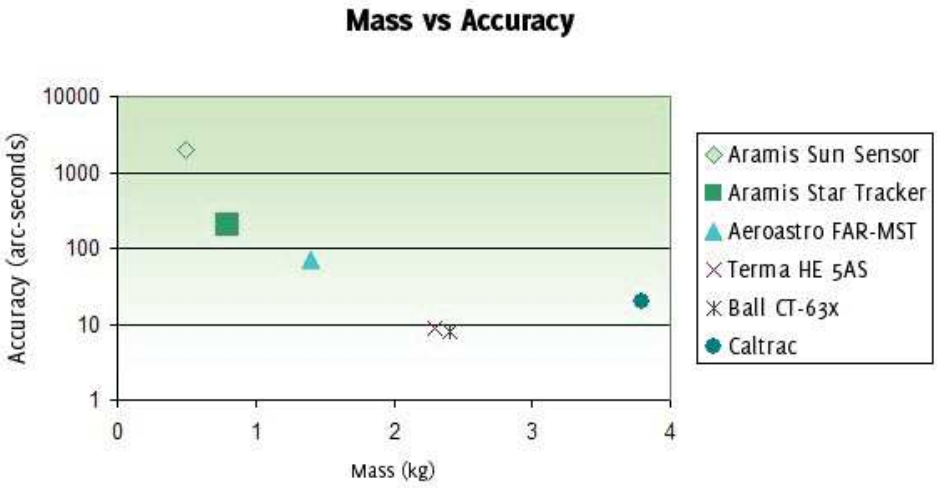


Figure 2.15: Mass versus Accuracy for various Star Trackers. Figures for `Aramis` are estimates.

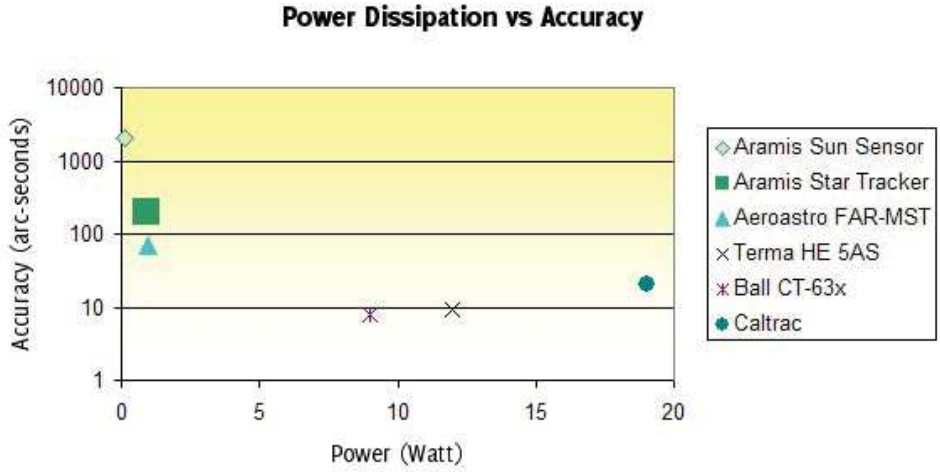


Figure 2.16: Power versus Accuracy for various Star Trackers. Figures for `Aramis` are estimates.

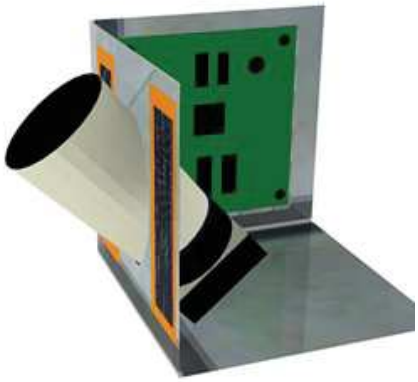


Figure 2.17: Internal setup (solid view)

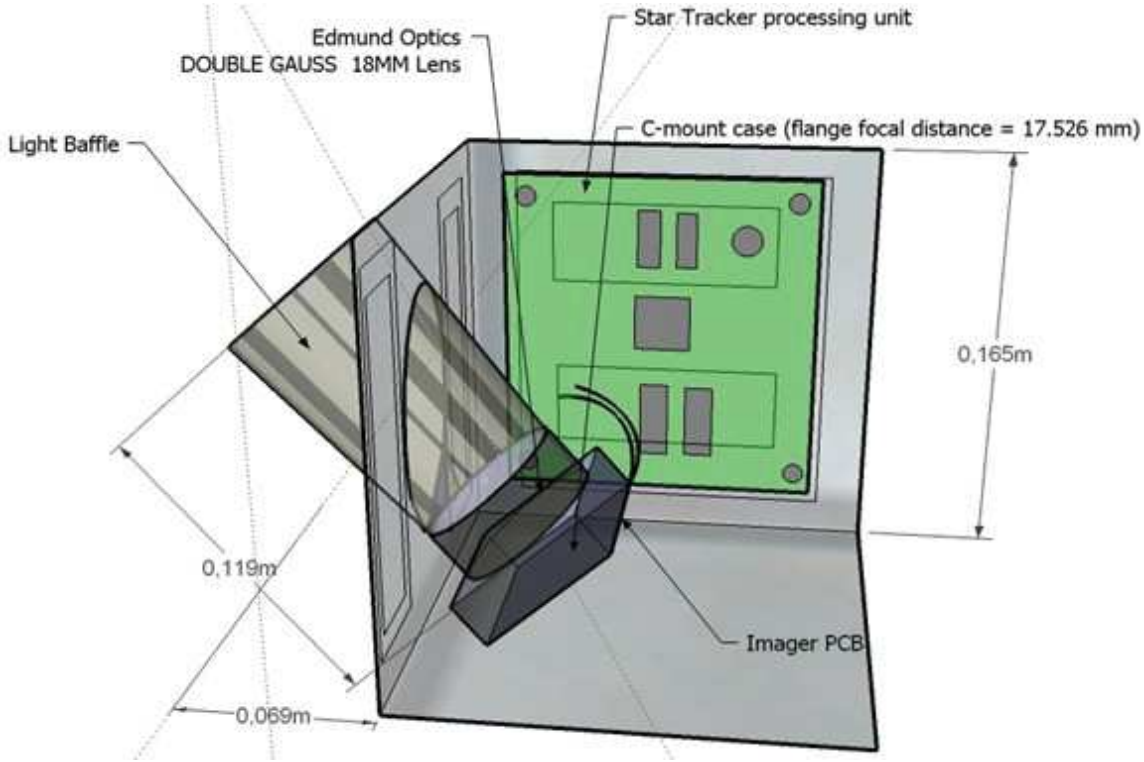


Figure 2.18: Setup (detailed view)

Chapter 3

Image Processing

3.1 Data Flow

Images taken by the SCU will need to be pre-processed in order to be able to do pattern identification. This is the first of the chapters describing the algorithms that will be used on board the Star Tracker.

3.2 Simulation Platform

In order to make and test the algorithms, one must have images that simulate the image generated by the sensor. Not having a directly available system, as appropriate cameras with good optics and the capacity of integrating a signal over a long period of time, were hard to find and expensive to acquire by the author, a new approach was taken with the aid of a planetarium program quite popular among the astronomical community.

Celestia is a 3D astronomy software, based on the Hipparcos Catalogue that permits the user to visualize any moment in any point of the known Universe. Celestia can display a myriad of objects that can go from the biggest known stars to some of the artificial satellites that orbit the earth. The software is based on the OpenGL graphics display library. It is widely by ESA and NASA for educational programs and as a visual interface for trajectory analysis software. No prior uses of Celestia as a Star Tracker simulator were found, so various tests were performed on it before starting its use on this project, namely, tests on visualized patterns were compared with patterns characterized directly from the accurate catalog. The pattern criteria used was that of planar triangles, which is explained in section 5.5.

Celestia allows the use of scripts written in Lua language. A tutorial on the basics of this language can be found in [22] and [23]. Celestia has two different scripting languages: celestia-scripts (.cel extension), which provides less control capabilities,

and the new Lua-scripting (suffix: .celx), a complete programming language that, as stated before, is based on Lua and allows for more interaction with the system. This allows the grabbing of screenshots and the possibility of writing files.

With this software, it is possible to visualize or not, atmospheres, clouds, constellations, orbits, planets, among many other factors or objects. For our purposes, on the visualization options menu, everything should be turned off, and allow only for visualization of stars and planets. In some cases it is useful to have the celestial grid **on** to visualize and have a sense of orientation, but not for frame capturing purposes.

Star signals, as we will soon see, are approximatively a 2D gaussian curve, so the unfocused point option should be chosen in the star visualization menu. The simulator must be very close to the Earth's position in order to avoid undesirable parallax effects on stars.

A series of scripts were developed in order to create the simulation. These scripts usually establish a FOV angle, visualization parameters.

The chosen field of view was based on the default size of the window of celestia. It is extremely important to note that the same FOV must be used in all the algorithms, including the Star Catalogue generation and the pattern recognition algorithm. This parameter will change when the optics are completely defined.

An example of the scripts used to scan parts of the celestial sphere is:

```
{
unmarkall{}
lookback{}
setvisibilitylimit {magnitude 6.0}
set{name "FOV" value 21.534638888888890}

select { object "HIP 33316" }
center {  }
wait {duration 5}
rotate {
        duration 3
        rate 10
        axis [0 1 0]
}

rotate {
        duration 3
        rate 10
        axis [1 0 0]
}
```

```
rotate {
        duration 3
        rate 10
        axis [0 0 1]
}
}
```

This is a simple .cel script, it sets the maximum visible star to magnitude 6.0, the field of view to 21.53 degrees (our chosen FOV). it centers the view on the Hipparcos object 33316 and waits for 5 seconds then rotates for 3 seconds around the y axis of the local frame of reference at 10 degrees per second, afterwards it rotates for 3 seconds at 10 degrees per second about the x local axis, and finally it rotates for 3 seconds at 10 degrees per second around the z local axis.

.celx scripts are more complex. The script used to get screenshots, that can be run at the same time as the .cel script that makes the rotations, obtains attitude values such as euler angles, attitude quaternion and RA/DEC values (all of them are explained in chapter 4 The .celx code used is available in the code appendix.

Other tools such as Microsoft Worldwide Telescope were used in order to manually identify stars and their location when no complete catalogue information was available in Celestia.

3.3 The Star Signal

The signal coming from a star and impinging the imager plane will be assumed to be a gaussian function. This is just an approximation, as there are refraction effects that shape the signal in a slightly different way: the light that comes from the stars and arrives to the focal plane is actually an Airy's function. The center of the Airy's function, which is commonly known as the Airy's Disk corresponds to about 86% of the photons that compose the star.

The Airy's function, up to the first zero is approximated by the Gaussian Function. The two dimensional Gaussian function corresponds to:

$$g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left[\frac{(x-\mu_x)^2}{2\sigma_x^2} + \frac{(y-\mu_y)^2}{2\sigma_y^2}\right]} \tag{3.1}$$

where x and y are the position on the plane, μ_i is the mean position of the centroid and σ is the standard deviation from the mean.

The entity that interests us in order to measure the position of a star in the imager plane is the stellar signal. The sensor translates the incoming photons into electrons. The formula that describes this translation is:

$$elect = a \cdot FFSR \cdot t_{shutter} \cdot g \cdot 10^{\frac{28-M_v}{2.5}} \tag{3.2}$$

where a is the area of the lens or pinhole, FFSR is the product of the fill factor and the spectral response of the sensor, as given by the manufacturer, $t_{shutter}$ is the time during which the shutter is open (either mecanically or electronically) to let the star photons convert into electrons, g is the gain, or the number of electrons equivalent to an analog-to-digital unit (ADU), finally M_v is the visual magnitude of the star.

The percentage the sensor area dedicated to actually collecting photons is called fill factor. CCDs have the advantage of a 100% fill factor but CMOS sensors, such as the one to be used on the `Aramis` Star Tracker have much less than that [24]. The spectral response is the mean value of imager responsiveness to the differents part of the spectrum, and is given in A/W units.

3.4 Noise

The image acquired by a real sensor based on current technologies is always affected by noise to some degree. There are different types of noise affecting the image. The most relevant in-pixel noises are dark current, signal shot and background noise, there are also other noises due to the AD conversion, such as the readout and quantization noises. This section presents the different types of noises that can be found in an electronic imager.

3.4.1 Signal shot noise

Signal shot noise is a process that is due to the fact that light comes in quanta and when the light hits the imager, due to the random nature of light, at a certain moment the number of photons that hits it may be a determinate amount, and in the next moment, that same number of photons from the same source may vary. The photon arrival is a Poisson process, and as such, the expected value is equal to the variance, and the standard deviation is:

$$\sigma_{shot} = \sqrt{eNum} \tag{3.3}$$

where eNum is the number of electrons of a certain star.

3.4.2 Dark current noise

Dark current is a comparatively small current that flows trough the sensor when no light is being impinged on it. It is due to the aleatory generation of electrons and

holes in the depletion region of the semiconductor. Dark current may be reduced if the sensor is physically in contact with a passive or active cooling device. The standard deviation of dark noise is:

$$\sigma_{dark} = \sqrt{d \cdot t} \tag{3.4}$$

where d is the dark current rate in $[\frac{e}{s}]$ and t is the period of time in which the dark current electrons are being generated.

3.4.3 Background noise

Background noise is a sum of all the other possible sources of noise. This can be the earth's albedo, the reflected light from the sun on a not perfectly black-painted sun shade, etcetera. This is not a value easy to model, and all the possible measures should be taken to reduce its influence to a value under that of the boundary between the noise and the performance requirements.

3.4.4 Quantization noise

Quantization noise is due to the fact that an Analog to Digital conversion is quantized, that is, the values are not defined all over the operating dynamic, but there will be ranges of photons quantities to which a common value will be assigned. This is because the discrete nature of the digital signal.

For example, to a range between 0 and 99 photons, a value of 50 electrons will be assigned, whereas any value between 100 and 199 produced will have a discrete value of 150 electrons.

Quantization can be modeled as a stochastic process with uniform distribution, where all values between two boundaries have the same likelihood of occurrence, and the distribution has a zero mean. Therefore, the variance of this type of noise is defined by the expected value of the mean-squared error, so the standard deviation is [25]:

$$\sigma_{quantization} = \frac{\Delta}{\sqrt{12}} \tag{3.5}$$

being Δ the gain associated to the sensor:

$$\Delta = \frac{\text{fullWell}}{\text{ADU}_{\text{max}}} \tag{3.6}$$

Where $fullWell$ is the maximum number of electrons that each pixel is able to hold, and ADU_{max} the maximum number that the analog-digital converter can represent (2 to the power of the number of bits), in the case in which the signal conditioning does convert fullWell into the voltage value ADC_{max}.

3.4.5 Readout noise

Readout noise is the effect of the not exact conversion from photons to electrons to the readout of the imager. The amplifier is not an ideal one, so it typically gives out the correct value on average, but presents aleatory scatter. Readout noise is a quantification of this scatter.

Its standard deviation is defined by:

$$\sigma = \frac{R}{g} \tag{3.7}$$

Where R are the electrons produced by the readout noise, and g is the gain in electrons per ADU. It is interesting to note that the readout error decreases with higher gain, as opposed to the quantization noise, that does the contrary.

Readout and quantization images are values on which we can not have influence, they are determined by the fabrication processes and are intrinsic to a specific sensor.

3.4.6 Total Noise

The noise that can be found at the output is the result of the different types of noise previously described.

The variance is the the mean difference between the squares of the real value and the measure. When we have large sample sizes, the variance of each variable gets closer to the statistical variance, and cross-correlations approach zero, therefore, for large sample sizes and for Poisson and Gaussian distributions, the variance of the noise electrons is:

$$\sigma_N^2 = \sigma_S^2 + \sigma_D^2 + \sigma_R^2 + \sigma_Q^2 + \sigma_B^2 \tag{3.8}$$

Where sigmas S,D,R,Q and B represent the variance of each type of noise, namely, signal, dark current, readout, quantization and background noises.

3.4.7 Signal-to-noise ratio

Signal to noise ratio is the value that tells us the quality of a signal. It is defined as the ratio between the signal and the background noise.

$$SNR = \frac{Signal}{\sqrt{\sigma_S^2 + \sigma_D^2 + \sigma_R^2 + \sigma_Q^2 + \sigma_B^2}} \tag{3.9}$$

where Signal is the number of electrons that should ideally be produced by the impinging photons. As we will see, it is important to keep a high SNR, so the noise

should be kept down. In cases when we have dim stars in the FOV, these may not present a stronger signal than the noise.

The noises we will take into account will be signal shot, dark current and readout noises. The other sources of noise are not as relevant comparatively.

3.5 Thresholding

The average noise is generally lower than the star signal. In order to separate the noise from the real image, it is possible to determine a threshold and eliminate every signal that is below this level. A typical star with noise is depicted in figure 3.1.

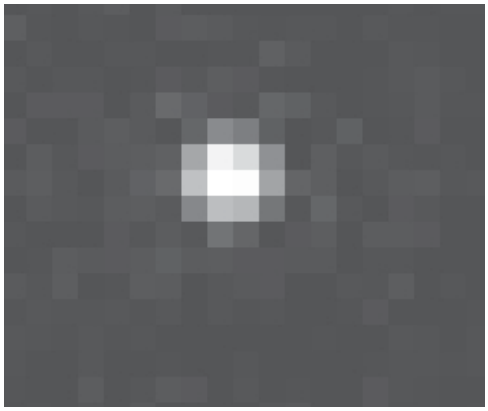


Figure 3.1: Original noisy star

One appropriate threshold could be the average level of the whole picture, since pixels containing stars are a very small part of the whole pixel array, even if they are included in an averaging process, the result will give a good evaluation of the black background level.

This background level can vary over time, as a rotation takes place, for example. This may be due to the presence of nebulae in the background, which rises the average value. The approach used to solve this is to take the last 5 frames and make an average out of them, instead of using one single frame. In the simulation, this algorithm is implemented, but nebulae are not activated in Celestia, as its representation is not realistic enough (too high levels of brightness, representation composed by sets of artificial OpenGL circles).

Different thresholds were tested, starting from the average level. This value though, left too much noise in the picture. The one that offered the best results of star identification was a level of 2σ, that is, twice the noise level.

Everything below the threshold was regarded as noise and a binary mask was created to mark these pixels as black, and potentially valid star pixels as white. The

results can be observed in figure 3.2.

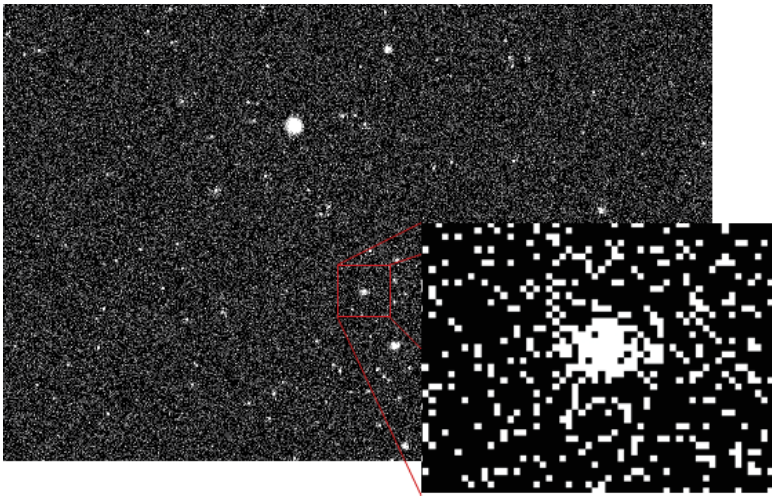


Figure 3.2: Resulting mask after thresholding

The term "potentially valid" is used here because even though noise under twice the average noise level was eliminated, there is still a remaining part of these disturbances, in fact, small sized points that are the peaks of the noise can be found in the picture and in the mask. They are usually quite sparse when compared to the evident star clusters in the image.

This characteristic can be exploited to get rid of these pixels. Two windowing methods proposed by Huffman [25] were considered to clean the image up from them.

The first one consisted in scanning the whole image, in *steps of 1 pixel* with a 5-by-5 window. As the window steps in one new pixel in the center of the window, it reads all the pixels in the window. If at least half of the pixels in the window are potentially valid, the center pixel is deemed valid. This method leaves a mask that reflects more closely the shape of the stars, but has a computational cost.

The second method involved using the 5-by-5 window with *5 pixel steps*. If half of the pixels in any given window are *potentially valid*, all the *potentially valid* pixels in the window are deemed *valid*. With this method, the stars have a more squared shape, but at a reduced computational cost

The time performance of the first method (spkremoval1) when compared to the second one (spkremoval2) was considerably better, and no significant difference was found in the actual spike removal, so the second one was chosen in order to minimize the load on the processor. The results of spkremoval2 can be appreciated in figure 3.3.

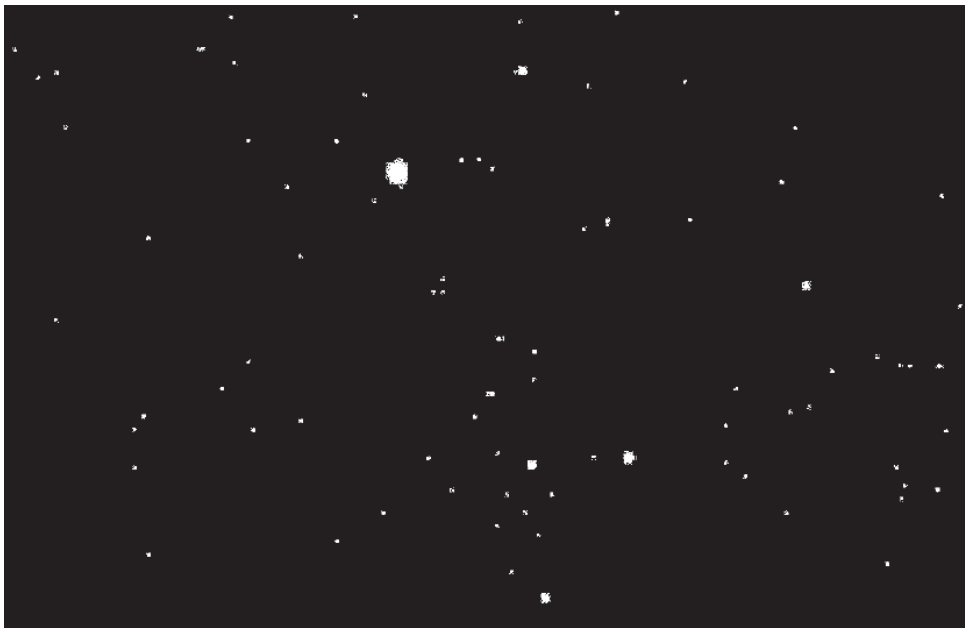


Figure 3.3: Mask after windowing with spkremoval2, showing star pixel clusters

3.6 Centroiding

The `Aramis` Star Tracker performs a weighted sum technique operation to accurately identify the center of the stars. This method determines a virtual center of gravity, where the quantity of photons is considered as the weight value of the pixel.

It should be noted that through this method, it is possible to obtain sub-pixel accuracies, this represents a major improvement on the position determination, if compared with the possibility of measuring the position of a star just by using its geometrical center pixel.

The clusters of pixels belonging to every single star, are marked with the aid of a second matrix where each pixel has a number greater than zero that is unique to the star it belongs to, or zero if it is not a valid pixel.

After the noise spike filtering is complete, a centroid algorithm is run on the valid pixels, and using the grayscale level as the weight it calculates the x-y position on the image plane with equation 3.10

$$pos = \sum_{i=1}^{n} \frac{Ph_i \cdot position}{\sum_{i=1}^{n} Ph} \tag{3.10}$$

where i is the actual pixel, n is the total number of pixels of the star and Ph is the number of photons, or for that matter, the grayscale level of the actual pixel.

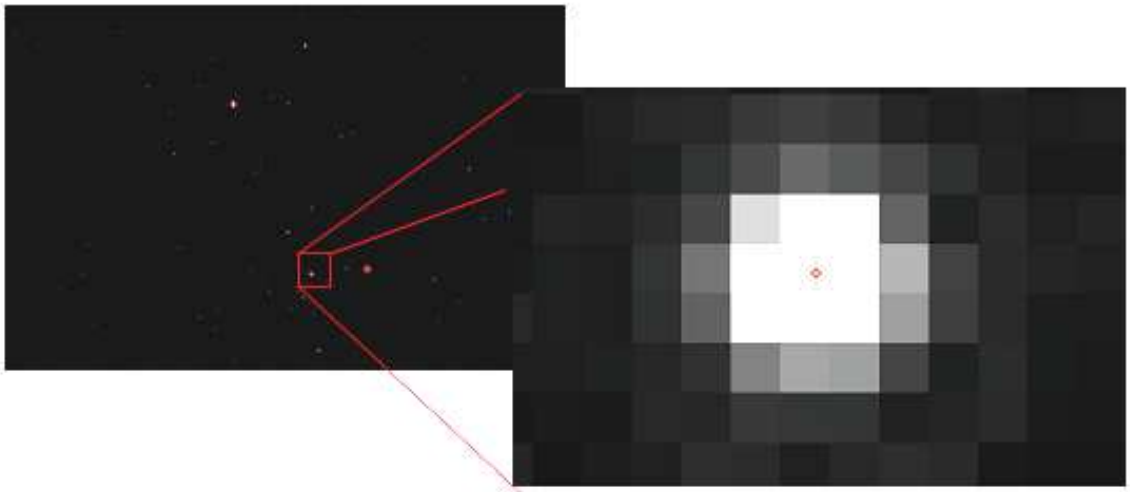


Figure 3.4: Centroiding of three stars with detail of one of the three

Chapter 4

Considerations on the Frames of Reference

A frame of reference is a coordinate system useful to identify the positions of the objects that are in it. The reference frame used for a Star Tracker needs special considerations that will be addressed in this chapter.

4.1 The celestial Sphere

A Star Tracker uses the Celestial Sphere as its main frame of reference. The Celestial Sphere is a virtual sphere of "infinite" radius that rotates around the Earth sharing the same axis, equator and poles with the planet. Every object in space can be considered as being on this imaginary sphere.

Parallax is the difference of the position of an object if viewed through two lines of sight. This can influence more or less the position of objects on the celestial sphere, depending on specific conditions. Nearby elements have a higher parallax, while the farther ones have a lower parallax.

There can be two ways of considering the celestial sphere: geocentrically or topocentrically. When we refer to a geocentrical celestial sphere, we are thinking of a sphere centered on an observer that is in the origin of Earth's coordinate system, so no parallax effects must be taken into account. If we refer to a topocentrical celestial sphere, the observer is on the surface of the earth, and parallax effects should be taken into account, specially to address objects that are not too far away from earth. In our case, the stars are practically on the same position on the celestial sphere, even in LEO, as they are quite far away.

The celestial sphere, just like the Earth, is divided into two hemispheres, and different coordinate systems can be established. The coordinates our catalogue uses are the equatorial coordinate system, which is the most widely utilized. Its

parameters are Right Ascension (RA) and Declination (DEC).

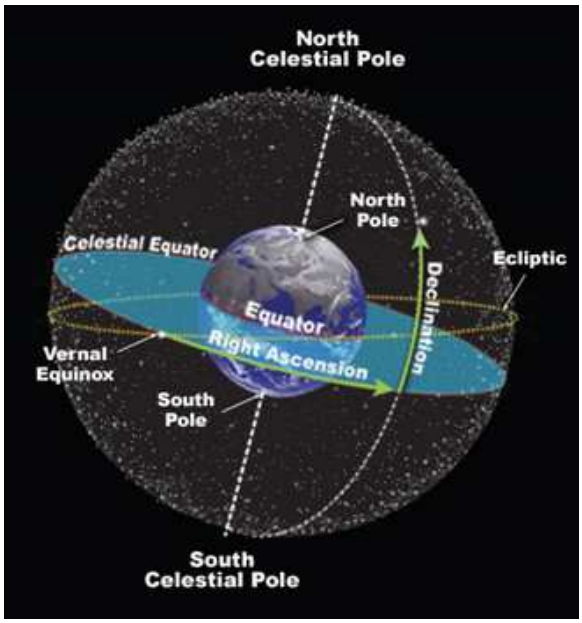


Figure 4.1: Celestial Sphere. Source: NASA

The location on the celestial sphere over an observer is the zenith. An arc passing by the poles of the celestial sphere and by the observer's zenith is the observer's meridian. The nadir is the direction opposite the zenith: for example, straight down from a spacecraft to the center of the planet.

The celestial sphere seems to be rotating around the earth with a period of 23 hours and 56 minutes. This is called the sidereal time, which is faster than the solar time (the period with which the sun appears to do a full rotation around the earth).

4.2 Right Ascension and Declination

Right Ascension is similar to terrestrial longitude, except it is not measured having an origin in a directly earth-referenced point, but on the First Point of Aries, which is the point in the sky where the sun crosses the celestial equator during the Vernal equinox. Right Ascension is measured in hours, minutes and seconds, and it increases going east from the First Point of Aries. RA can be also abbreviated as α.

The full celestial sphere, has 24 hours of RA. This means that each hour corresponds to 15 degrees of arc, each minute of RA to 15 arc seconds and each second of RA, 15 arc minutes.

Declination (Dec or δ) is much like the terrestrial latitude. The unit that is normally used is the degree, with its minutes and arc seconds. Any object that lies

on the celestial equator has a Declination of 0 degrees; if it's on the north pole, the declination is +90 degrees; and finally if it's on the south pole, Dec is -90 degrees.

As an example, we can read the coordinates from our catalogue, for the star Sirius (the brightest star on the sky), whose RA is 6h, 45m and 8.9173 sec and Dec -16 degrees, 42 minutes and 58.017 seconds.

4.3 Precession, nutation and polar motion

Precession is the change in direction of the rotational axis of the Earth with respect to the sky. The orientation of said axis is permanently changing, forming a virtual cone with a half-angle cone of 23.5 degrees in a cycle that lasts about 25765 years. This movement is caused basically due to the gravitational forces that the Moon and the Sun exert on the Earth. Therefore, the poles move in circles over the same time period. Currently, Polaris is the northernmost star, but this will change over time, when other stars will find a position as the "northern star".

Nutation is an semi-regular motion in the axis of planet Earth. It happens because of the tidal forces that cause precession change in time in a way that makes precession inconstant. It consists in a small nodding motion with a period of 18.5 years and amplitude of 9.5 arc seconds. This is a very small change that we won't consider.

Other effects are the changes in the Earth's pole orientation due to motions in the Earth's core and mantle, and changes in the distribution of water. There are two movements associated with this, the Chandler Wobble, and a drift. They are much smaller and almost irrelevant for our purposes.

4.4 Epochs

As the position of the stars change, albeit in a very slow manner, there is a need to establish a fixed moment at which the coordinates of the celestial bodies are specified, and which will be approximately valid for a period of time.

With time, imprecisions accumulate, and this causes errors in ephemeris prediction, so it is necessary for astronomers to recalculate the values. This is when an Epoch is defined in international agreements among astronomers.

The current epoch is called J2000,which refers to the RA and DEC of an object on January the 1st, 2000 12 Universal Time. This is what is called a Julian Epoch, that is because it is exactly 100 x 365.25 days since the previous standard epoch J1900 (January 0 12UT). J2000 is the epoch that the catalogue of the `Aramis` Star Tracker uses.

4.5 Catalogue

A list of the stars with their location is necessary in order to produce the attitude measurement given a set of identified stars on the sensor.

4.5.1 The Yale Bright Star Catalogue

The Yale Bright Star Catalogue, is a star catalogue that lists all the stars with magnitude lower (brighter) than 6.5. This is basically every star on the celestial sphere visible to the naked human eye. The magnitudes that a sensor can detect are generally those brighter than Mv=5 (this obviously depends largely on the sensor's properties which will be discussed later).

The YBS catalogue is broadly used as a source of astronomical data. It contains a compilation of many stars contained in other catalogues, double and multiple star identifications, indication of variability, equatorial positions for the B1900.0 and J2000 epochs, galactic coordinates, photoelectric photometric data when they exist, spectral types, proper motions, parallax, radial and rotational velocity, among other useful data.

The YBSC contains 9110 objects of which 9096 are stars. 14 objects in the original compilation of 1908 were actually novae or extragalactic objects preserved to maintain the numbering, and they should be excluded in the optimization process.

The catalogue, that is available online for download [26], contains the variables listed in figure 4.2.

4.5.2 Catalogue Optimization

One major constraint when assessing the hardware of the Star Tracker is the size of star database. If we were to use a whole catalogue of stars, it would occupy more than a hundred Megabytes, only for the pattern database [27]. Since the database is not only made up of the list of stars, but also of the patterns, we have therefore to optimize the catalogue to handle it better. Besides, a great quantity of these stars can't be seen by the sensor.

First, only use of brightness and location (RA and DEC) data is made. Furthermore, stars that won't be used are removed from the catalogue.

In order to read the catalogue, a small C program was written so that MATLAB could interpret the YBSC with its native reading routines.

Then, a MATLAB script was developed, which first filters the stars to use those with brightness higher than a certain magnitude.

The 9110 stars and objects from the bright star catalogue are plotted and shown in figure 4.3.

1- 4	I4	---	HR	[1/9110]+ Harvard Revised Number = Bright Star Number
5- 14	A10	---	Name	Name, generally Bayer and/or Flamsteed name
15- 25	A11	---	DM	Durchmusterung Identification (zone in bytes 17-19)
26- 31	I6	---	HD	[1/225300]? Henry Draper Catalog Number
32- 37	I6	---	SAO	[1/258997]? SAO Catalog Number
38- 41	I4	---	FK5	? FK5 star Number
42	A1	---	IRflag	[I] I if infrared source
43	A1	---	r_IRflag	*[':] Coded reference for infrared source
44	A1	---	Multiple	*[AWDIRS] Double or multiple-star code
45- 49	A5	---	ADS	Aitken's Double Star Catalog (ADS) designation
50- 51	A2	---	ADScomp	ADS number components
52- 60	A9	---	VarID	Variable star identification
61- 62	I2	h	RAh1900	?Hours RA, equinox B1900, epoch 1900.0 (1)
63- 64	I2	min	RAm1900	?Minutes RA, equinox B1900, epoch 1900.0 (1)
65- 68	F4.1	s	RAs1900	?Seconds RA, equinox B1900, epoch 1900.0 (1)
69	A1	---	DE-1900	?Sign Dec, equinox B1900, epoch 1900.0 (1)
70- 71	I2	deg	DEd1900	?Degrees Dec, equinox B1900, epoch 1900.0 (1)
72- 73	I2	arcmin	DEm1900	?Minutes Dec, equinox B1900, epoch 1900.0 (1)
74- 75	I2	arcsec	DEs1900	?Seconds Dec, equinox B1900, epoch 1900.0 (1)
76- 77	I2	h	RAh	?Hours RA, equinox J2000, epoch 2000.0 (1)
78- 79	I2	min	RAm	?Minutes RA, equinox J2000, epoch 2000.0 (1)
80- 83	F4.1	s	RAs	?Seconds RA, equinox J2000, epoch 2000.0 (1)
84	A1	---	DE-	?Sign Dec, equinox J2000, epoch 2000.0 (1)
85- 86	I2	deg	DEd	?Degrees Dec, equinox J2000, epoch 2000.0 (1)
87- 88	I2	arcmin	DEm	?Minutes Dec, equinox J2000, epoch 2000.0 (1)
89- 90	I2	arcsec	DEs	?Seconds Dec, equinox J2000, epoch 2000.0 (1)
91- 96	F6.2	deg	GLON	?Galactic longitude (1)
97-102	F6.2	deg	GLAT	?Galactic latitude (1)
03-107	F5.2	mag	Vmag	?Visual magnitude (1)

Figure 4.2: Information provided by the YBSC

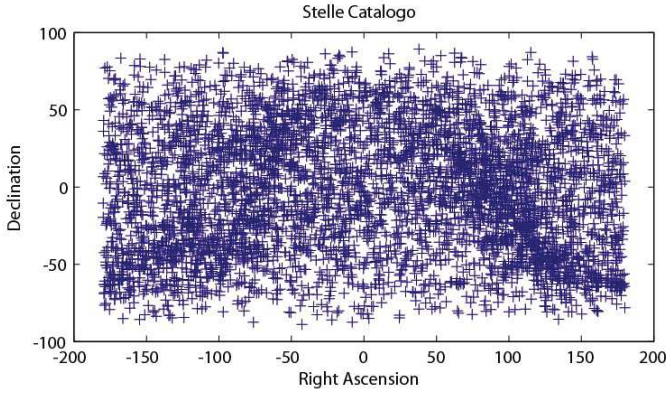


Figure 4.3: Full set of stars from the Yale Bright Star Catalogue

Then the filtering procedure occurs, and the whole celestial sphere is scanned with a window with the dimension of the FOV (entered by the user), and with intervals of one degree. First ordering stars in function of their magnitude and leaving the 10 brightest ones for each iteration. Also the objects that are not regarded as

stars are conveniently filtered away from the catalogue as they are lacking any useful data on the catalogue.

The result of this filtering can be seen in figure 4.4. It produces 1312 stars for 5 stars per FOV and a FOV of 21^o 32" 4.7'. This FOV was established for testing on the Celestia simulator, only for practical reasons, as it is the FOV of the default window.

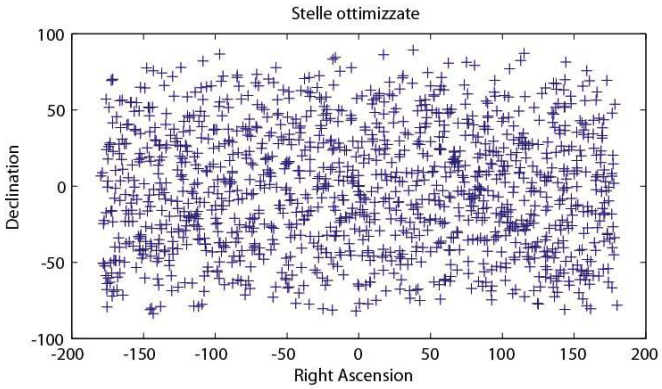


Figure 4.4: Filtered stars, leaving at least 6 for any given FOV

The position of an object in the celestial sphere is represented by a unit vector pointing from the origin (this can be the star tracker frame body or the earth) to a point in the celestial sphere.

These vectors can be obtained from the image with the following relationships:

$$vecpos = (comp_x, comp_y, comp_z) \tag{4.1}$$

$$\begin{aligned} comp_x &= x\frac{pp_x}{f}\left[1+\left(x\frac{pp_x}{f}\right)^2+\left(y\frac{pp_y}{f}\right)^2\right]^{-\frac{1}{2}} \\ comp_y &= y\frac{pp_y}{f}\left[1+\left(x\frac{pp_x}{f}\right)^2+\left(y\frac{pp_y}{f}\right)^2\right]^{-\frac{1}{2}} \\ comp_z &= \left[1+\left(x\frac{pp_x}{f}\right)^2+\left(y\frac{pp_y}{f}\right)^2\right]^{-\frac{1}{2}} \end{aligned}$$

Where pp_x and pp_y are the x and y pixel pitches of the sensor. x and y are the coordinates in the imager plane. Frequently, pixels are square so most of the time $pp_x = pp_y$. f is the focal length of the lens.

Then, screenshots from stars were taken from Celestia, an open source planetary simulator based on the ESA Hipparcos Catalogue which displays star data (among data from many other bodies) using OpenGL.

The stars Wezen δCMa , Adhara δCMa and Sirius δCMa (the brightest star in the celestial sphere), all from the Canis Major constellation, were selected, and the correspondence of the pattern seen in Celestia and the one seen calculated from the catalogue were proven right, although other sets of stars, with different brightness values, in different positions might not give the same successful result. For more information on the pattern matching technique, see section 5.5.

Chapter 5

Pattern Recognition

Several different algorithms have been proposed to assess the attitude of the spacecraft based on image readings. Their difference is determined by the quantity of stars analyzed, the regime in which they operate, the load they put onto the processor and their actual effectiveness.

Once an image has been acquired and the positions of the stars have been identified in the imager plane, equations and are used to obtain an unitary vector for each of the chosen stars, for all the following algorithms, grid pattern matching is an exception. Afterwards, an algorithm to identify the pattern of stars is applied. Among many other algorithms it is possible to find the following:

5.1 Single Star Matching

Acquiring an image and individually identifying a star could theoretically be possible by using visual magnitude, data that we have readily available from the YBSC. The difficulty in this case would be achieving an almost perfect noise rejection, since stars with a similar visual magnitude are likely to appear with exactly the same magnitude on the data provided by the sensor; added noise could be enough to modify it. Only in cases where the star brightness exceeds magnitude 0 or 1, we can have a high probability of getting a good result. This is however not an option; the chances of having such a bright star in a FOV of even a high value like 30 degrees are quite low, as there are few stars with that brightness.

The response for an individual star in different sensors varies widely, and different stars with the same magnitude have different spectral components, so even if the sensor gets the same apparent magnitude for two stars, there is the possibility of them not having the same magnitude because one could be considered brighter than it actually is if the sensor has a better response at its wavelength; being able to discern that would add much complexity. Moreover, the sensor response changes

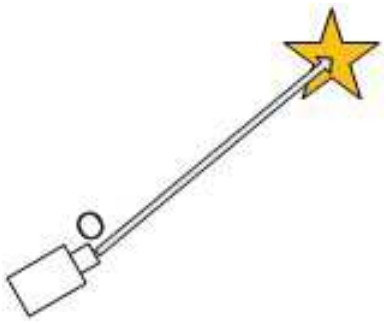


Figure 5.1: Single Star identification

over time, so we might not get the same response for a given star at the beginning of the mission, or 6 months later.

It has been demonstrated in tests [25] that, for a given star, mean distance between the sensed value and the actual Mv increases when the brightness decreases, and so does the standard deviation of said difference. For Mv values of 4 and dimmer, Mv data can not be obtained if the noise floor is not dropped below 5σ, that is 99.99994% of the times, the signal must exceed the noise by the SNR, this is quite impractical.

5.2 Grid Pattern Matching

Grid algorithms can identify the pattern through one basic technique (although there are variants of it). A pattern database is generated from the star database and a pattern is constructed from the image produced by the sensor and looked for in the database. The pattern is designed through the use of grids, with zeros identifying the lack of a star, and ones identifying the presence thereof.

Given a set of stars in the FOV, one of them will be called *reference star*, then a section of the FOV that surrounds this star with a determinate radius is determined. Then all the stars contained on the zone are translated with the reference star to the center of the FOV. Then, an *align star* is selected from the accompanying stars, and the whole set is rotated with the reference star in the center, until the align star coincides with a reference frame.

Afterwards, a grid of g x g is made. if a grid cell contains a reference star, the cell will have a 1 assigned, otherwise it will have a 0. Those bits constitute pattern information for each star, and the database is made by sets of bits that give information on each single star. [28]

This method has the advantage of giving a light load to the DSP, and is rather simple. Neural network matching instead of direct comparison could even be tried

with it because it produces the kind of patterns that can be recognized by such a kind of algorithm.

The drawback is that at least 10 stars must be seen at any time in the FOV, and at least 7 stars need to be matched to properly identify the pattern. [27].

5.3 Angle Method Pattern Matching

Once a set of stars is located in the FOV, a simple method of establishing the relationship between two of them is determining the angle that separates the stars, matching this distance to an angle database that provides every possible angle smaller than the FOV.

In order to create the database of angles, a data structure named spherical quadtree is used. It is used to store objects located in a 2D space, being able to find one of them within a determined area without the need of going through every object in the space.

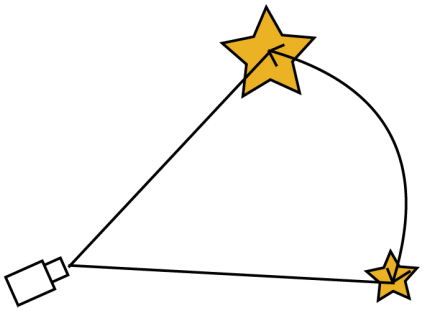


Figure 5.2: Angle pattern matching

The database angles are sorted by angle, and a technique named k-vector is used to have a faster search. The angle of each pair of stars is plotted against its location in the catalog, the equation of this line can then be used with the generated k-vector to locate the pair of stars given their angle. This allows a search within an uncertainty region, as opposed to a search of the whole catalog. [27].

The angle between two stars is given by $\theta = \arccos(r_1 \cdot r_2)$ where r_1 and r_2 are the vectors obtained from the image or the original catalogue (YBSC in our case).

Since the position of the star mark after the centroiding is subject to errors due to noise, it is possible that in a given FOV, there are more than one pair of stars with the same angles. Therefore, it is wise to use more than one couple in the measurement to reduce the probability of an error. This sis achieved through the use of the pivoting technique.

One of the stars from the original pair is tagged as the pivot. A set of couples from the catalogue or from a second image (in the case of tracking mode) that have

a similar angle, within a threshold margin, are selected. Then a third star in the original image is picked and the distance to the pivot star is calculated. Afterwards, only the pivot star that shares those two distances is identified. If duality should persist, an additional pivot star can be picked and the process is repeated.

The more stars are present in the FOV, the more likely it is that the results will be accurate.

5.4 Spherical Triangle Method Pattern Matching

A method was developed by C. Cole and J. Crassidis [29] to use spherical triangles made up by combinations of three stars. The core idea is that more information can be obtained from a triangle than just from three angles. This will accelerate the attitude calculation time and will allow us to use less stars in average than the angle method with pivoting.

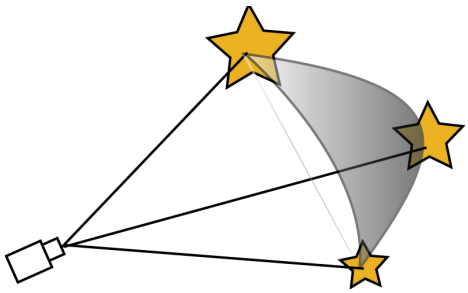


Figure 5.3: Spherical Triangle Pattern Recognition

One drawback is that we always will need at least three stars in the FOV, contrasting with the angle method, that could theoretically work with only two stars, but we have seen that in practice we will always need to do pivoting, which will make us use three stars and, quite frequently, even more.

The spherical triangles must be sorted by area and polar moment so we can use the k-vector, just like in the angle method.

The area of the spherical triangle is defined as follows, with b_i being the vector of the i^{th} star.

$$A_{SphT} = 4tan^{-1}\sqrt{tan\left(\frac{s}{2}\right)tan\left(\frac{s-a}{2}\right)tan\left(\frac{s-b}{2}\right)tan\left(\frac{s-c}{2}\right)} \tag{5.1}$$

where:

$$a = cos^{-1}\left(\frac{b_1 \cdot b_2}{|b_1||b_2|}\right) \tag{5.2}$$

$$b = cos^{-1}\left(\frac{b_2 \cdot b_3}{|b_2||b_3|}\right) \tag{5.3}$$

$$c = cos^{-1}\left(\frac{b_3 \cdot b_1}{|b_3||b_1|}\right) \tag{5.4}$$

and:

$$s = \frac{1}{2}(a + b + c) \tag{5.5}$$

Normally, two spherical triangles that have the same area can have a very different polar moment.

The polar moment is calculated using a recursive algorithmm dividing the triangle into small triangles. The area of each small triangle, dA_{SphT} is multiplied by the square of the arc distance from the centroid of each smaller triangle to the centroid of the large, original one. The result for each triangle is added.

$$I_{pSphT} = \sum \theta^2 dA \tag{5.6}$$

Where θ is the angular distance between the centroid of the small spherical triangle to the centroid of the greater spherical triangle, and dA is de area of the smaller triangle.

Even though the standard deviation error that we should use to provide an error threshold to match triangles is difficult to compute, through random simulations it has been possible to demonstrate that the probability of I_{pSphT} being inside the 3σ bound is 99.7% [25, p. 97].

5.5 Planar Triangle Method Pattern Matching

After the publication of the spherical triangle pattern matching, the same authors developed a new similar method that was based on planar triangles instead of spherical ones. [29]

Just like the previously explained algorithm, this one relies on the calculation of the area and the polar moment of the geometrical figure.

Like for the spherical triangle method, b_i is the vector of the i^{th} star. Then, the area of the triangle is calculated by using Heron's formula:

$$A_{PlanT} = \sqrt{s(s-a)(s-b)(s-c)} \tag{5.7}$$

where:

$$a = ||b_1 - b_2|| \tag{5.8}$$

$$b = ||b_2 - b_3|| \tag{5.9}$$

$$c = ||b_1 - b_3|| \tag{5.10}$$

and

$$s = \frac{1}{2}(a + b + c) \tag{5.11}$$

This equation can be used for both the local frame and the Earth inertial frame. However, given that the measurement is not error-free, it is necessary to obtain a set of boundaries to work on.

The planar area is a non-linear function of b_1, b_2 and b_3, so a linearization must be performed to establish a variance.

If

$$b_i = Dr_i \tag{5.12}$$

where b_i is the vector pointing to a star in the local frame of reference, D is the direction cosine matrix and r_i is the vector pointing to the star in the inertial frame of reference. For more information on the concept of direction cosine matrices, refer to section 6.1.1.

Shuster has demonstrated that nearly all the probability of error is concentrated in a small area in the direction of Dr_i, so that the sphere containing this point can be approximated by:

$$\tilde{b}_i = Dri + vi, v_i^T Dri = 0 \tag{5.13}$$

where $\tilde{b}_i$ is the ith measurement and the sensor error v_i is nearly gaussian. This can satisfy:

$$\begin{aligned} E\{v_i\} &= 0 \\ R_i &= E\{v_i vi^T\} = \sigma[I - (Dr_i)(Dr_i^T] \end{aligned}$$

where σ_i^2 is the variance and E means expectation.

In order to estimate the variance of the triangle area, the following partial derivative matrix must be evaluated:

$$H = \left[h_1^T h_2^T h_3^T\right] \tag{5.14}$$

where:

$$\begin{aligned} h_1^T &= \frac{\delta A}{\delta a}\frac{\delta a}{\delta b_1} + \frac{\delta A}{\delta c}\frac{\delta c}{\delta b_1} \\ h_2^T &= \frac{\delta A}{\delta a}\frac{\delta a}{\delta b_2} + \frac{\delta A}{\delta b}\frac{\delta b}{\delta b_2} \\ h_3^T &= \frac{\delta A}{\delta b}\frac{\delta b}{\delta b_3} + \frac{\delta A}{\delta c}\frac{\delta c}{\delta b_3} \end{aligned}$$

The partials with respect to a, b and c correspond to:

$$\begin{aligned} \frac{\delta A}{\delta a} &= \frac{u1 - u2 + u3 + u4}{4A} \\ \frac{\delta A}{\delta b} &= \frac{u1 + u2 - u3 + u4}{4A} \\ \frac{\delta A}{\delta c} &= \frac{u1 - u2 + u3 - u4}{4A} \end{aligned}$$

where:

$$\begin{aligned} u_1 &= (s-a)(s-b)(s-c) \\ u_2 &= s(s-b)(s-c) \\ u_3 &= s(s-a)(s-c) \\ u_4 &= s(s-a)(s-b) \end{aligned}$$

and the partials with respect to b_1, b_2 and b_3 are:

$$\begin{aligned}
\frac{\delta a}{\delta b_1} &= \frac{(b_1 - b_2)^T}{a} \\
\frac{\delta b}{\delta b_2} &= \frac{(b_2 - b_3)^T}{b} \\
\frac{\delta c}{\delta b_1} &= \frac{(b_1 - b_3)^T}{c} \\
\frac{\delta a}{\delta b_2} &= -\frac{\delta a}{\delta b_1} \\
\frac{\delta b}{\delta b_3} &= -\frac{\delta b}{\delta b_2} \\
\frac{\delta c}{\delta b_3} &= -\frac{\delta c}{\delta b_1}
\end{aligned}$$

The variance of the area is:

$$\sigma_A^2 = HRH^T \tag{5.15}$$

where:

$$R = \begin{bmatrix} R_1 & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & R_2 & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & R_3 \end{bmatrix} \tag{5.16}$$

where 0_{3x3} is a 3 x 3 matrix of zeros and R_1,R_2 and R_3 are given by the equation 5.14. Matrices H and R are supposed to be the evaluation of the true values, however using the measured value as an input implies second-order errors that are negligible [27].

Once the variance is found, the standard deviation can be obtained, and the boundaries in which the triangle is likely to exist in the ordered list of possible triangles, can be accurately estimated.

The polar moment for the planar triangle is much easier and less computationally expensive to calculate than for the spherical triangle:

$$I_{PPlanT} = A_{PlanT}\frac{a^2 + b^2 + c^2}{36} \tag{5.17}$$

Just like for the area, we must determine the boundaries with the standard deviation. To do this, the following derivative matrix must be calculated:

$$\bar{H} = \begin{bmatrix} \bar{h}_1^T & \bar{h}_2^T & \bar{h}_3^T \end{bmatrix} \tag{5.18}$$

where

$$\begin{aligned}\bar{h}_1^T &= \frac{\delta I}{\delta a}\frac{\delta a}{\delta b_1} + \frac{\delta I}{\delta c}\frac{\delta c}{\delta b_1} + \frac{\delta I}{\delta A}h_1^T \\ \bar{h}_2^T &= \frac{\delta I}{\delta a}\frac{\delta a}{\delta b_2} + \frac{\delta I}{\delta b}\frac{\delta b}{\delta b_2} + \frac{\delta I}{\delta A}h_2^T \\ \bar{h}_3^T &= \frac{\delta I}{\delta b}\frac{\delta b}{\delta b_3} + \frac{\delta I}{\delta c}\frac{\delta c}{\delta b_3} + \frac{\delta I}{\delta A}h_3^T\end{aligned}$$

with

$$\frac{\delta I}{\delta a} = Aa/18, \frac{\delta I}{\delta a} = Ab/18, \frac{\delta I}{\delta a} = Aa/18 \tag{5.19}$$

$$\frac{\delta I}{\delta A} = \frac{(a^2 + b^2 + c^2)}{36} \tag{5.20}$$

Ad with all the remaining quantities obtained from the area variance calculation. Finally, the variance of the polar moment is:

$$\sigma_I^2 = \bar{H} R \bar{H}^T \tag{5.21}$$

And just like in the area variance, the true values can be replaced with the measured values.

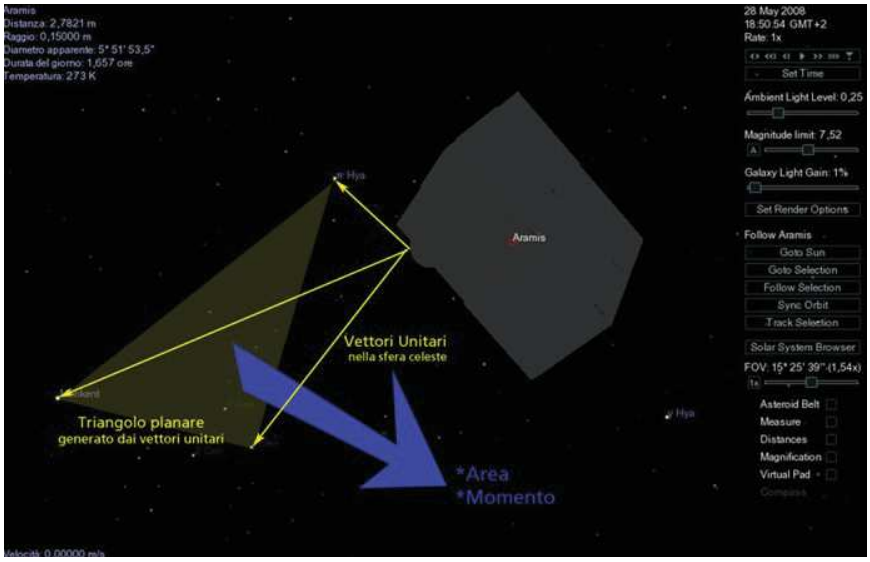


Figure 5.4: Planar triangle method, extraction of area and moment

A database with all the possible triangles that can fit into the FOV is generated. This database is located in variable triplette.

The contents of this database are displayed in figure 5.5. Column 1 (violet) contains the area of the triangle, Column 2 contains its polar moment, and columns 3 to 5 (shown in green) show the Harvard Revised Number or HRN (a number that identifies every star in a unique manner) of each star composing the triangle.

triplette <265292x5 double>

	1	2	3	4	5
1	0.0029	4.0898e-06	3	33	46
2	7.4200e-04	1.8146e-06	3	33	48
3	0.0045	6.3474e-06	3	33	74
4	0.0028	8.1889e-06	3	33	85
5	0.0052	2.3804e-05	3	33	118
6	0.0115	3.2434e-05	3	33	142
7	0.0109	3.6721e-05	3	33	188
8	0.0130	3.3212e-05	3	33	194

Figure 5.5: A sample of the contents of the triangle database

These parameters (area and polar moment) are also generated on board for the selected triplet on the imager. Given this measurement, the software shall look for the match on the triplet database that has the same planar triangle area. There are though, much more than one triangle with the same area in variable triplette. Therefore, the value of the polar moment is used, as it is difficult to find two triangles with both the same area and polar moment. This search is complicated by the fact that there are more than two hundred thousand triplets, and the measurement is affected by an error caused by noise.

5.6 K-vector search algorithm

Once the area, the polar moment and their respective standard deviations (σ) are measured, the next step is searching for the correspondent triangle in the compiled list of triangles.

As stated in a previous section, finding a direct match of the measured database is not nearly as easy as making two direct searches using conventional search algorithms (linear search, binary search, et cetera). Furthermore, the non-infinite accuracy of the reading makes it so that the area and polar moment measurements of the triangle formed by the catalog stars will not coincide exactly with the area and polar moment of the measured triangle. The so-called k-vector method provides efficiency to the triangle search.

For this purpose, the values of triangle area and polar moment were ordered. The resulting curve has a paraboloid shape (blue line in figure 5.6). It is possible now to produce an actual parabola (green line in figure 5.6) that closely resembles the paraboloid with the area values.

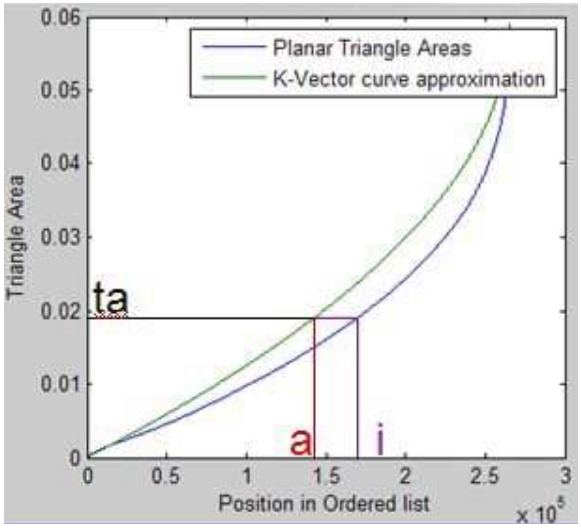


Figure 5.6: K-vector method applied to the Planar Triangle Areas

5.7 Pivoting

Until now, a process of selection has been made, composed by two searches: one of the measured area and another one of the measured moment. Each of these searches produces a group of results that fall within certain boundaries. Typically, various thousands of items are found for each of the two parameters. Then an intersection is made between the two groups to find a matching triangle. Ideally, this would produce a single item. In reality, a couple hundred of results are obtained.

The solution to this problem can be achieved with *pivoting*, This technique involves the selection of more than three stars.

The database created on the ground stores at least the 6 most brilliant stars per FOV at any time. Pivoting consists in not only using the three brightest stars, but more of them.

An area and moment measurement is performed on the triangle. Three things can happen: no match is found, case in which measurements are restarted from the beginning with a new acquired image (*no result*); a match is found (*conclusive result*), case in which the process is considered successful; and a third case in which no single match is found (*partial inconclusive result*). In the last case, the measurement

of a second triangle is made, this time keeping two of the original triangle's stars, and picking the fourth brightest star, and so on. Each time that a new pivoting occurs, all the triangles that do not share two stars with the previous triangle, are eliminated from the list. At the end, when no more triangles are available, there are three possibilities, *no result*, *conclusive result* and *inconclusive result*.

The limitation is in that the process can only be repeated three times with the current 6 stars per FOV configuration, as the Star Tracker would not "know" any further stars that were analyzed.

5.8 Star Selection and Matching

Many different algorithms exist for star selection, some predictive algorithms take into account the fact that a star will probably go out of the FOV before choosing it. Most of them have the brightness of the star as the main factor, since they have a lower error in the centroiding process. Other factors are the relative distance to other stars, and the probability that a star will disappear because of noise.

For the purposes of this thesis, the three brightest stars are picked. After the triangle identification, and in order to determine which of the stars of the triangle corresponds to each of the three catalog stars that were identified, a polling system was established.

Two criteria are used in order to determine the identity of each star of the triplet. The first criteria is the brightness. In a matrix that describes the likelihood of correspondence for each star against the three stars in the catalog, each star is associated with a catalog counterpart.

	Star b	Star m	Star d
Star m_1	•		
Star m_2		•	
Star m_3			•

Table 5.1: Star identification matrix

Where Star b is the brightest star in the triangle found in the database, Star m is the medium brightness star, Star d is the dimmest star in the database triangle, Star m_1 is the brightest star on the imager, Star m_2 is the medium brightness star on the imager, and Star m_3 is the dimmest star on the imager.

However, when the three stars have a very similar brightness, this can lead to false matching, as noise can make one dimmer star look as bright or brighter than the others.

The coefficient of variation is used to assign a value to this measurement. This way, if the set of star brightnesses are too similar, a small score is assigned to this criteria, and if the three stars are very different, they will get a greater score. In order to provide a unit-less score, the coefficient of variation is used. The coefficient of variation is a the normalized measure of dispersion in a set of data [30]. This way, the correspondance matrix takes the shape of table 5.2.

	Star b	Star m	Star d
Star m_1	$\frac{\sigma_{br}}{\mu_{br}}$		
Star m_2		$\frac{\sigma_{br}}{\mu_{br}}$	
Star m_3			$\frac{\sigma_{br}}{\mu_{br}}$

Table 5.2: Star identification matrix

where σ_{br} is the variance of the brightness of the set of measured stars, and μ_{br} is the average brightness of the measured stars.

Afterwards, a criteria based on the angles between stars is used. The angles between the stars on the imager are measured, and they are calculated as well for the set of three stars of the catalog. A correspondence between the three angles is established, and each star on the imager is matched with a star on the catalog. The coefficient of variation of the angle measurement is used as a score in the correspondence matrix. Therefore, if the match established by brightness similarity coincides with the that established by angle similarity, they are summed up and a clear matching is made.

Still, if they differ, more weight will be given by the measurement that has the higher coefficient of variation. An example of this is provided in table 5.3.

	Star b	Star m	Star d
Star m_1	$\frac{\sigma_{br}}{\mu_{br}} + \frac{\sigma_{\angle}}{\mu_{\angle}}$		
Star m_2		$\frac{\sigma_{br}}{\mu_{br}}$	$\frac{\sigma_{\angle}}{\mu_{\angle}}$
Star m_3		$\frac{\sigma_{\angle}}{\mu_{\angle}}$	$\frac{\sigma_{br}}{\mu_{br}}$

Table 5.3: An example of star matching

where $\sigma_{\angle}$ is the variance of the measured angles, and $\mu_{\angle}$ is the average angle. Here, both criteria coincided in matching Star m_1, but differed in Stars m_2 and m_3, in this case, the criteria with higher $\frac{\sigma}{\mu}$ will decide the matching process.

This specific case could be originated by the sides of the planar triangle being too similar, thus not providing a significant difference that can help clearly identify the sides of the triangle (greater, medium, smaller), implying an incorrect identification of individual stars. This will be remedied by the fact that the variance of the measurement, divided by the average angle will give a small value, and the score of the (incorrect) association of Stars m_2, m_3 and Stars b and d, respectively, will "lose" if the brightness values are less homogeneous, as $\frac{\sigma_{\angle}}{\mu_{\angle}} < \frac{\sigma_{br}}{\mu_{br}}$.

5.9 Aided Mode

As seen in the introduction, apart from the Star Tracker, the `Aramis` Satellite has other sources of attitude information, namely the sun sensor and the earth magnetic field sensor.

Originally, the Star Tracker was designed to operate only in Lost in Space (LIS) mode, but later, a scheme was devised for an Aided Mode too. The Star Tracker shall operate in one of these two modes, depending on the availability of a coarser attitude measurement from the aforementioned sensors.

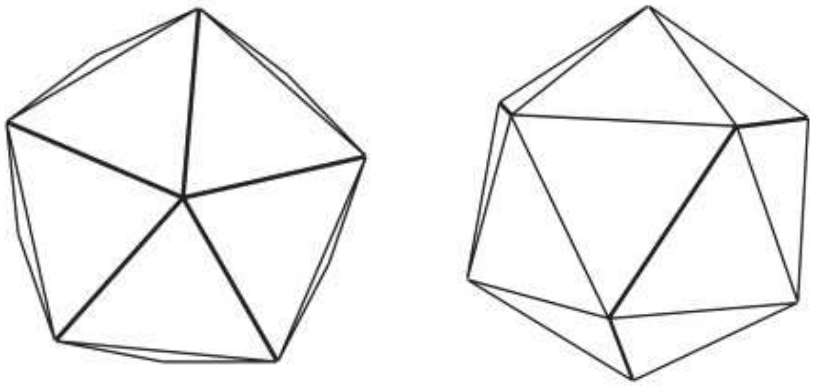
The aided mode makes use of a Geodesic Sphere, or *Dymaxion Map*, often credited to Buckminster Fuller, but whose concept was actually created a couple of decades earlier by Walter Bauerfeldwhile while working on a planetarium projector at Carl Zeiss [31].

The Dymaxion Map is a projection of a map of the Earth onto the surface of a polyhedron. This polyhedron can be flattened and form a 2D map that can maintain an accurate reflection of the proportions of the Earth's map, but the characteristic that is of most interest for the star tracker project, is that it provides an homogeneous subdivision of the celestial sphere.

The first alternative to this approach would have been a more intuitive and easier to understand RA/DEC subdivision, having a zone, for example defined as the area delimited by RA 0 hours, RA 1 hours and DEC 0 degrees, DEC 10 degrees. This has some important inconveniences. First, there would be a greater amount of subdivisions near the poles than near the equator, so a larger grid would be needed to represent the equator in an adequate manner. Second, there are two singularities at the poles, where longitude has no longer a meaning.

Therefore, a grid is generated by the iterative subdivision of an icosahedron (a polyhedron with 20 faces). Each triangle can be split into 4 new triangles in each iteration, giving shape to a new figure each time, iteration number one receives the name of 2v (1v would be the icosahedron itself), iteration number two receives the name of 3v and so on. Two iterations are made, that can be seen in figure 5.8, creating at the end a 3v polyhedron with 320 faces.

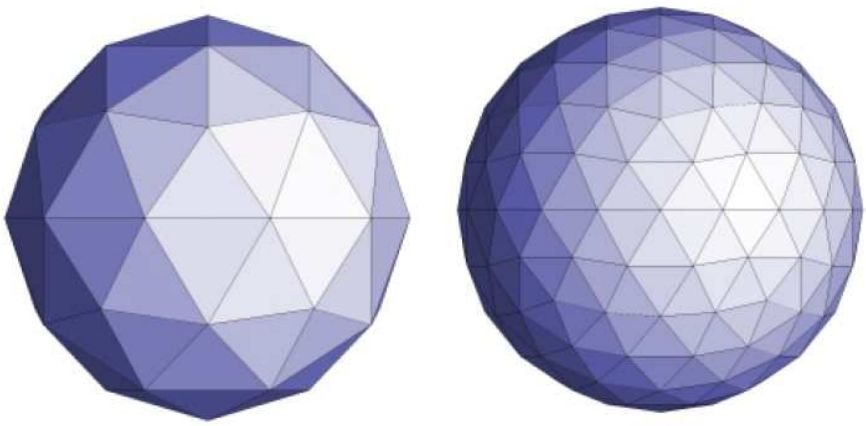
This iteration was chosen so the distance from the farthest angle of the face of



(a) Icosahedron view from *top* (b) Icosahedron view from *side*

Figure 5.7: Icosahedra shown from different angles

interest, to the nearest external side of an adjacent triangle will be equal or greater than the radius of the FOV. This will guarantee that any triangle whose center is in the face of interest, will have all of its stars in the area that comprises the face of interest plus all its adjacent faces.



(a) A 2v geodesic sphere (b) A 3v geodesic sphere

Figure 5.8: Geodetic spheres iteratively produced from an icosahedron

Every triangle in the database of star triangles (not to be confused with the triangles of the Dymaxion Map) is tagged with the number of the face of the iteratively-divided polyhedron in which its unitary vector falls. Once a first approach of attitude is obtained from the coarser satellite sensors, the system evaluates on which of the

320 faces this vector falls in. It will then exclude any triangle whose center is located outside this face or its directly adjacent faces.

In figure 5.9, the face in which a coarse measurement fell, and the adjacent faces are highlighted.

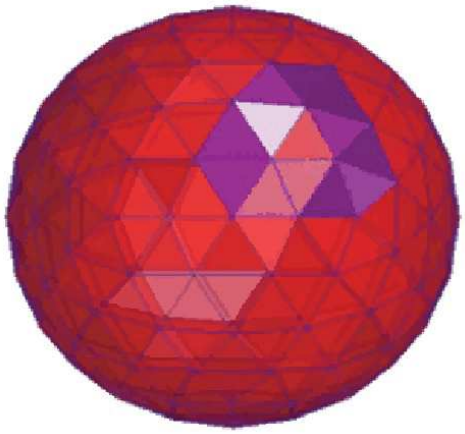


Figure 5.9: Geodetic sphere showing the area of interest upon reception of the coarse attitude measurement

This method, other than helping avoid identifying a triangle that is very far away from the zone of interest, would also reduce the probability of mistakingly identifying a triangle formed by false stars (planets, whose ephemeris data would add too much unnecessary complexity if its use was intended; or SEUs), as the initial pool of triangles will be much smaller, and the chances of finding a database triangle that resembles in area and polar moment the acquired one, decrease considerably.

The polyhedron for the tagging of the database triangles and the on board software uses a file containing the 3D coordinates of the vertexes, the vertexes of each face, and a file that defines for each face, which sides are adjacent to it.

The tags that are created in compilation phase are a new vector of values that complement the database shown in figure 5.5, or triplette variable in the matlab code.

Chapter 6

Attitude Determination

Up to now, an image of the star field has been captured, the position of the stars in the imager plane have been identified, the properties of a planar triangle formed by the three individual vectors were measured, and with the help of a triangle database, a star catalog and the coarse approximation of the attitude from other sensors, the Harvard Reference Number and coordinates of the three stars have been determined.

The last step in the process, other than communicating the measurements to the other subsystems, is actually provide a useful measurement of the attitude.

There is more than one way to do this, some Star Trackers (mostly the older ones) have as an output only the position of the stars, as there was a time when there were not so powerful microcomputers to process this data, and this had to be done by the Spacecraft main computer.

6.1 Attitude Representation

Attitude can be represented by Star Trackers in a variety of ways. The most used are Direction Cosine Matrixes (DCMs), Euler Angles and Quaternions. They all have in common that they represent a rotation, and rotating the three axes of the inertial frame with one of these transformations will yield the attitude with respect said inertial frame.

6.1.1 Direction Cosine Matrix

Rotation matrixes are matrixes that applied to a vector, rotate it, preserving its length. The three unit vectors that result in the current attitude can be expressed as the elements of a 3×3 matrix, called the Direction Cosine Matrix.

If the initial axes, that is, the axes of the inertial frame are (x,y,z) and the local axes are (x',y',z') , and $\theta_{x',z}$ is the angle between the x' axis and the z axis, the

rotation matrix can be expressed as in equation 6.1.

$$\mathbf{R} = \begin{bmatrix} \cos(\theta_{x',x}) & \cos(\theta_{x',y}) & \cos(\theta_{x',z}) \\ \cos(\theta_{y',x}) & \cos(\theta_{y',y}) & \cos(\theta_{y',z}) \\ \cos(\theta_{z',x}) & \cos(\theta_{z',y}) & \cos(\theta_{z',z}) \end{bmatrix} \tag{6.1}$$

The elements of the DCM are not completely independent, as Euler's theorem states that a rotation can have three degrees of freedom. [32]

This method is intuitive and straightforward. The disadvantage lies in the burden added by having 9 elements with a great deal of redundancy. [33]

6.1.2 Euler Angles

Euler angles are a way of representing the rotation by the means of three successive rotations. The angles that describe thsese rotations are called Euler Angles. There are a variety of conventions, the most common is the rotation about the z axis, then a rotation in [0,pi]about the x axis and finally a rotation about the z' axis. [34]

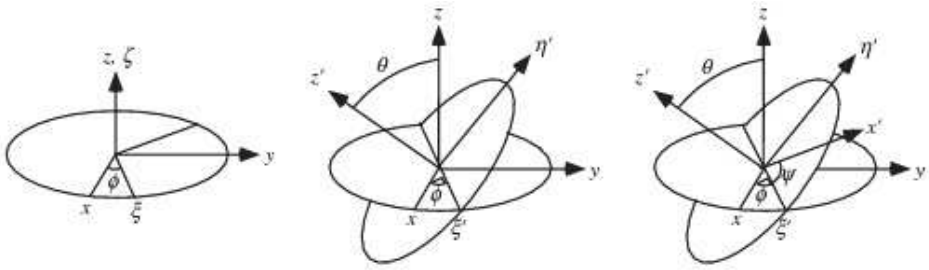


Figure 6.1: Euler Angles rotation in the z-x-z convention. Source: Mathworld

This convention is referred to as the z-x-z However, other conventions are also widely used, such as x-y-z and z-y-x. The latter, because of its relationship with Tait-Bryan rotations, is commonly denoted as *roll*, *pitch* and *yaw* rotation.

- Roll, or rotation about the x axis is often denoted as ϕ
- Pitch, or rotation about the y axis is often denoted as θ
- Yaw, or rotation about the z axis is often denoted as ψ

In this text, Tait-Bryan angles or rotations and Euler angles or rotations are used interchangeably, and they refer to the z-y-x Euler angles or rotations.

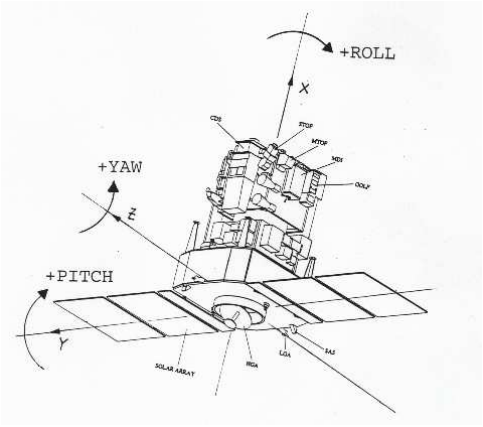


Figure 6.2: Roll, Pitch and Yaw on the SOHO spacecraft. Source: ESA

6.1.3 Quaternions

Quaternions are an extension of complex numbers. They are a way to note mathematically orientations and rotations. Their advantage over Roll Pitch and Yaw or Euler angles, is that they avoid the problem of gimbal lock, the singularity that occurs when two of the axes coincide and one degree of freedom is lost [35].

They are also advantageous if compared with DCMs, since their computational efficiency is superior, and are more stable from a numerical point of view [36].

A unit quaternion is essentially:

$$\begin{aligned} \mathbf{q} &= \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T \\ |\mathbf{q}|^2 &= q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \end{aligned}$$

A quaternion can be interpreted as a rotation around an axis with equation 6.2.

$$\begin{aligned} \mathbf{q}_0 &= \cos(\alpha/2) \\ \mathbf{q}_1 &= \sin(\alpha/2)\cos(\beta_x) \\ \mathbf{q}_2 &= \sin(\alpha/2)\cos(\beta_y) \\ \mathbf{q}_3 &= \sin(\alpha/2)\cos(\beta_z) \end{aligned}$$

Where β_x,β_y,β_z represent the axis of rotation (direction cosines), and α is an arbitrary angle of rotation.

They can also be calculated from the Tait-Bryan angles as described by formula 6.2.

$$\mathbf{q} = \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix} \tag{6.2}$$

6.2 QUEST Algorithm

The least-square cost function

$$J_A(A) = \frac{1}{2}\sum_{k=1}^{n} |\hat{W}_k - A\hat{V}_K|^2 \tag{6.3}$$

is Wahba's cost equation, where $n = 2$ and $a_1 = a_2$. A is the attitude matrix, $\hat{W}_k$,$k = 1,...,N$, are the measured directions in the body frame and $\hat{V}_k$,$k = 1,...,N$, the corresponding reference directions in the Inertial frame. This equation is the cornerstone of modern attitude determination.

Wahba's problem can be rewritten as:

$$\begin{aligned} J_A(A) &= \sum_{k=1}^{n} a_k - \left[\left(\sum_{k=1}^{n} a_k \hat{W}_k \hat{V}_k \right)^T A \right] \\ &= \lambda_o - tr[B^T A] \\ &= \lambda_o - g_a(A) \end{aligned}$$

The matrix B is called the *attitude profile matrix*. The function $g_a(A)$ is alled the *Wahba gain function* and it is a maximum when $J_A(A)$ is a minimum.

Davenport showed that the Wahba gain function could be written in terms of the quaternion $\bar{q}$ in the following manner:

$$g_{\bar{q}}(\bar{q}) = g_A(A(\bar{q})) = q^{-T} K \bar{q} \tag{6.4}$$

where

$$K = \begin{bmatrix} \begin{pmatrix} S - sI & Z \\ Z^T & s \end{pmatrix} \end{bmatrix} \tag{6.5}$$

and $S = B + B^T$, $s = trB$, $Z = [B_{23} - B_{32}, B_{31} - B_{13}, B_{12} - B_{21}]^T$

The minimization of $J_A(A)$ can be done through the solution of the eigenvalue problem:

$$K\bar{q}^* = \lambda_{max}\bar{q}^* \tag{6.6}$$

where λmax is the largest eigenvalue for the $4X4$ matrix K.

The QUEST algorithm was developed by Malcolm D. Shuster and it is regarded as an important discovery, as it provides a solution that at the time was 1000 times faster than any other method. In it, the characteristic polynomial for K:

$$\psi(\lambda) \equiv det[\lambda I_{4x4} - K] \tag{6.7}$$

is defined as

$$\psi_{QUEST} = \lambda^4 - (a+b)\lambda^2 - c\lambda + (ab + cd - d) \tag{6.8}$$

where

$$\begin{aligned} a &= s^2 - tr(adj)S \\ b &= s^2 + Z^T Z \\ c &= det(S) + Z^T S Z \\ d &= Z^T S^2 Z \end{aligned}$$

Then the attitude quaternion is calculated with Rodrigues' parameters:

$$q_{opt} = \frac{1}{\sqrt{1 + |y_{opt}|^2}} \begin{bmatrix} \begin{pmatrix} y_{opt} \\ 1 \end{pmatrix} \end{bmatrix} \tag{6.9}$$

$$y_{opt} = [(\lambda + s)I - S]^{-1} Z \tag{6.10}$$

Shuster proposed that a good enough initial value to calculate the eigenvalue with the Newton-Raphson method is $\lambda = 1$. However, simply by inserting $\lambda = 1$ into equation , a very good precision is achieved [37].

After obtaining the attitude, a Kalman filter may be needed to smooth the output.

Chapter 7

Code Usage and Description

So far, a set of algorithms was chosen from the pool of available image processing and pattern recognition algorithms and coded into m-files. The process in order to optimize a catalog, and to create a pattern catalog for the chosen algorithm was also described. All of them were coded into mostly m-files, but also C files and celestia celx files that can be found in the thesis CD. The main files can be found in the appendix.

The code of the Star Tracker is divided into 3 sections. The first, deals with the code that must be run on the ground, while preparing the system. A second part of the code corresponds to the Celestia scripts that are used to capture images from the simulator, and finally a third part of the code are the m-files that simulate the code on board the spacecraft.

Since the MATLAB code was created using procedural programming, it doesn't translate quite well into UML Objects and Classes. Therefore, a linear description of the main files will be done in this chapter, as well as of the functions they call. The procedures required to use the algorithms are included in this description. A scheme of the file inputs and outputs of each m-file is given in figure 7.1.

7.1 Ground Software

7.1.1 Star Compilation

The application `scanning.exe` (source at scanning.cpp) must be run in order to process the file `catalog.dat` (complete YBSC). It converts it to a format MATLAB can read and saves it as `catalogy.dat`. `catalog.dat` must be in the same directory of `scanning.exe`. `catalogy.dat` can be now read by MATLAB.

`compilastelle.m` is the m-file that generates the catalogs of the star tracker. At its beginning, important factors are defined such as the dimmest star it will take

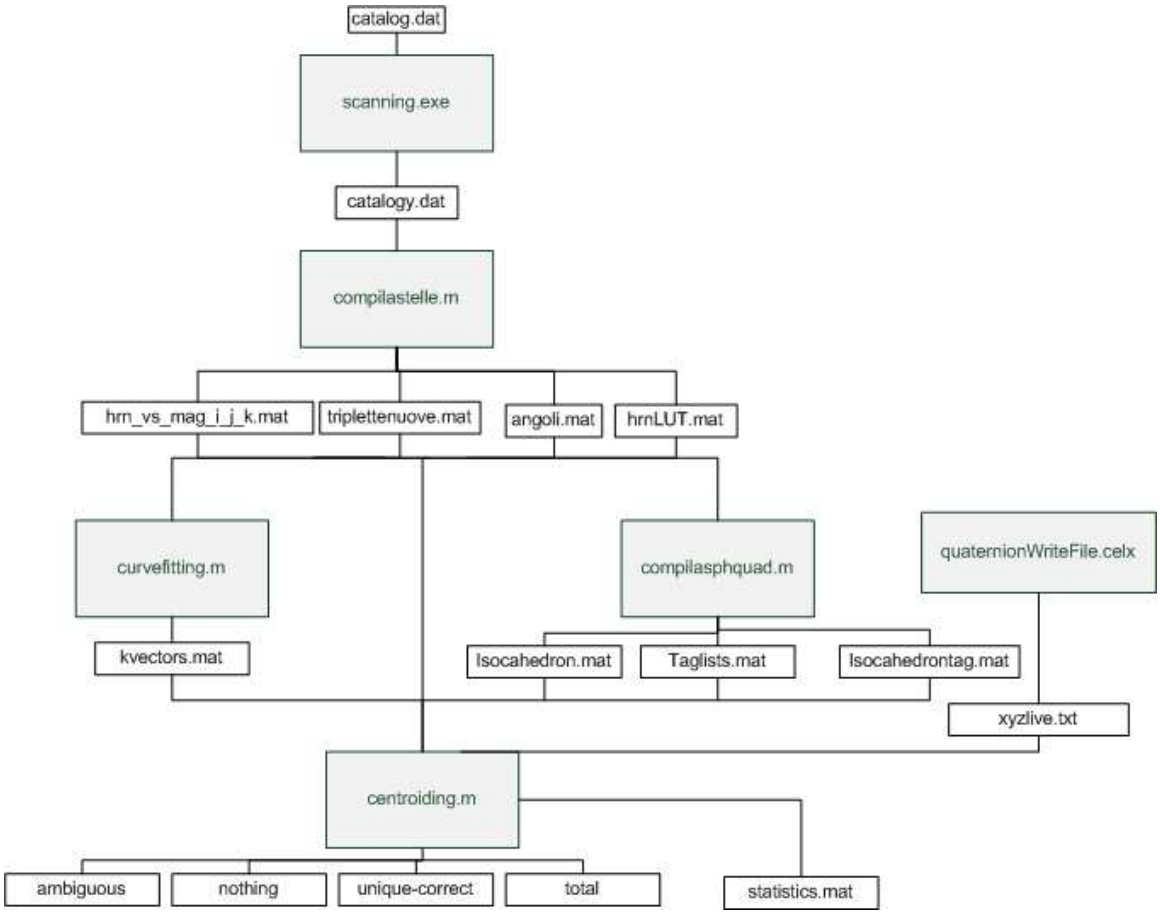


Figure 7.1: Scripts and files used as an input and output for each script

into account (variable `magnitude_filter`), the number of stars per FOV (variable `StelleXFOV`) and the FOV itself (variable `FOV`).

`compilastelle.m` calls function `getstars(magnitude_filter)`, which loads the stars from catalogy.dat, taking away the stars dimmer than `magnitude_filter`. It obtains a list of Harvard Revised Numbers, visual magnitudes, RAs and DECs.

This list is passed to `pulisci (hrnyale, vis_magyale, iyale, jyale, kyale, FOV, StelleXFOV)` which returns a list of only the `StelleXFOV` brightest stars per `FOV`. **Note:** This proccess can take several hours (up to one day) to complete on an average desktop computer. The list of stars are saved to file `hrn_vis_mag_i_i_j_k.mat`

Following this, a plot of the optimized and unoptimized stars is displayed.

Then, a sparse matrix with all the angles between stars (this was used to do various tests but not is not required by the planar triangle algorithm) is created in variable `angoli`, where angles can be addressed as `angoli(hrn1,hrn2)` where `hrn1` is the HRN of the first star and `hrn2` is the HRN of the second star.

At this point, a database of the planar triangles is built. A loop is run, and for each cycle it checks the neighbors of every star, that are within the limits specified for the FOV, with the help of function `getneighbors2 ([double(hrn), i, j, k], FOV, hrn(p), angoli)`. Then it checks whether each possible triangle of this star and its neighbors has been already added, and if not, it calculates the necessary planar triangle algorithm values and adds the triangle, the two values (aea and polar moment), and the three identities of the stars to a list named `triplette`. `triplette` is saved in file `triplettenuove.mat`.**Note:** this process can take many hours (up to three days) to perform.

As these processes can take long and overwrite previous versions of the catalogs, it is advisable to make backups of `catalog.dat` and `triplette.mat` before executing the code.

7.1.2 K-vectors generation

`curvefitting.m` receives `triplettenuove.mat`, `hrn_vis_mag_i_j_k.mat`, `bound_coef.mat`, `hrnLUT.mat` and `angoli.mat`.

It calculates the parabolas for the area and polar moment vectors with MATLAB routines, making the curve that describes the ordered planar moments or areas coincide with the generated cuve at a) the lowest value b) the highest value.

It then saves the parameters that describe the curves in `kvectors.mat`.

7.1.3 Buckminster Fuller polyhedron tags generation

`compilasphquad.m` is an m-file that receives `triplettenuove.mat`, `hrn_vis_mag_i_j_k.mat`, `bound_coef.mat`, `hrnLUT.mat` and `angoli.mat` as an input.

With the aid of function `sphere_tri` (GNU GPL code by J. Leech) an 2v polyhedron originated from an icosahedron is created in variable `FV`. `FV` contents are the vertices `FV.vertices` and faces `FV.faces`. Then the centers of each face are determined and all the triangles from `triplettenuove.mat` are tagged with the number of the faces that is the nearest to the centroid of said triangle.

The list of ordered HRN triplet indexes, organized in base to the face they are located (`finaltags`), plus a vector with delimiting markers for this list, to know where a face starts and ends (`tagmarkers`), are saved in `taglists.mat`.

7.2 Simulation scripts for Celestia

Image capture is done through scripts that tell Celestia how to move the camera in the 3D environment and take screenshots with certain intervals.

7.2.1 Celestia initialization

The file `init.cel` sets the desired magnitude limit and Field of View. Then, it weights for some time, selects an arbitrary star, and stars a rotation around the three axes one after another, with given parameters of duration and angle rate in degrees per second. For example, parameters `duration 3, rate 10, axis [0 1 0]` give a rotation of 3 seconds at 10 degrees per second, around the y axis.

7.2.2 Image captures and attitude reference files

During the 5 (extendable) seconds that precede the rotation, function `quaternion WriteFile.celx` must be called. It records the unitary vector that describes the current attitude with two degrees of freedom in the file `xyzlive.txt`. It also calculates saves a file with the attitude quaternion.

Finally, its most important function is grabbing the screenshots and saving them as png files that are recorded in the Celestia main directory (no way to overcome this inconvenient was found).

If one wants only to visualize the information on the current attitude and euler angle, it is possible to use the file `quaternionWatch.celx`.

7.3 In-flight Software

The m-file `centroiding.m` is the core of the simulator (the name may be misleading since it refers to a single part of the process, but it really performs all the algorithms). It grabs all the produced catalogs and loads the image frames.

These frames must be moved from the Celestia directory into the base directory. One also must be careful, because at the beginning of the sequence frames, there may be frames with written characters that can affect the measurement. They must all be deleted before feeding the frames to `centroiding.m`.

While loading, it uses MATLAB's function `imnoise()` to add the desired gaussian noise to each frame before storing it into memory.

A circular mask is applied in order not to see stars that are out of the FOV.

A loop is started, which scans every frame to process relevant attitude information.

Function `avgthreshold (sample, accufrms, imsize)` makes an average of the threshold over the last 5 samples and returns a mask with the pixels that are above the threshold.

After this, the m-file will pass the mask and the image size to, and call either `spikeremoval1(mask,imsize)` or `spikeremoval1(mask,imsize)`. This functions can filter away the remaining noise peaks.

Then, `locatestars(sample)` is called. Sample is the image of the stars after the final mask has been applied to it, that is, the stars with noise removed. This function performs the weighted sum technique and returns the location of the star centroids in the imager plane.

Function `identifyFace(xyzlive(fri,:),centers,adjacent)` takes the measurement from the sun sensor attitude (`xyzlive`) and identifies the faces of the Buckminster Fuller Polyhedron in which the vector falls, along with the adjacent faces.

The triangles produced by the centroids are then analyzed with function `planart (sample, pos(1:3,:), FOV, sensorsz, pp, varang);`. As can be seen, the function is passed the positions of the stars in the imager plane, the FOV, the sensorsize, the pixel pitch and the angular variance.

Function `idtriplet3(triplette, areat(1), areaMaxOffset(1), ipmt(1), momMaxOffset(1), tvectorarea,tvectormoment, indarea, indmoment, pm, hm, km, ka,pa, ha,faces2look4, finaltags, tagmarkers)` is in charge of performing the k-vector search and identify a set of triplets that include the solution. This function is called for each pivot triangle too, and the results are intersected in order to reduce the pool of candidates.

When a unique solution is found for a triangle, the voting matrix `score` is created, it contains the 3 stars to be identified in the vertical axis, and the catalogue counterparts in the horizontal axis. A score is applied based on the angle and brightness criteria. Finally, unique star of the catalog, is assigned to each of the three stars that are being measured.

The image and overlays with the centroids and lines showing the triangles are then displayed using MATLAB native functions.

Chapter 8

Algorithm Performance

The basic data flow in the device can be summarized with figure 8.1. These functionalities will be assesed, up to the identification of the single stars and the vectors associated to each one of them (that is, no quaternions are obtained).

8.1 Dynamic Image Thresholding and Centroiding

An image sequence in figure 8.2 depicts the results of the thresholding and centroiding algorithms on a Celestia image sequence imported into MATLAB.

There are 8 subfigures, each one depicting one frame of the sequence. Each subfigure, on its topmost, left area, shows the noisy simulated image of the sky. On the upper right, the mask that takes away most of the noise thanks to the averaging technique is visible. Below, on the left of each subfigure there's the mask used to remove noise peaks and do the image segmentation, in order to do the centroiding.

8.2 Results with pivoting (LIS)

Figure 8.3 depicts the results of the centroiding, planar triangle calculation and pivoting, and the use of k-vectors to accelerate search time.

A noisy simulated image of the sky is shown. The simulation was done using Celestia with an appropriate script that initiated with the Canis Majoris constellation, particulary, Sirius, Adhara and Wezen, in the FOV.

The FOV corresponds to the inner part of the blue circle in the image. Every star outside of the FOV was taken out this time, since this showed an increase of performance in the algorithm: with stars outside the FOV, triangles greater than the

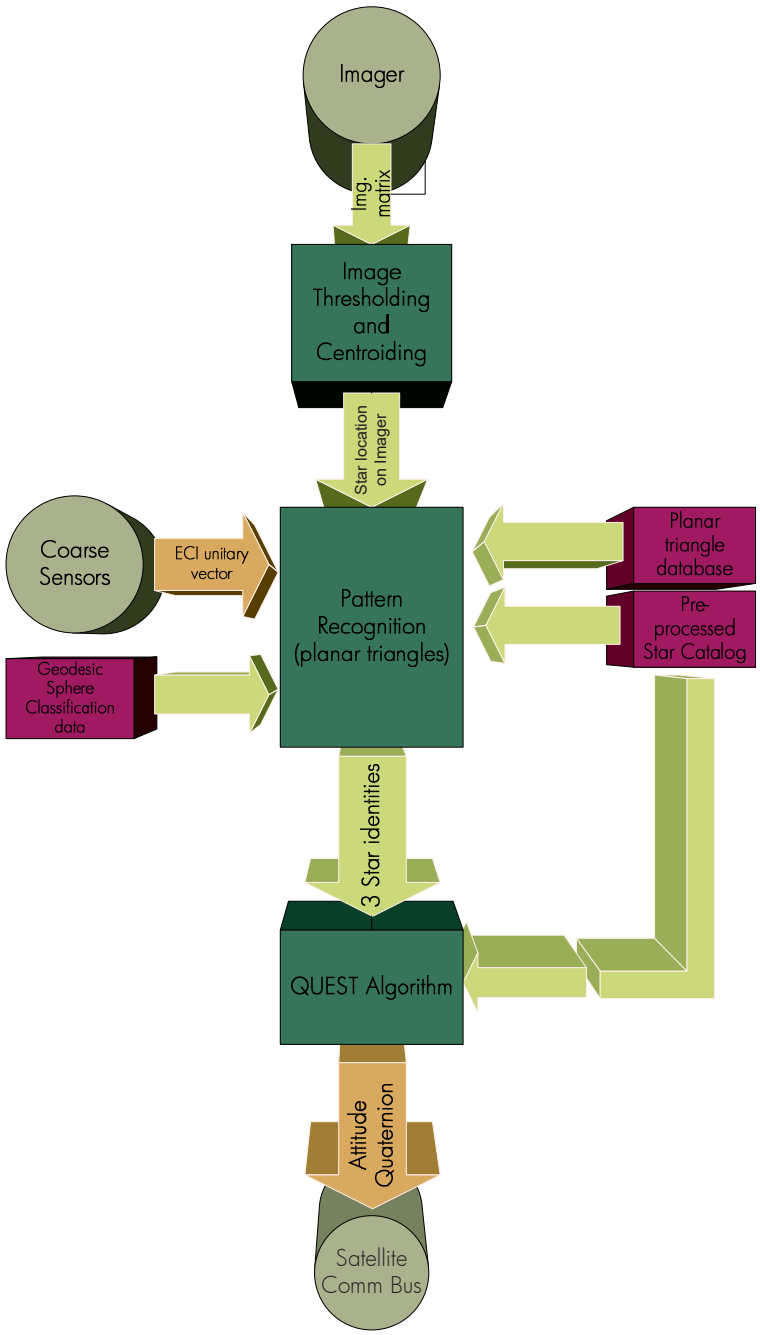


Figure 8.1: Star Tracker data flow

FOV were generated, triangles which obviously couldn't be found in the database. Besides, in the actual hardware, the shade will cover the zones outside this circle.

The main triangle is depicted in red color. This will be the triangle of which we will verify validity through the use of pivot triangles. Blue triangles show the pivots used.

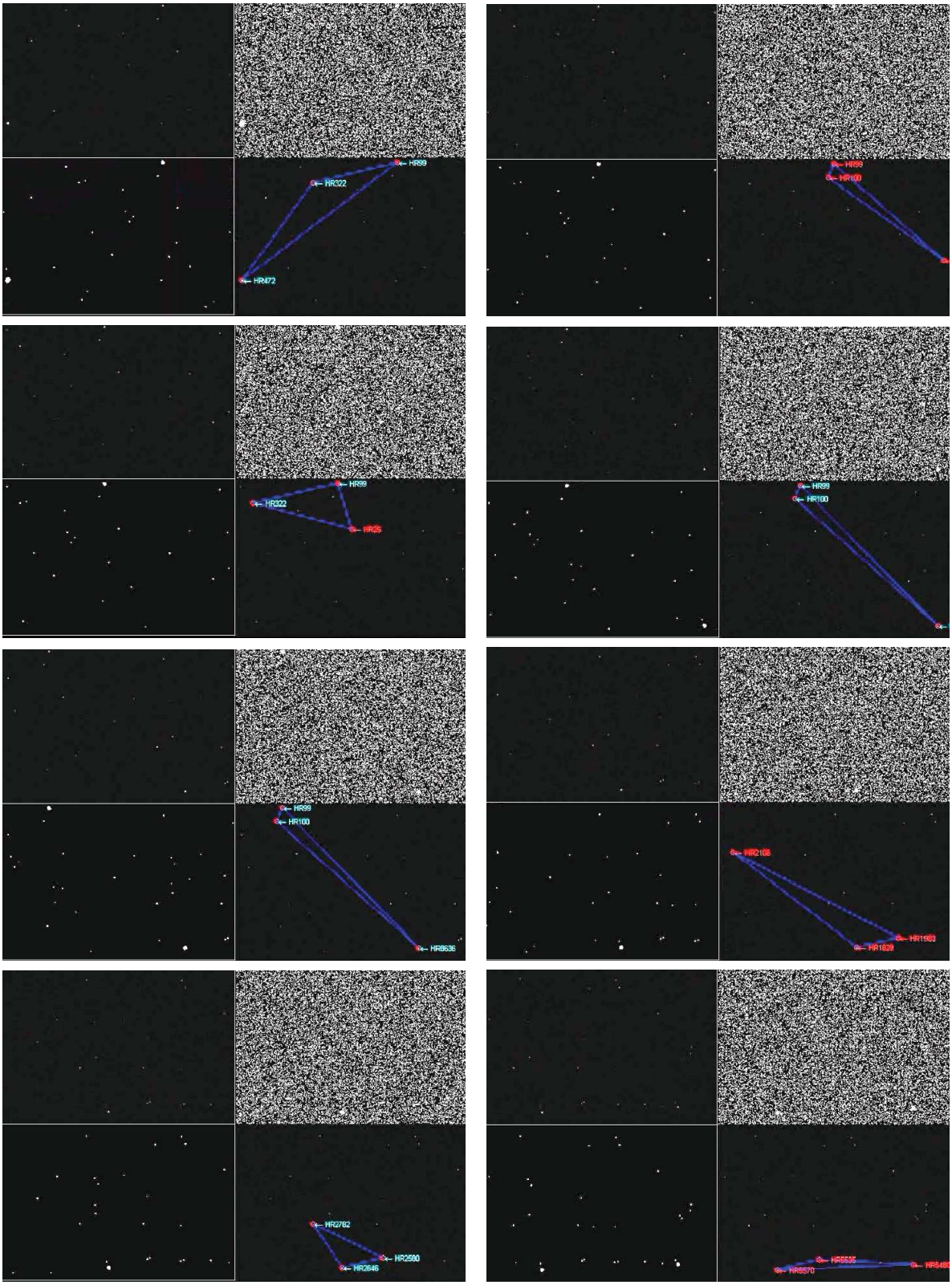
When three pivot triangles are generated and no conclusive result is found, we call it an "ambiguous result", and it is displayed in yellow on the top right corner of the image. When less than 3 pivot triangles are needed, we display a green sign indicating it, and a list of the 3 Harvard Revised Numbers for the identified triplet.

If no result is found within three pivot triangle iterations, a red sign is displayed indicating it.

A non ambiguous result was found 60% of the time. Many of these results though, were found to be wrong. Figure 8.3 shows the initial frames of the sequence.

8.3 Results with pivoting (aided mode)

The simulation sequence, when the aided mode was enabled, gave only true conclusive results as opposed to when the LIS mode was active. It helped increase the reliability of the algorithm, as it automatically rejected results that are blatantly far from reality. The distribution of results can be seen in figure 8.4.



86

Figure 8.2: Dynamic Image Thresholding and Centroiding on a Simulated Sequence

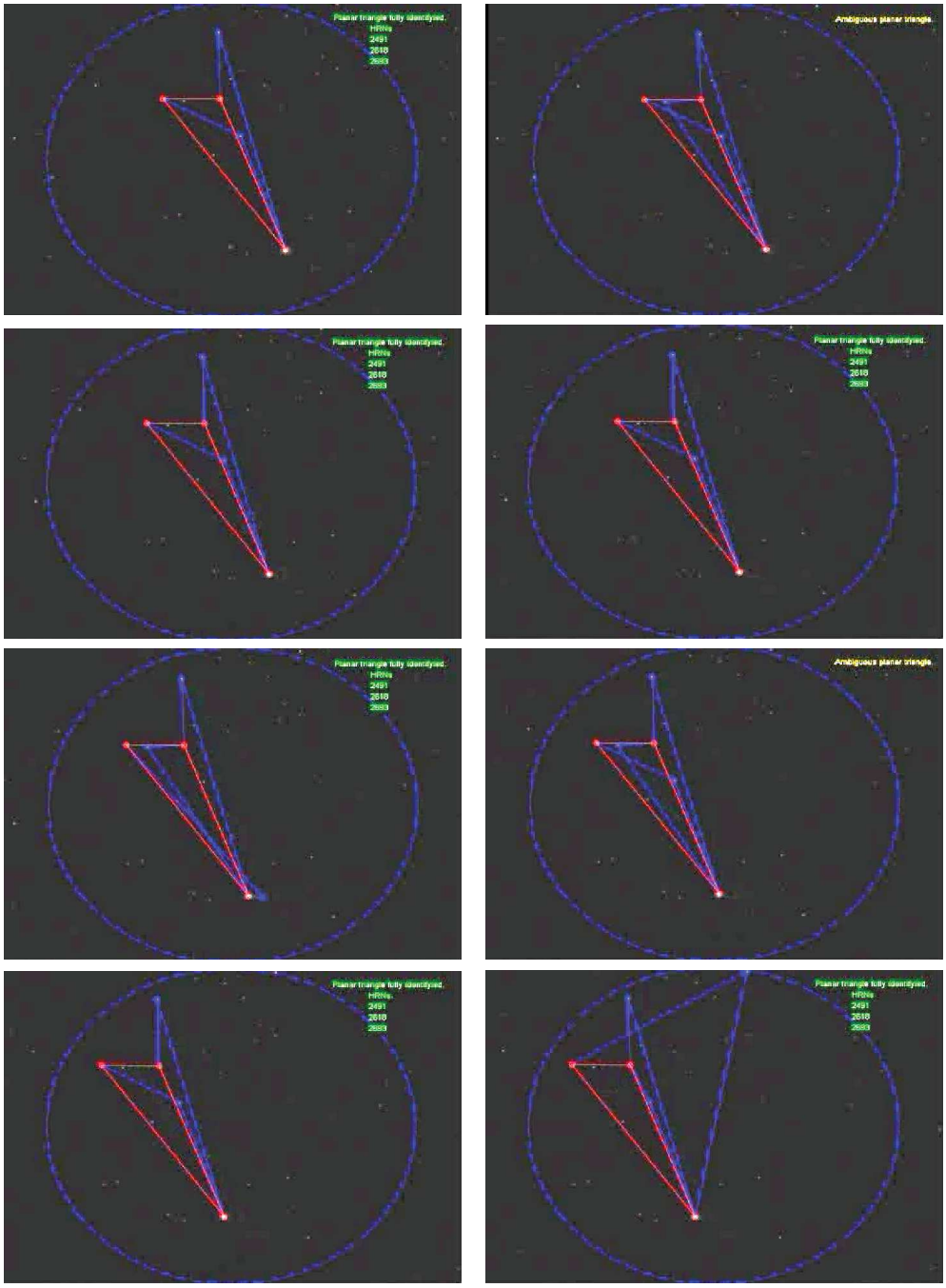


Figure 8.3: A sequence of identified stars. In this sequence, Adhara, Wezen and Sirius of Canis Majoris are all correctly identified

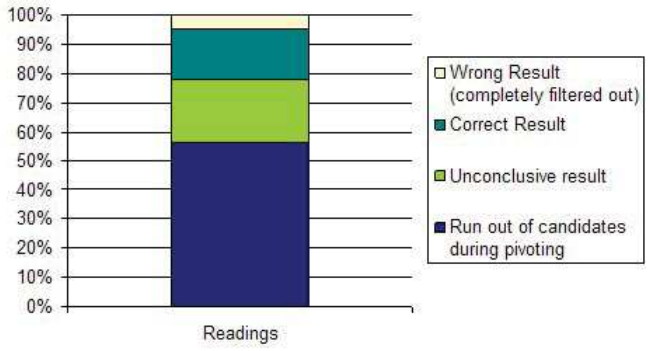


Figure 8.4: Distribution of the simulation results

Chapter 9

Conclusions and Future Work

A low cost, low mass and low power consuming Star Tracker design was proposed and compared with the state of the art. A preliminary schematics design was produced, and components were chosen based on radiation tolerance, performance and commercial off-the-shelf availability criteria.

Radiation effects on the imager can be mitigated by data encoding and unused pattern exclusion by the means of an operative mode (aided mode) in which data from coarser `Aramis` sensors is used. This allows to have a smaller pool of patterns, so the likelihood of identifying a triangle that contains a false star

Different available algorithms were analyzed and a set was picked from them. The thresholder, centroider and pattern recognition algorithms were first tested with the use of the Celestia 3D planetarium.

A set of m-files was produced and documented in order to support the future development of this project.

A star database filterer that can generate a new database with the x brightest stars per FOV was developed and used to make a 6 stars per FOV list with all the necessary data for each star. Along with it, a pattern database generator for the planar triangle was made and used to produce a database of 265292 triangles with their correspondant HRNs, areas and polar moments.

The algorithms were first tried on carefully configured screenshots and then on image sequences.

The simulation sequence, when aided mode was enabled, gave less seemingly correct results than when the LIS mode was active. Aided mode helped increase the reliability of the algorithm, as it automatically rejects results that are blatantly far from reality. This is why the "positive" matches are less but more reliable.

In a parallel test, specifically the centroiding algorithm was tested on real night time sky imagery taken from a photography website, performing flawlessly on images fairly free of compression artifacts.

In the future, work should focus in improving the throughput of correct results

in order to lower the processor load. It is believed that OpenGL (the API Celestia uses to display graphics) adds some sort of distortion that might have lowered the real success rate, increasing the number of shots necessary to get one successful measurement. Therefore, the next step should include building a prototype based on the provided schematics and use the generated algorithms and databases to do real night-sky shots with known optics assembly parameters.

Appendix A

SPU Miscellaneous Parts

This memory devices were considered during the initial development of the project. However, the size of the software is yet to be determined, therefore they will most likely be replaced and they are provided for reference purposes only.

A.1 SDRAM Memory

SDRAM will be used since the mapping of the BF533 allows having glueless external memory from 16MB to 128MB memory sizes, as opposed to only 4MB of possible SRAM, which would be probably not enough to hold image data and the star catalogues.

The part of choice is the Micron MT48LC16M16A, a 4 Meg x 16 x 4 banks synchronous dynamic memory [18] which has also been tested by JPL and has a LET of $\sim 25 MeV \cdot cm2/mg$, a saturated cross-section of 1E-4%cm^2, and a probability of failure of 1E-4% for a 6 month, 14-hour operation LEO mission. [17]

A.2 Flash Memory

Different options were explored for the flash memory component. All the Parallel access Flash memories have a size of 4 MB, that, is the size of the Async addressable memory block of the BF533 DSPs.

STMicro M29W320DB 32 Mbit

32 Mbit (4Mb x8 or 2Mb x16) automotive-rated memory based on NOR gates which, from a study by the Jet Propulsion Lab (CA, USA) [38], we know less prone to failure from radiation doses than their NAND counterparts. [39]

STMicro PSD4256G6V

PSD4256G6V is a parallel port-programmable memory which would offer the advantage of simplifying the programming process, as it can be done directly without

the intervention of the microprocessor through a JTAG interface. [40]

This memory has been tested for SEL by the JPL. It showed a threshold LET of $\sim 11MeV \cdot cm2/mg$, saturated cross-section $\sim 1E-5\ cm^2$and a SEL likelihood of 0.01% for a 6 month mission and 14 hours of operation in polar LEO. [17]

A.3 Alternative embedded option

One of the options considered, apart from the memories, was an all-inclusive component has been developed by Cambridge Signal Processing,the Minotaur BF537, sub miniature Computer On Module which features [41]

- Solderless/connectorless baseboard mating
- 600 MHZ BF537
- 32MByte PC133 SDRAM. This RAM module is the Micron MT48LC16M16 whose properties we discussed previously.
- 4MByte SPI Flash
- 10/100 Ethernet MAC and PHY onboard

This item could offer a great advantage in the Star Tracker, offering large space savings and quicker prototyping. Nevertheless, just a few of its components have undergone a SEE test.

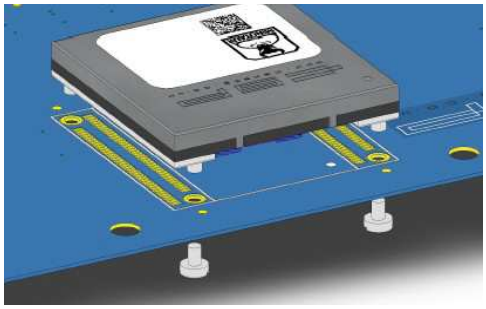


Figure A.1: Minotaur

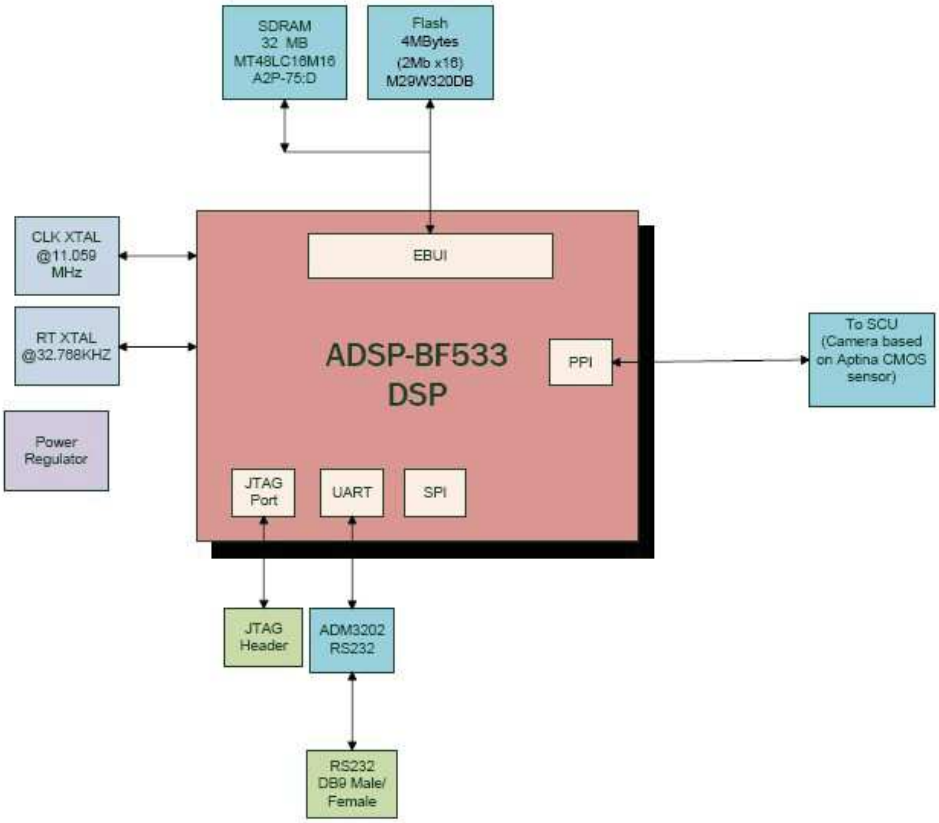
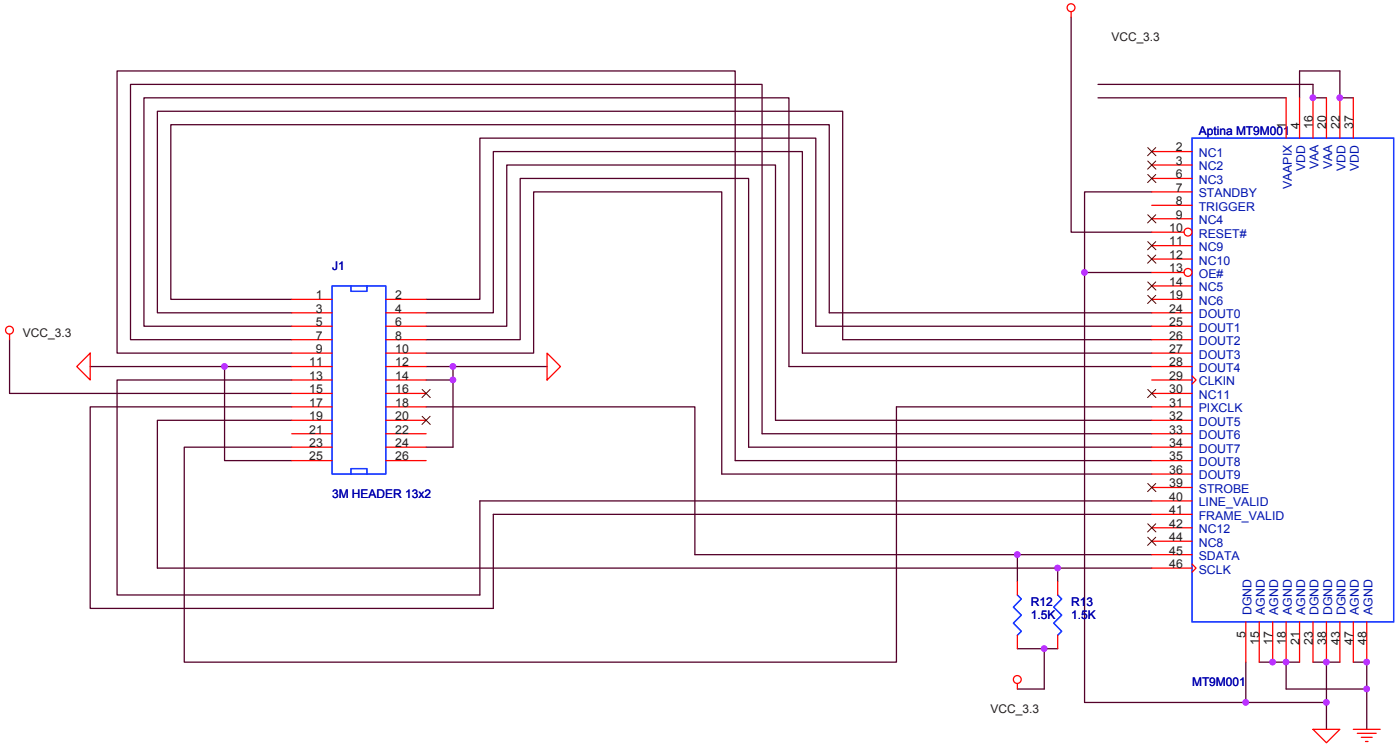


Figure A.2: Aramis Star Tracker HW Block Scheme

Appendix B

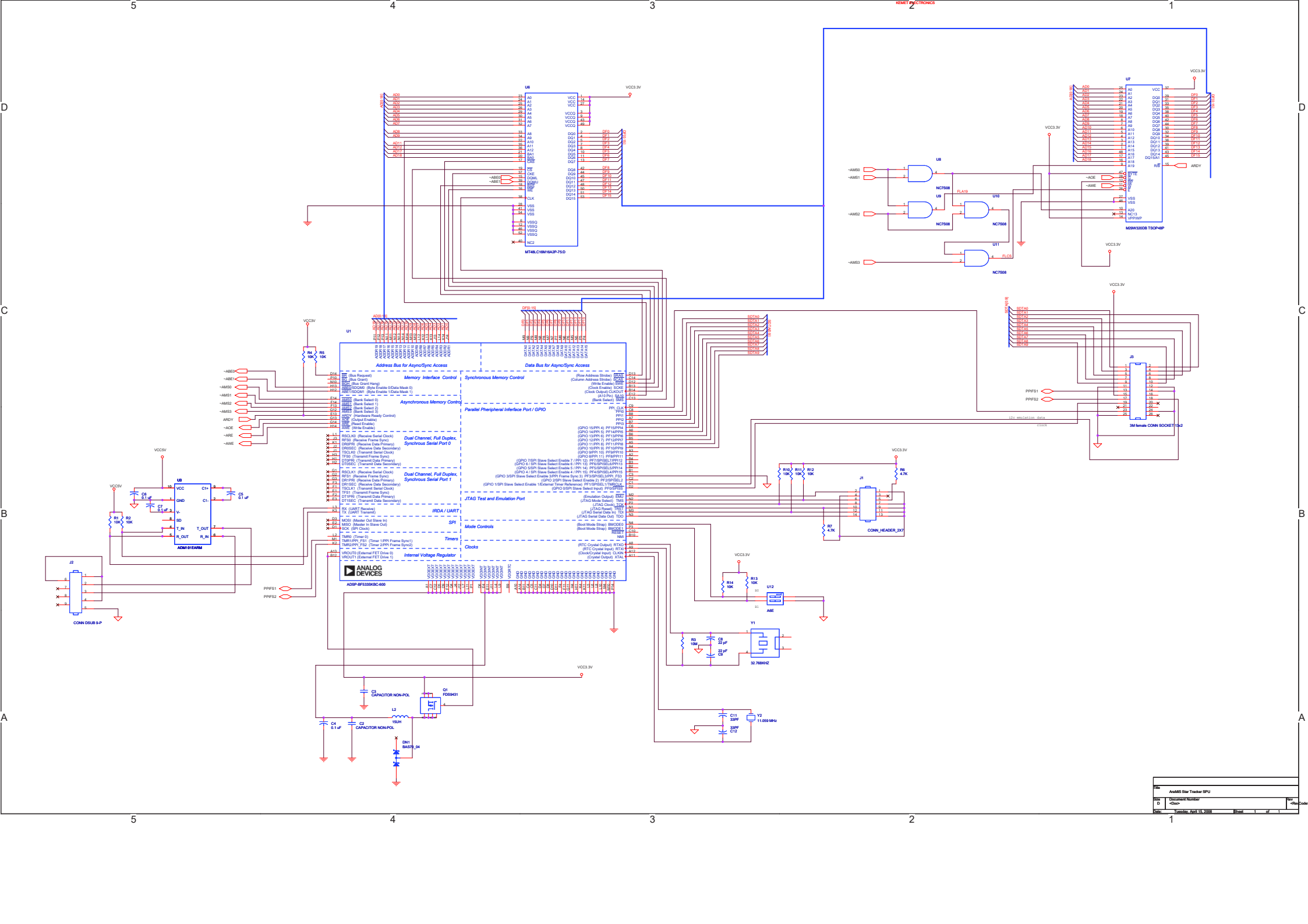
Schematics

B.1 Star Tracker Camera Unit



Title	AraMiS Star Tracker Headboard	
Size A3	Document Number <Doc>	Rev 0.01
Date: Tuesday, April 15, 2008	Sheet 1 of 1	

B.2 Star Tracker Processing Unit


U6
MT48LC16M16A2P-75:D
U7
M29W320DB TSOP48P
U8
NC7S08
U9
NC7S08
U10
NC7S08
U11
NC7S08
U1
ADSP-BF533SKBC-600
Address Bus for Async/Sync Access
Data Bus for Async/Sync Access
Memory Interface Control
Asynchronous Memory Control
Synchronous Memory Control
Parallel Peripheral Inferface Port / GPIO
Dual Channel, Full Duplex, Synchronous Serial Port 0
Dual Channel, Full Duplex, Synchronous Serial Port 1
IRDA / UART
SPI
Timers
Internal Voltage Regulator
JTAG Test and Emulation Port
Mode Controls
Clocks
U3
ADM101EARM
J2
CONN DSUB 9-P
J1
CONN_HEADER_2X7
J3
3M female CONN SOCKET 15x2
U12
AAE
Y1
32.768KHZ
Y2
11.059 MHz
Q1
FDS9431
L3
15UH
DN1
BAS70_04
C3
CAPACITOR NON-POL
C2
CAPACITOR NON-POL
VCC3.3V
VCC5V
Title
AraMIS Star Tracker SPU
Size
D
Document Number
<Doc>
Rev
<RevCode>
Date:
Tuesday, April 15, 2008
Sheet
1
of
1

Appendix C

Ground Code

The main functions of ground and in-flight software are provided. For information on the specialized functions, please refer to the attached CD.

C.1 Catalogue reader.

Converts the catalogue to a format Matlab can read.

```
#include <stdio.h>


int main ()
{
  char f;
  int lol;
  int temp;
  int j;
  int i;
  int s;

  FILE * pFile;
  FILE * pointf;

  pFile = fopen ("catalog.dat","r"); //catalogue input data
//(Yale Bright Star Catalogue)
  pointf = fopen ("catalogy.dat","w+"); //output file
```

```
   f=0;

for (j=1; j<=9110; j++) //scan all the stars
{
i=1;
do
{
    fscanf (pFile, "%c", &f); //scan each character
//of the star line
    fprintf (pointf, "%c", f); //copy characters
//into the second file
    //printf ("%c", f);
    lol=i;
    i++;
    }while((i < 197)&&(f!=10)); //until maximum number of
//characters or EOL is reached

  for (s =1; (s<=(197-lol)); s++)
        {
        fprintf(pointf, "%c", ' ');
        }

  fscanf (pFile, "%c",&temp);
  fprintf(pointf, "\n");
}

fclose (pFile);
fclose (pointf);
return 0;
}
```

C.2 Star reading and filtering

```
%This script loads the totality of stars of the Yale Bright Star
%catalogue. It then leaves only the StelleXFOV brightest stars
%for each field of view, and saves them in 6starsxfov.mat .
%Then it calculates the angle between each couple of stars
%that lie at a distance less than that of the FOV, saving the
%results to angoli.mat.
%Finally, every possible triangle of stars is calculated and
%the results are stored in triplettenuove.mat
clear all;

dofiltering=1;%To bypass star filtering
doangles=1;%to bypass angle calculation
dotriangles=1;%to bypass triangle creation


% dofiltering = input('Do star filtering? 1=yes/0=load from file [1]: ');
% if isempty(dofiltering)
%     dofiltering=1;
% end
%
% doangles = input('Do angle calculation? 1=yes/0=load from file [1]: ');
% if isempty(doangles)
%     doangles = 1;
% end
%
% dotriangles = input('Do triangle creation? 1=yes/0=load from file [1]: ');
% if isempty(dotriangles)
%     dotriangles=1;
% end

magnitude_filter=6;%faintest star to take into account
StelleXFOV=6;%minimum number of stars per FOV

FOVtemp=dms2degrees([21 32 4.7]);

FOV=deg2rad(FOVtemp);


fprintf('Loading stars from file...\n')
[hrnyale, vis_magyale, rayale, decyale]=getstars(magnitude_filter);
fprintf('Done.\n')




subplot(2,2,1);
plot(rad2deg(rayale),rad2deg(decyale),'LineStyle','none','Marker','+');
xlabel('Right Ascension');
ylabel('Declination');
title('Stelle del Bright Star Catalog')


[iyale,jyale,kyale]=sph2cart(rayale,decyale,1);
```

```
if (dofiltering)
  fprintf('Filtering best stars...\n')
  [hrn, vis_mag, i,j,k]=pulisci(hrnyale, vis_magyale, iyale, jyale, kyale, FOV,↙
StelleXFOV);
  hrn=hrn';
  vis_mag=vis_mag';
  i=i';
  j=j';
  k=k';
  fprintf('Done.\n')
else
  fprintf('Filtering skipped. Loading best stars from file...\n')
  load ('6starsxfov.mat');
  fprintf('Done.\n')
end



[ralol,declol,rlol]=cart2sph(iyale,jyale,kyale);
[ra2,dec2,r2]=cart2sph(i,j,k);

subplot(2,2,1);
plot(rad2deg(ralol),rad2deg(declol),'LineStyle','none','Marker','+');
xlabel('Right Ascension')
ylabel('Declination')
title('Stelle Catalogo')


subplot(2,2,2);
plot(rad2deg(ra2),rad2deg(dec2),'LineStyle','none','Marker','+');
xlabel('Right Ascension')
ylabel('Declination')
title('Stelle ottimizzate')

figure;

plot3(rad2deg(ra2),rad2deg(dec2),vis_mag,'LineStyle','none','Marker','+')


save('6starsxfov.mat', 'hrn', 'vis_mag', 'i','j','k');



lookupt=zeros(max(hrn),1);




for lc=1:1:max(hrn)
    lol=find(hrn==lc);
```

```
    if (any(lol))
        lookupt(lc)=find(hrn==lc);

    end
end



vectors=[i,j,k];



if(doangles)
fprintf('Building angle database...\n')
% for hrnscan1=1:1:int16(max(hrn))
%
%     if (lookupt(hrnscan1))
%         for hrnscan2=1:1:int16(max(hrn))
%             if (hrnscan1~=hrnscan2)&&(lookupt(hrnscan2))
% %                  if (hrnscan1==2491)
% %                     hrnscan1
% %                 end
% %
% %                 if ((hrnscan1==2491)&&(hrnscan2==2618)) %Debug
% %                     hrnscan1
% %                 end
%
%                 angolia(hrnscan1,hrnscan2)=acos( dot(vectors(lookupt(hrnscan1),:),↙
vectors(lookupt(hrnscan2),:)) );
%
%             end
%         end

popul=zeros(int16(max(hrn)),3);


%populate list of vectors

[len,dummy]=size(vectors);

angolistg1=zeros(len,len); %matrix with no meaningful index

for scan1=1:1:len
    factor=repmat(vectors(scan1,:),len,1);
    angolistg1(:,scan1)=acos( dot(vectors,factor,2));

end

angolistg1=~eye(len,len).*angolistg1;

angolistg2=sparse(max(hrn),len);%matrix with hrn index in rows


for scan1=1:1:int16(max(hrn))
    if (lookupt(scan1))
```

```
        angolistg2(scan1,:)=angolistg1(lookupt(scan1),:);
    end
    scdoub=double(scan1);
    if (scdoub./1000==round(scdoub./1000))
        fprintf('*')
    end
end


angoli=sparse(max(hrn),max(hrn));%matrix with hrn index in rows and columns

for scan1=1:1:int16(max(hrn))
    if (lookupt(scan1))
        angoli(:,scan1)=angolistg2(:,lookupt(scan1));
    end
    scdoub=double(scan1);
    if (scdoub./1000==round(scdoub./1000))
        fprintf('*')
    end
end
fprintf('*\n')


%
%   if (hrnscan1==round(20*max(hrn)/100))
%       fprintf('*\n')
%   end
%   if (hrnscan1==round(40*max(hrn)/100))
%       fprintf('**\n')
%   end
%   if (hrnscan1==round(60*max(hrn)/100))
%       fprintf('***\n')
%   end
%   if (hrnscan1==round(80*max(hrn)/100))
%       fprintf('****\n')
%   end
%   if (hrnscan1==round(100*max(hrn)/100))
%       fprintf('***** Done.\n')
%   end



save('angoli.mat', 'angoli');



else
    load('angoli.mat');
end



if (dotriangles)

count=1;
```

```
taccu=0;
triplette=zeros(10000,5);
tripsum=zeros(10000,1);

fprintf('Building planar triangle database...\n')
for p=1:1:size(hrn)
    tic;
    p./size(hrn);
    primolvl=zeros(100,4); %cambiare 100
    primolvl=getneighbors2([double(hrn),i,j,k],FOV,hrn(p),angoli); %ottieni lista di elementi con angolo minore di FOV/2 (tranne la posz p)

    [plm,pln]=size(primolvl);



    for q=1:1:plm
%         q./plm
        secondolvl=getneighbors2(primolvl,FOV,primolvl(q,1),angoli);%ottieni sottolista di elementi con angolo minore di FOV/2 (tranne q)

        [slm,sln]=size(secondolvl);
        for s=1:1:slm

            if (~checktrip2(triplette,hrn(p),primolvl(q,1),secondolvl(s,1),tripsum)) %controlla presenza della tripletta

                ip=i(p);
                jp=j(p);
                kp=k(p); %il livello 0 ha l'indice p che si riferisce all'elenco principale

                indexq=lookupt(primolvl(q,1));
                indexs=lookupt(secondolvl(s,1));

                iq=i(indexq);
                jq=j(indexq);
                kq=k(indexq); %i livelli 1 e 2 devono cercare l'indice attraverso l'hrn

                is=i(indexs);
                js=j(indexs);
                ks=k(indexs);

                %calcolo del momento e l'area


                a=norm([ip,jp,kp]-[iq,jq,kq]);
                b=norm([iq,jq,kq]-[is,js,ks]);
                c=norm([is,js,ks]-[ip,jp,kp]);
                esse=0.5*(a+b+c);
                area=sqrt(esse*(esse-a)*(esse-b)*(esse-c));

                momentum=area*(a^2+b.^2+c^2)/36;
```

```
                triplette(count,:)=[area, momentum, double(hrn(p)),double(hrn(indexq)),↙
double(hrn(indexs))];
                tripsum(count)=double(hrn(p))+double(hrn(indexq))+double(hrn(indexs));
                count=count+1;
            end

        end
    end

    perc=p./size(hrn);
    perc(1)=perc(1)*100
    fprintf('percent of stars scanned.\n')
    taccu=taccu+toc
    eta=((100-perc(1)).*taccu./perc(1))./(60.*60)
    pause(0.1)
end

fprintf('Done.\n')



save('triplettenuove.mat', 'triplette');

else
    load('triplettenuove.mat')

end
```

C.3 k-vector tags generation

```
%This file generates the k-vectors for both the areas and moments of
%the planar triangles. Vector tvector--- are the indexes to reach vector
%ind--- from the curve.

clear all;
load triplettenuove.mat;
load hrn_vis_mag_i_j_k.mat;
load bound_coef.mat;
load hrnLUT.mat;
load angoli.mat;



%make an indexed version of the values and order them
supindxd=sortrows([triplette(:,1),(1:1:size(triplette(:,1),1))'],1);
%separate the values from the indexes
sup=supindxd(:,1);
indarea=supindxd(:,2);



%get x vector
xa=(1:1:size(sup,1));

%assign known values (lowmost, uppermost) for the approximation function
y1a=sup(1);
x1a=1;
ka=max(sup);
ha=size(xa,2);

%get p(distance of focus)
pa=subs(solve('(y1a-ka)^2=4*pa*(x1a-ha)','pa'));

%solve the generic parabola equation for y, just to plot the graph
eqyarea=solve('(ya-ka)^2=4*pa*(xa-ha)','ya')
%and substitute, just to plot the graph too
yareas=subs(eqyarea);

%now solve and substitute to get the vector of tees
eqxarea=solve('(ya-ka)^2=4*pa*(xa-ha)','xa')
%result: x=1/4*(y^2-2*y*k+k^2+4*p*h)/p
%wrong%ya=sup(1:1:size(sup,1));
ya=sup;


%we take the runded value of the evaluation of x in the function as an
%index. Then we obtain the y value for the approx. function. We look for
%this value in the real function (sup), to obtain the index into ind
tic;

tvectorindexes=round(subs(eqxarea));


tvectormoment( tvectorindexes )=   bsearch ( sup, yareas(2, tvectorindexes )' );
toc;
```

```
plot(1:1:size(xa,2),sup,1:1:size(xa,2),yareas(2,:));

legend('Planar Triangle Areas','K-Vector curve approximation')

hold off;


%%%%
figure


%make an indexed version of the values and order them
supindxd=sortrows([triplette(:,2),(1:1:size(triplette(:,2),1))'],1);
%separate the values from the indexes
sup=supindxd(:,1);
indmoment=supindxd(:,2);



%get x vector
xm=(1:1:size(sup,1));

%assign known values (lowmost, uppermost) for the approximation function
y1m=sup(1);
x1m=1;
km=max(sup);
hm=size(xm,2);

%get p(distance of focus)
pm=subs(solve('(y1m-km)^2=4*pm*(x1m-hm)','pm'));

%solve the generic parabola equation for y, just to plot the graph
eqymoment=solve('(ym-km)^2=4*pm*(xm-hm)','ym')
%and substitute, just to plot the graph too
ymoments=subs(eqymoment);

%now solve and substitute to get the vector of tees
eqxmoment=solve('(ym-km)^2=4*pm*(xm-hm)','xm')
%result: x=1/4*(y^2-2*y*k+k^2+4*p*h)/p
%wrong%ym=sup(1:1:size(sup,1));
ym=sup;

%we take the runded value of the evaluation of x in the function as an
%index. Then we obtain the y value for the approx. function. We look for
%this value in the real function (sup), to obtain the index into ind
tic;
tvectormoment( round(subs(eqxmoment)) )=   bsearch ( sup, ymoments(2, round(subs↙
(eqxmoment)') ) );
```

```
toc;

plot(1:1:size(xm,2),sup,1:1:size(xm,2),ymoments(2,:));

legend('Planar Triangle Moments','K-Vector curve approximation')

%save kvectors, equation of the curves and necessary curve parameters
save('kvectors.mat', 'indarea', 'tvectorarea','indmoment', 'tvectormoment',↙
'eqxarea', 'eqxmoment', 'km', 'pm', 'hm','ka','pa','ha')
```

C.4 Buckingham Fuller Polyhedron tags compilation

```
clear all;
load triplettenuove.mat;
load hrn_vis_mag_i_j_k.mat;
load bound_coef.mat;
load hrnLUT.mat;
load angoli.mat;

FV=sphere_tri('ico',2,1,0);
lighting phong; shading interp;
 figure;
patch('vertices',FV.vertices,'faces',FV.faces,'facecolor',[1 0 0],'edgecolor',[.2 .2
.6]);
axis off; camlight infinite; camproj('perspective');
alpha(.7);
hold on;

ang=rad2deg (acos (dot(FV.vertices(FV.faces(1,1),:),FV.vertices(FV.faces(1,2),:))))

maximumFOV=sqrt(ang^2-(ang/2)^2)+sqrt(ang^2-(ang/2)^2)/2


added=zeros(80,2);



v1=FV.vertices(FV.faces(:,1),:);
v2=FV.vertices(FV.faces(:,2),:);
v3=FV.vertices(FV.faces(:,3),:);

mean([v1(1,:);v2(1,:);v3(1,:)],1)

FV.facecenters=mean(cat(3,v1,v2,v3),3);

for contafaccia=1:1:size(FV.facecenters,1)

    FV.facecenters(contafaccia,:)=FV.facecenters(contafaccia,:)./norm(FV.facecenters
(contafaccia,:));

end


%create the list of adjacent faces
for contafaccia=1:1:size(FV.facecenters,1)

    adjcount=1;
    for contafaccia2=1:1:size(FV.facecenters,1)
        if contafaccia~=contafaccia2

            sizeinter=size(intersect(FV.faces(contafaccia,:),FV.faces
(contafaccia2,:)),2);



            if(sizeinter~=0)
```

```
                commonvert(contafaccia,adjcount)=sizeinter;

                adjacent(contafaccia,adjcount)=contafaccia2;
                adjcount=adjcount+1;

            end
        end
    end

    localadj=adjacent(contafaccia,:)';
    localadj=localadj(localadj~=0);


    clf;
    lighting phong; shading interp;
    patch('vertices',FV.vertices,'faces',FV.faces,'facecolor',[1 0 0],'edgecolor',[.2
.2 .6]);
    alpha(.5);
    patch('vertices',FV.vertices,'faces',FV.faces(localadj,:),'facecolor',[0 0
1],'edgecolor',[.2 .2 .6]);
    axis off; camlight infinite; camproj('perspective');

end




for count=1:1:size(triplette,1)

    %calculate the centroid of the triangle
    xx=mean([i(lookupt(triplette(count,3))),i(lookupt(triplette(count,4))),i(lookupt
(triplette(count,5)))]);
    yy=mean([j(lookupt(triplette(count,3))),j(lookupt(triplette(count,4))),j(lookupt
(triplette(count,5)))]);
    zz=mean([k(lookupt(triplette(count,3))),k(lookupt(triplette(count,4))),k(lookupt
(triplette(count,5)))]);
    %normalize the vector
    centroid_vector=[xx,yy,zz]/norm([xx,yy,zz]);



    %calculate the angle of the vector against all the possible half-points
    angles=acos(FV.facecenters*centroid_vector');

    %index and then order the angle list
    indexed_angles=[angles,(1:1:size(angles,1))'];
    indexed_angles=sortrows(indexed_angles,1);
    %take the nearest vector
    face=indexed_angles(1,2);

%     clf;
%     patch('vertices',FV.vertices,'faces',FV.faces,'facecolor',[1 0 0],'edgecolor',
[.2 .2 .6]);
%     axis off; camlight infinite; camproj('perspective');
```

```
%     alpha(.7);
%     hold on;
%     vectarrow([0,0,0],FV.facecenters(face,:));
%     hold on;
%     vectarrow([0,0,0],centroid_vector);
%     hold off;

    if ((count/1000)-round(count/1000))==0
        count
    end

    tag(count)=face;


end


%load isocahedrontag.mat;

indexedtag=sortrows([tag',(1:1:(size(tag,2)))'],1);


finaltags=[];
for contafac=1:1:max(tag)

    localtags=indexedtag(indexedtag(:,1)==contafac,2);
    finaltags=[finaltags;localtags]; %the list of ordered triplet indexes for each↙
face
    tagmarkers(contafac)=size(finaltags,1); %the list of delimiting markers for each↙
face


end




vert=FV.vertices;
faces=FV.faces;
centers=FV.facecenters;

save ('isocahedron.mat', 'vert', 'faces', 'centers', 'adjacent');
save('isocahedrontag.mat', 'tag');
save ('taglists.mat', 'finaltags', 'tagmarkers');
```

Appendix D

Simulator scripts

D.1 Celestia Initializer

init.cel

```
{
unmarkall{}

#select{object "hubble"}
#follow{}


#goto{}
lookback{}
setvisibilitylimit {magnitude 6.0}
set{name "FOV" value 21.534638888888890}

#setposition{

#       base    [ -1.323515988135877e-005
-1.270447054050724e-009 8.750235516536438e-006 ]
#       offset [ -2.606422022655153e-016
5.421010862427522e-020 -4.954261827172513e-016 ]
#}


select { object "HIP 33316" }
center {  }
wait {duration 5}
```

```
rotate {
        duration 3
        rate 10
        axis [0 1 0]
}



rotate {
        duration 3
        rate 10
        axis [1 0 0]
}

#wait {duration 5}

rotate {
        duration 3
        rate 10
        axis [0 0 1]
}

#wait {duration 10}

}
```

D.2 Image reader/attitude reference obtainer

quaternionWritefile.celx

```
-- Title: Display current RA/Dec for observer

KM_PER_LY = 9460730472580.8
KM_PER_AU = 149597870.7
PI = math.pi
degToRad = PI / 180;
J2000Obliquity = 23.4392911 * degToRad
fov=math.rad(21.534638888888890);
```

```
LOOK  = celestia:newvector(0, 0, -1)
earth = celestia:find("Sol/Earth")

-- Convert coordinates from cartesian to polar:
xyz2rtp = function (x, y, z)
   local r = math.sqrt(x * x + y * y + z * z)
   local phi = math.atan2(y, x)
   local theta = math.atan2(math.sqrt(x * x + y * y), z)

   return r, theta, phi
end

-- Return current distance of observer from Earth:
getR = function (obs)
   return earth:getposition():distanceto(obs:getposition())
end

-- Return current RA, Dec for observer:
getRADec = function (obs)
   local base_rot = celestia:newrotation
 (celestia:newvector(1, 0, 0), -J2000Obliquity)

   local startracker_rot =
celestia:newrotation(celestia:newvector(0, 1, 0), -1.5708)
   local rot = obs:getorientation() * base_rot
   local attitude_rot =  rot * startracker_rot
   local look = rot:transform(LOOK):normalize()
   local r, theta, phi = xyz2rtp(look.x, look.z, look.y)
   local phi = math.mod(720 - math.deg(phi), 360)
   local theta = math.deg(theta)
   if theta > 0 then
      theta = 90 - theta
   else
      theta = (-90 - theta)
   end
   return phi, theta, attitude_rot, look.x, look.y, look.z
end

km2Unit =
   function(km)
      local sign, value, units
```

```
        if km < 0 then sign = -1 else sign = 1 end
        km = math.abs(km)
        if km > 1e12 then
            value = km / KM_PER_LY
            units = "ly"
        elseif km >= 1e8 then
            value = km/KM_PER_AU
            units = "AU"
        else
            value = km
            units = "km"
        end;
        return string.format("%.2f", sign * value).." ".. units
    end

deg2dms = function(deg)
    local   a = math.abs(deg)
   local   d = math.floor(a)
    local   r = (a - d) * 60
    local   m = math.floor(r)
    local   s = (r - m) * 60
    if deg < 0 then d = -d end
    return string.format("%0.0fd %02.0f' %2.0f''",d,m,s)
end

deg2hms =   function(deg)
    local   a = math.abs(deg / 15)
   local   d = math.floor(a)
    local   r = (a - d) * 60
    local   m = math.floor(r)
    local   s = (r - m) * 60
    return string.format("%0.0fh %02.0fm %2.0fs", d, m, s)
end

quat2euler =   function(qtrn)
    local   q4 = qtrn.w
    local   q2 = -qtrn.x
    local   q3 = qtrn.y
    local   q1 = qtrn.z
    local   yaw =
math.atan2 (2*(q1*q2+q4*q3),((q4^2  + q1^2 - q2^2- q3^2)))
```

```
    local    pitch=
math.asin(-2*(q1*q3-q4*q2))
    local    roll=
math.atan2(2*(q4*q1+q2*q3),(q4^2  - q1^2 - q2^2+ q3^2))
    yawdeg = yaw/degToRad
    pitchdeg = pitch/degToRad
    rolldeg = roll/degToRad

    return yawdeg, pitchdeg, rolldeg
end

while true do




    obs = celestia:getobserver()
    obs:setfov(fov)
    obsR = getR(obs)
    obsRA, obsDec, quat, xobs, yobs, zobs = getRADec(obs)
    yaw, pitch, roll = quat2euler(quat)
    celestia:print(obsR..' '..obsRA..' '..obsDec)
    if obsR >= 0 then
       -- Display geocentric coordinates for observer:
       obsRStr = km2Unit(obsR)
       obsRAStr = deg2hms(obsRA)
       obsDecStr = deg2dms(obsDec)

            celestia:print
("\nRA: "..obsRAStr.."\nDec: "..obsDecStr.."\nq4:
"..quat.w.."\nq1: "..-quat.z.."\nq2: "..-quat.x.."\nq3:
 "..quat.y.."\nRoll: "..-roll.."\nPitch: "..pitch.."\nYaw:
"..yaw.."\nX: "..xobs.."\nY: "..-zobs.."\nZ: "..yobs.."\nCelX:
"..xobs.."\nCelY: "..yobs.."\nCelZ: "..zobs, 1, -1, -1, 1, 19)

       --celestia:print("\nRA: "..obsRAStr.."\nDec:
"..obsDecStr.."\nq4: "..quat.w.."\nq1: "..quat.z.."\nq2:
"..-quat.x.."\nq3: "..quat.y.."\nRoll: "..roll.."\nPitch:
"..pitch.."\nYaw: "..yaw.."\nx: "..xobs.."\ny: "..yobs.."\nz:
"..zobs, 1, -1, -1, 1, 20)
```

```
            --celestia:print("\nRoll: " .. roll .. "\nPitch:
" .. pitch .. "\nYaw: " .. yaw, 1, 1, -1, 1, 8)
    end



    wait(0)
end
```

Appendix E

In-flight Code

```
%This code analyzes an sequence of images, finding an identifiable triangle
%of stars.


%% Load required databases and constants
clear all;




load triplettenuove.mat;
load hrn_vis_mag_i_j_k.mat;
load bound_coef.mat;
load hrnLUT.mat;
load angoli.mat;
load kvectors.mat;
load isocahedrontag.mat;
load xyzlive.mat;
load taglists.mat;
load isocahedron.mat;


global isocahedron;
global vert;
global faces;
global centers;
global adjacent;
global tag;

sensorsz=582;%688;%6.8608; %sensor or celestia screenshot size (short side)
pixelradapprox=0.3759/sensorsz;
%pixelradapprox=0.3759/603;


firstValidFrame=2;   %just because Celestia takes useless shots on the first few frames

imload=2;

%% Load image sequences
if imload==1
    fprintf('Loading Sequence.\n')
    obj = mmreader('vidverysmall.avi');
    %obj = mmreader('3vid1xscript21_32_47.avi');
    %obj = mmreader('vidfull2.avi');
    %obj = mmreader('1vid8xSyncWithISS2538162.avi');
    %obj = mmreader('060901_Startracker_Auto_Imaging_v4.wmv');
    numFrames = get(obj, 'numberOfFrames');
    FrameRate = get(obj, 'FrameRate');


    loadingframe=1;


    for loadingframe=1:1:numFrames
        imagen =rgb2gray( read(obj,loadingframe));
        %imagen(510:576,180:530)=zeros(67,351); %to cover the celestia text output
```

```
        imagen=imnoise(imagen, 'gaussian',0,0.0001);
        images(:,:,loadingframe)=imagen(:,:);
%         count=count+1
        loadingframe
    end

    save ('preloadedimages.mat','images','numFrames','FrameRate','obj');
elseif imload==2 %load screenshots from the celestia dir

    fileFolder = fullfile('C:', 'Programmi', 'Celestia');
    dirOutput = dir(fullfile(fileFolder,'screenshot-skyscan-*.png'));
    fileNames = {dirOutput.name}';
    numFrames = numel(fileNames);

    for p = 1:numFrames
        images(:,:,p) = rgb2gray(imread(fileNames{p}));
        images(:,:,p)=imnoise(images(:,:,p), 'gaussian',0,0.0001);
    end

    %overides the default values (we know that the size of the screenshot==
    %==full FOV

    sensorsz=size(images(:,:,1),1);
    pixelradapprox=0.3759/sensorsz;


else
    load preloadedimages.mat

end
sampno=0;
accufrms=[0,0,0,0,0];


%Test with one image

%imagen =imnoise(rgb2gray( imread('canismajorsans.jpg')), 'gaussian',0,0.0001);
%imagen =rgb2gray( imread('canismajorsans.jpg'));
fprintf('Done.\n')


%statistics variables
averagingtime=zeros(numFrames,1);
finalfilteringtime=zeros(numFrames,1);
tripletidtime=zeros(numFrames,1);
variancestime=zeros(numFrames,1);
triangleanalysistime=zeros(numFrames,1);
dimstarremovaltime=zeros(numFrames,1);
spikeremovaltime=zeros(numFrames,1);
pivotingtime=zeros(numFrames,1);
whoiswho=zeros(numFrames,1);

pivotcount=zeros(numFrames,1);
```

```
% ambiguous=0;
% nothing=0;
% unique=0;
% total=0;

ambiguous=zeros(500,1);
nothing=zeros(500,1);
unique=zeros(500,1);
correct=zeros(500,1);
total=zeros(500,1);

 last=zeros(3,1);


sampletemp=images(:,:,1);
imsizetemp=size(sampletemp);
[x y] = meshgrid(1:imsizetemp(2), 1:imsizetemp(1));
FOVmask=sqrt((x-imsizetemp(2)/2).^2 + (y-imsizetemp(1)/2).^2)<sensorsz/2; %mask the stars not belonging to the FOV


 for (tweakcount=10:1:10)
%% Begin Loop
for fri=1:1:numFrames

    showCorrectFlag=0;
    clear hrntripletta;

    sample=images(:,:,fri);



    imsize=size(sample);%absolute size of image


    noisysample2show=sample;

    tic
    [mask,accufrms]=avgthreshold(sample,accufrms,imsize);%% Averaging to find threshold and mask noise away
    averagingtime(fri,1)=toc;

    tic
    mask2=spkremoval1(mask,imsize);%% Spike removal Method 1
    %mask2=spkremoval2(mask,imsize);%% Spike removal Method 2 (faster)

    mask2=mask2.*FOVmask;

    sample(~mask2) = 0;%apply spike removal mask first
    spikeremovaltime(fri,1)=toc;

    tic
    pos=locatestars(sample);% Dim star removal and bright star determination
    dimstarremovaltime(fri,1)=toc;
```

```
%% Reference (sun sensor) face


    faces2look4=identifyFace(xyzlive(fri,:),centers,adjacent); %identifies all the↙
adjacent faces



%% Planar triangle analysis

    tic
    FOV=deg2rad(dms2degrees([21 32 4.7]));
    pp=1;%6.7E-3;%pixel pitch



    %variance of the angle???
    %varang=((0.13)*pixelradapprox)^2;%best one so far: 0.13
    varang=((0.01+2.5*tweakcount*0.01)*pixelradapprox)^2;
    %varang=((0.01+16*0.01)*pixelradapprox)^2;

    %varang=5.3291e-015


    [ipmt(1),varmom(1),areat(1),vararea(1),swt(1),sat(1),wat(1),il(:,1),jl(:,1),kl(:,↙
1)]=planart(sample,pos(1:3,:),FOV,sensorsz,pp,varang);
    triangleanalysistime(fri,1)=toc;
    tic
    areaMaxOffset=3*sqrt(vararea(1));

    momMaxOffset=3*sqrt(varmom(1));

%% Identify triplet
    %triplette --is-> area momentum hrn1 hrn2 hrn3 (CHANGE THIS FOR A
    %K-VECTOR
    %ALGORITHM!!!)
    tic
    %k-vector algorithm

    %this function works WITHOUT sun reference:
    %hrntripletta(:,:,1)=idtriplet2(triplette,areat(1),areaMaxOffset(1),ipmt(1),↙
momMaxOffset(1),tvectorarea,tvectormoment,indarea,indmoment, pm, hm, km, ka,pa, ha);

    %this function works WITH the sun reference:
    hrntripletta(:,:,1)=idtriplet3(triplette,areat(1),areaMaxOffset(1),ipmt(1),↙
momMaxOffset(1),tvectorarea,tvectormoment,indarea,indmoment, pm, hm, km, ka,pa, ha,↙
faces2look4, finaltags, tagmarkers);



    %slow, non k-vector algorithm
    %[hrntripletta(:,:,1),absmags]=idtriplet1(triplette,areat(1),areaMaxOffset(1),ipmt↙
(1),momMaxOffset(1),lookupt,vis_mag);
```

```
    tripletidtime(fri,1)=toc;


%% final filtering method 3
    pivotlimit=3;


    starXYPositions=zeros(3,2,pivotlimit+1);

    starXYPositions(:,:,1)=pos(1:3,2:3);




    uniquefound=0;
    extpivot=4;
    count=2;
    notfound=0;
    hrntriplettaaccu=hrntripletta(:,:,1);


    while (~uniquefound)&&(count-1<=pivotlimit)
        if count-1==1
            [ipmt(count),varmom(count),areat(count),vararea(count),swt(count),sat↙
(count),wat(count),il(:,count),jl(:,count),kl(:,count)]=planart(sample,[pos(1:2,:);pos↙
(extpivot,:)],FOV,sensorsz,pp,varang);
            starXYPositions(:,:,count)=[pos(1:2,2:3);pos(extpivot,2:3)];
        elseif count-1==2
            [ipmt(count),varmom(count),areat(count),vararea(count),swt(count),sat↙
(count),wat(count),il(:,count),jl(:,count),kl(:,count)]=planart(sample,[pos(1,:);pos↙
(3,:);pos(extpivot,:)],FOV,sensorsz,pp,varang);
            starXYPositions(:,:,count)=[pos(1,2:3);pos(3,2:3);pos(extpivot,2:3)];
        else
            [ipmt(count),varmom(count),areat(count),vararea(count),swt(count),sat↙
(count),wat(count),il(:,count),jl(:,count),kl(:,count)]=planart(sample,[pos(1:2,:);pos↙
(extpivot,:)],FOV,sensorsz,pp,varang);
            starXYPositions(:,:,count)=[pos(1:2,2:3);pos(extpivot,2:3)];
%             [ipmt(count),varmom(count),areat(count),vararea(count),swt(count),sat↙
(count),wat(count)]=planart(sample,[pos(2:3,:);pos(extpivot,:)],FOV,sensorsz,pp,↙
varang);
%             starXYPositions(:,:,count)=[pos(2:3,2:3);pos(extpivot,2:3)];
        end
        areaMaxOffset(count)=3*sqrt(vararea(count));
        momMaxOffset(count)=3*sqrt(varmom(count));
        %k-vector algorithm

        %this function works WITHOUT sun reference:
        %triptemp=idtriplet2(triplette,areat(count),areaMaxOffset(count),ipmt(count),↙
momMaxOffset(count),tvectorarea,tvectormoment,indarea,indmoment, pm, hm, km, ka,pa,↙
ha);
        %this function works WITH sun reference:
        triptemp=idtriplet3(triplette,areat(count),areaMaxOffset(count),ipmt(count),↙
momMaxOffset(count),tvectorarea,tvectormoment,indarea,indmoment, pm, hm, km, ka,pa, ha,↙
faces2look4,finaltags,tagmarkers);
```

```
        %slow, non k-vector algorithm
        %[triptemp,dummy]=idtriplet1(triplette,areat(count),areaMaxOffset(count),ipmt↙
(count),momMaxOffset(count),lookupt,vis_mag)

        %if triptemp~=[0,0,0,0,0]
            hrntriplettaaccu=searchtree2(hrntriplettaaccu,triptemp);%look for triplets↙
that have at least one common pair (binary search)


            if any(hrntriplettaaccu,1)
                if size(hrntriplettaaccu,1)==1
                    uniquefound=1;
                end

            else
                notfound=1;
            end


        %end




        extpivot=extpivot+1;
        count=count+1;
    end



    if (~uniquefound)&&(~notfound)
       %ambiguous result!
       ambiguous(tweakcount)=ambiguous(tweakcount)+1;
    end

    if (uniquefound)&&(~notfound)
        unique(tweakcount)=unique(tweakcount)+1;
        pivotcount(fri,1)=count-2;%pivot stats
    end



    if (notfound)
        %no results found
        nothing(tweakcount)=nothing(tweakcount)+1;
    end

    total(tweakcount)=total(tweakcount)+1;

    pivotingtime(fri,1)=toc;
%% final filtering method 2
```

```
% %change this for a pivot algorithm!!!
%     tic
%     [finstars,poll]=selecttriplet(hrntripletta,angoli,lookupt,swt,sat,wat,vis_mag);
%     finalfilteringtime(fri,1)=toc;
    %% Find stars coordinates from the database
    %(CHANGE THIS FOR A K-VECTOR ALGORITHM!!!)

%     stindex=[0,0,0];
%     for triploop=1:1:3
%         stindex(triploop)=find(hrn==hrntripletta(:,triploop)); %find the indexes for the three hrns
%     end




    %starone=find

    tic
      %% Identify individual stars


     if (uniquefound)&&(~notfound) %only if we have identifyied one triplet

        score=zeros(3); %initialize score array
        brSortCat3=zeros(3,2); %initialize the

        %now order HRNs by brightness
        brSortCat3(1:3,1)=[hrntriplettaaccu(3);hrntriplettaaccu(4);hrntriplettaaccu(5)];
        brSortCat3(1:3,2)=[vis_mag(lookupt(hrntriplettaaccu(3)));vis_mag(lookupt(hrntriplettaaccu(4)));vis_mag(lookupt(hrntriplettaaccu(5)))];
        sortrows(brSortCat3,-2);

        %we will work more confortably by giving a name to the catalogue stars
        cb=brSortCat3(1,1);  %brightest
        cm=brSortCat3(2,1);  %medium brightness
        cd=brSortCat3(3,1);  %dimmest

        %now, who is who? : voting algorithm

        %1) Brightness criterium

        %the weight given by a brightness correspondance will depend on the
        %variation coefficient, which is the standard deviation divided by
        %the average of the brightness. It will be the same for every star, so
        %it doesn't have a meaning right now, but it will, when we have the
        %votes from the angle comparison

        %get variation coeff.
        bVrtCff=std(pos(1:3,1))/mean(pos(1:3,1));


        score=eye(3).*bVrtCff; %assign votes
```

```
        bVrtCff1=min( [abs(pos(2,1)-pos(1,1)),abs(pos(3,1)-pos(1,1))])/mean(pos(1:↙
3,1));
        bVrtCff2=min( [abs(pos(1,1)-pos(2,1)),abs(pos(3,1)-pos(2,1))])/mean(pos(1:↙
3,1));
        bVrtCff3=min( [abs(pos(1,1)-pos(3,1)),abs(pos(2,1)-pos(3,1))])/mean(pos(1:↙
3,1));

        score(1,1)=bVrtCff1;
        score(2,2)=bVrtCff2;
        score(3,3)=bVrtCff3;


        %2) Angle criterium

        %retrieve the vectors of each catalogue star
        bVect=[i(lookupt(cb)),j(lookupt(cb)),k(lookupt(cb))];
        dVect=[i(lookupt(cd)),j(lookupt(cd)),k(lookupt(cd))];
        mVect=[i(lookupt(cm)),j(lookupt(cm)),k(lookupt(cm))];

        %get the angles between catalogue stars
        bmAng=acos(dot(bVect,mVect));
        bdAng=acos(dot(bVect,dVect));
        mdAng=acos(dot(mVect,dVect));

        %luckily, we already have the angles between measured star vectors
        %we now substract them from the catalogue angles and obtain the error
        bmErrors=abs(ones(3,1).*bmAng-[sat(1);swt(1);wat(1)]);
        bdErrors=abs(ones(3,1).*bdAng-[sat(1);swt(1);wat(1)]);
        mdErrors=abs(ones(3,1).*mdAng-[sat(1);swt(1);wat(1)]);

        %now look for the index of the minimal error, it will tell us
        %which the most similar angles are. We don't use matlab's find to
        %make it more "embeddable"
        mine=min(bmErrors);
        for angcount=1:1:3
            if bmErrors(angcount)==mine
                bmInd=angcount;
            end
        end

        mine=min(bdErrors);
        for angcount=1:1:3
            if bdErrors(angcount)==mine
                bdInd=angcount;
            end
        end

        mine=min(mdErrors);
        for angcount=1:1:3
            if mdErrors(angcount)==mine
                mdInd=angcount;
            end
        end

        %for each possibility of triangle side correspondance
%         example
```

```
%             bm|bd|md|
%          sa   *|  |  |
%          sw    | *|  |
%          aw    |  |* |
%          bmInd==1     bdInd==2  mdInd==3

        ang_score_applied=0;
        if bmInd==1 && bdInd==2 && mdInd==3
            meaS=1;%cb;
            meaW=3;%cd;
            meaA=2;%cm;
            ang_score_applied=1;

        elseif bmInd==1 && bdInd==3 && mdInd==2
               meaA=1;%cb;
               meaW=3;%cd;
               meaS=2;%cm;
               ang_score_applied=1;

        elseif bmInd==2 && bdInd==1 && mdInd==3
               meaS=1;%cb;
               meaA=3;%cd;
               meaW=2;%cm;
               ang_score_applied=1;

        elseif bmInd==2 && bdInd==3 && mdInd==1
               meaW=1;%cb;
               meaA=3;%cd;
               meaS=2;%cm;
               ang_score_applied=1;

        elseif bmInd==3 && bdInd==1 && mdInd==2
               meaA=1;%cb;
               meaS=3;%cd;
               meaW=2;%cm;
               ang_score_applied=1;

        elseif bmInd==3 && bdInd==2 && mdInd==1
               meaW=1;%cb;
               meaS=3;%cd;
               meaA=2;%cm;
               ang_score_applied=1;
        end

        if(ang_score_applied) %only if we did find nice, different angles
            %to assign star correspondances
            distanze=[bmAng,bdAng,mdAng];
            weight=std(distanze)./mean(distanze);
            score(1,meaS)=score(1,meaS)+weight;
            score(2,meaA)=score(2,meaA)+weight;
            score(3,meaW)=score(3,meaW)+weight;
        end

        winner=max(score(1,:));
        for candcount=1:1:3
            if score(1,candcount)==winner
```

```
                win1=candcount;
            end
        end

        winner=max(score(2,:));
        for candcount=1:1:3
            if score(2,candcount)==winner
                win2=candcount;
            end
        end

        winner=max(score(3,:));
        for candcount=1:1:3
            if score(3,candcount)==winner
                win3=candcount;
            end
        end

        winCatalogHRNs=[cb;cm;cd];
        winMeasured=[winCatalogHRNs(win1);winCatalogHRNs(win2);winCatalogHRNs(win3)];


       whoiswho(fri,1)=toc;
%%     Check validity of the result

    if (uniquefound)&&(~notfound)
        %one of the detected stars
        samplestar1=[i(lookupt(int16(hrntriplettaaccu(3)))),j(lookupt(int16↙
(hrntriplettaaccu(3)))),k(lookupt(int16(hrntriplettaaccu(3))))];
        samplestar2=[i(lookupt(int16(hrntriplettaaccu(4)))),j(lookupt(int16↙
(hrntriplettaaccu(4)))),k(lookupt(int16(hrntriplettaaccu(4))))];
        samplestar3=[i(lookupt(int16(hrntriplettaaccu(5)))),j(lookupt(int16↙
(hrntriplettaaccu(5)))),k(lookupt(int16(hrntriplettaaccu(5))))];
        samplestaravg=mean([samplestar1;samplestar2;samplestar3],1);

        %if the angle between the star and the simulated reference vector<FOV
        if (acos(dot(xyzlive(fri+firstValidFrame-1,:),samplestaravg))<=(FOV))
            correct(tweakcount)=correct(tweakcount)+1;
            showCorrectFlag=1;
        end
    end




    %% Calculate Attitude Quaternion EXPERIMENTAL

     %the unitary vectors in the AraMiS body frame
     %convert from the left-handed frame to a right-handed, rotated frame.
     %They are unitary vectors, so a simple change of sign and
     %rearranging does the trick
     %y->z
     %z->x
     %-x->y
     wk=[kl(1:3)',-il(1:3)',jl(1:3)']';
```

```
     %the target unitary vectors in the catalogue (primary ref reame = celestial sphere)
     vk=[i(lookupt(winCatalogHRNs)),j(lookupt(winCatalogHRNs)),k(lookupt(winCatalogHRNs))]';

     B=wk*ones(3)*vk';


     S=B+B';
     s=trace(B);
     Z=[B(2,3)-B(3,2),B(3,1)-B(1,3),B(1,2)-B(2,1)]';

     K=[S-s.*eye(3),Z;Z',s];

     aq=s.^2-trace(adj(S));
     bq=s^2+Z'*Z;
     cq=det(S)+Z'*S*Z;
     dq=Z'*S^2*Z;

     feq4=1;
     feq3=0;
     feq2=-(aq+bq);
     feq1=-cq;
     feq0=aq*bq+cq*s-dq;

     rooty=roots([feq4,feq3,feq2,feq1,feq0])

     yopt=inv((max(real(rooty))+s).*eye(3)-S)*Z;

     qopt=(1/sqrt(norm(yopt).^2))*[yopt;1];

     [rotaz,rotay,rotax] = quat2angle(qopt')
     rad2deg([rotaz,rotay,rotax])


     end

    subplot(2,2,1);
%  % Show the brightest identified stars
    recordyn=1;
    %[framezor,last]=showRecordStarframe(mask,mask2,noisysample2show,recordyn,last,images,fri,pos,hrntripletta);
    imshow(images(:,:,fri));
    hold on;
    circle([imsize(2)/2,imsize(1)/2],sensorsz/2,1000,'--');
    hold on;
    for tricount=1:1:count-1
       if tricount==1
           tricolor='r';
       else
           tricolor='b';
       end

       hold on;
```

```
        showTriangle(starXYPositions(:,:,tricount),tricolor);
%         pause(1);
        hold on;
    end
    showTriangle(starXYPositions(:,:,1),'r');

    if (~uniquefound)&&(~notfound)
        text(imsize(1)*(18./20),imsize(1)*(1./20),'Unidentifyied planar triangle.','FontSize',10,'color','r')
        hold on;
    end

    if (uniquefound)&&(~notfound)
        text(imsize(1)*(18/20),imsize(1)*(1./20),'Planar triangle fully identifyied.','FontSize',10,'color','g')
        hold on;
        text(imsize(1)*(999/1000),imsize(1)*(1./20)+20,'HRNs','FontSize',10,'color','g')

        text(imsize(1)*(999/1000),imsize(1)*(1./20)+40,num2str(int16(hrntriplettaaccu(3))),'FontSize',10,'color','g')

        text(imsize(1)*(999/1000),imsize(1)*(1./20)+60,num2str(int16(hrntriplettaaccu(4))),'FontSize',10,'color','g')

        text(imsize(1)*(999/1000),imsize(1)*(1./20)+80,num2str(int16(hrntriplettaaccu(5))),'FontSize',10,'color','g')

        hold on;

        %show them with their locations

        text(pos(1,3),pos(1,2),strcat(' \leftarrow HR ', num2str(winMeasured(1)) ),'FontSize',10,'color','r')
        text(pos(2,3),pos(2,2),strcat(' \leftarrow HR ', num2str(winMeasured(2)) ),'FontSize',10,'color','r')
        text(pos(3,3),pos(3,2),strcat(' \leftarrow HR ', num2str(winMeasured(3)) ),'FontSize',10,'color','r')
    end

    if (notfound)
        text(imsize(1)*(18./20)+30,imsize(1)*(1./20),'Ambiguous planar triangle.','FontSize',10,'color','y')
        hold on;
    end



    %% Show the unitary vectors of reference stars
    if (uniquefound)&&(~notfound)
        subplot(2,2,2);
        vectarrow([0,0,0],[i(lookupt(int16(hrntriplettaaccu(3)))),j(lookupt(int16(hrntriplettaaccu(3)))),k(lookupt(int16(hrntriplettaaccu(3))))]);
        hold on;
        %star2
        vectarrow([0,0,0],[i(lookupt(int16(hrntriplettaaccu(4)))),j(lookupt(int16
```

```
(hrntriplettaaccu(4))),k(lookupt(int16(hrntriplettaaccu(4)))),j(lookupt(int16(hrntriplettaaccu(4))))]);
        hold on;
        %star3
        vectarrow([0,0,0],[i(lookupt(int16(hrntriplettaaccu(5)))),j(lookupt(int16(hrntriplettaaccu(5)))),k(lookupt(int16(hrntriplettaaccu(5))))]);
        hold on;
        vectarrow([0,0,0],[0,0,1]);
        hold on;
        %star2
        vectarrow([0,0,0],[0,1,0]);
        hold on;
        %star3
        vectarrow([0,0,0],[1,0,0]);
        hold on;
        vectarrow([0,0,0],[0,0,-1]);
        hold on;
        %star2
        vectarrow([0,0,0],[0,-1,0]);
        hold on;
        %star3
        vectarrow([0,0,0],[-1,0,0]);
        hold off;
    end

 subplot(2,2,4);
        vectarrow([0,0,0],xyzlive(fri+firstValidFrame-1,:));
        hold on;

        vectarrow([0,0,0],[0,0,1]);
        hold on;
        %star2
        vectarrow([0,0,0],[0,1,0]);
        hold on;
        %star3
        vectarrow([0,0,0],[1,0,0]);
        hold on;
        vectarrow([0,0,0],[0,0,-1]);
        hold on;
        %star2
        vectarrow([0,0,0],[0,-1,0]);
        hold on;
        %star3
        vectarrow([0,0,0],[-1,0,0]);
        hold off;



    if showCorrectFlag
        pause(1);
    end

    fri

%     q1=getframe;
%     qtr1=imresize(q1.cdata, [300 440]);%[241*2 361*2]);
%     framezor(:,:,:,fri)=qtr1;
```

```
    ambiguous(tweakcount)
    nothing(tweakcount)
    unique(tweakcount)
    correct(tweakcount)
    total(tweakcount)


     tweakcount




end
    %figure
    subplot(2,1,2);
    plot(1:1:500,ambiguous,1:1:500,nothing,1:1:500,unique-correct,1:1:500,total,1:1:↙
500,correct)
    legend('ambiguous','no result','wrong result','total frames','correct result')
%     pause(2);

  end %tweakcount
%% Record frames
% mov=immovie(framezor,q1.colormap);
% movie2avi(mov,'results2','fps',1);

plot(1:1:numFrames,pivotingtime,1:1:numFrames,tripletidtime,1:1:numFrames,↙
spikeremovaltime,1:1:numFrames,triangleanalysistime,1:1:numFrames,dimstarremovaltime,1:↙
1:numFrames,whoiswho)
legend↙
('PivotingTime','tripletIdTime','spikeRemovalTime','triangleAnalysisTime','dimStarRemov↙
altime','whichStarIsWhich')

pivotcount=pivotcount(pivotcount~=0);

save('statistics',↙
'pivotingtime','tripletidtime','spikeremovaltime','triangleanalysistime','dimstarremova↙
ltime','pivotingtime','whoiswho','pivotcount')
```

Bibliography

[1] Analog Devices. Ee-258 interfacing micron mtv022 image sensors to blackfin processors engineer-to-engineer note ee-258.

[2] Kosmotras. Results of dnepr lv launch failure investigation.

[3] Kenneth A. LaBel. Single event effect criticality analysis, 15/02/1996 (Retrieved on 15/10/2008). http://radhome.gsfc.nasa.gov/radhome/papers/seecai.htm.

[4] NASA. Single event effects specification (draft), (Retrieved on 15/10/2008). http://radhome.gsfc.nasa.gov/radhome/papers/seespec.htm.

[5] Charles D. Brown. *Elements of Spacecraft Design*. AIAA (American Institute of Aeronautics and Astronautics, 2003.

[6] John Stark et al. *Spacecraft Systems Engineering 3rd Edition*. Wiley, 2003.

[7] James Richard Wertz and Wiley J. Larson. *Space Mission Analysis and Design*. Springer, 1999.

[8] C.C. Liebe. Star trackers for attitude determination. *Aerospace and Electronic Systems Magazine, IEEE*, 10(6):10–16, Jun 1995.

[9] V. L. Pisacane. *Fundamentals of Space Systems*. Oxford University Press, 2005.

[10] Jet Propulsion Laboratory Media Relations. The star, not the instrument was on the blink, 2001.

[11] C.C. Liebe. Star trackers for attitude determination. *Aerospace and Electronic Systems Magazine, IEEE*, 10(6):10–16, Jun 1995.

[12] Wikipedia. Active pixel sensor, 1/04/2008. http://en.wikipedia.org/wiki/Active_pixel_sensor.

[13] Flickr. Search: Night sky pictures, (Retrieved on 15/10/2008). http://flickr.com/search/?q=night+sky+stars&m=text.

[14] University of British Columbia Mathematics department. Field curvature, (Retrieved on 15/10/2008). http://www.math.ubc.ca/ cass/courses/m309-01a/chu/Aberration/field.htm.

[15] Alexander Lee. Carl zeiss lens design, (Retrieved on 15/10/2008). http://www.cartage.org.lb/en/ themes/arts/photography/.

[16] Analog Devices Inc. Adspbf533 data sheet, 2007.

[17] Jet Propulsion Laboratory M. Elghefari. Interoffice memorandum iom 5144-07-017a on ecliptic sel test, 2007.
[18] Micron Technologies. Mt48lc16m16a datasheet, 2007.
[19] Micron. 1/2-inch megapixel cmos digital image sensor mtm001c12stm datasheet, 2006.
[20] Thomas J. McGuire Ray Zenick. Lightweight, low-power coarse star tracker. *17th Annual AIAA/USU Conference on Small Satellites*, 2003.
[21] S. Speretta, L. M. Reyneri, C. Sansoè, M. Tranchero, C. Passerone, and D. Del Corso. Modular architecture for satellites. *58th International Astronautical Congress*, 2007.
[22] Harald Schmidt. Celestia scripting stuff, (Retrieved on 15/10/2008). http://celestia.h-schmidt.net/.
[23] Don Goyette. Don g's celestia scripting resources, 8/9/2004 (Retrieved on 15/10/2008). http://www.donandcarla.com/Celestia/.
[24] Silicon Imaging. Cmos fundamentals.
[25] Kara M. Huffman. Designing star trackers to meet micro-satellite requirements. Master's thesis, Massachussets Institute of Technology, 2006.
[26] University of Maryland. Website, astronomical data center, goddard space flight center, 10/10/2008. http://adc.astro.umd.edu/adc/sciencedata.html.
[27] Craig L. Cole (1) ; John L. Crassidis. Fast star-pattern recognition using planar triangles. *Journal of guidance, control, and dynamics*, pages 64–71, 2006.
[28] Hyunjae Lee, Choong-Suk Oh, and Hyochoong Bang. Modified grid algorithm for star pattern identification by using star trackers. *Recent Advances in Space Technologies, 2003. RAST '03. International Conference on. Proceedings of*, pages 385–391, 20-22 Nov. 2003.
[29] Craig L. Cole (1) ; John L. Crassidis. Fast star pattern recognition using spherical triangles. *AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Providence, RI*, 2004.
[30] QSAR World. Coefficient of variation explanation, 1/04/2008. http://www.qsarworld.com/qsar-statistics-coeff-variance.php.
[31] Percival Goodman. *The Double E.* Anchor Press, 1977.
[32] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. 2006.
[33] Peter Betsch. On the use of euler parameters in multibody dynamics. 2006.
[34] Mathworld. Euler angles, 10/10/2008. http://mathworld.wolfram.com /EulerAngles.html.
[35] John L. Weston Di David H. Titterton. Strapdown inertial navigation technology. 2005 p. 67.
[36] San Jose State University. Quaternions, 3d transformations lecture, 10/10/2008. www.cs.sjsu.edu/ teoh/teaching/cs116a/lectures/lecture07a_quaternions_3dtransform.ppt.

[37] Malcolm D. Shuster. "in quest of better attitudes". *Paper No. AAS-01-250, 11th AAS/AIAA Space Flight Mechanics Meeting, Santa Barbara, California, February 11-14, 2001 ; Advances in the Astronautical Sciences, Vol. 10*, 2001, pp. 2089-2117.

[38] D.N. Nguyen, C.I. Lee, and A.H. Johnston. Total ionizing dose effects on flash memories. *Radiation Effects Data Workshop, 1998. IEEE*, pages 100–103, 24 Jul 1998.

[39] ST Microelectronics. M29w320db datasheet, 2007.

[40] ST Microelectronics. Psd4256g6v datasheet, 2007.

[41] Cambridge Signal Processing. Minotaur bf537 hardware reference, 2007.