SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence Techniques Applied to On-board Space Navigation, Surveillance and Tracking

Physics Department
PhD in Astronomy Astrophysics and Space Science (XXXVI cycle)

Marco Mastrofini
ID number 1486116

Advisor	Co-Advisor
Prof. Fabio Curti	Dr. Andrea D'Ambrosio

Academic Year 2020/2021

The present work has been reviewed by:

- Dr. Moulson Matthew, First Researcher at INFN-LNF, moulson@lnf.infn.it.
- Prof. Furfaro Roberto, Department of Systems & Industrial Engineering, Space Systems Engineering Laboratory (SSEL), The University of Arizona, robertof@arizona.edu.

Artificial Intelligence Techniques Applied to On-board Space Navigation, Surveillance and Tracking
PhD thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: mastrofini.1486116@studenti.uniroma1.it

Alle stelle! Compagne di questo lungo viaggio

Abstract

Space debris constitute a huge threat for the actual and future space traffic. The constant monitoring and Resident Space Objects (RSOs) catalogue maintenance are essential to enhance the "space segment safety". Ground based networks of radar and optical sensors are not enough to face the evolution of such a risky phenomenon. By this, the idea of using already on-board star sensors for a fast, deployable and cost-effective constellation of space sentries. What if all this would be integrated with Artificial Intelligence (AI) techniques? This work presents an AI-based algorithm development for RSOs detection & tracking within the Field Of View (FOV) of electro-optical attitude sensors. This can also be used for navigation functionalities such as High Angular Rate (HAR) determination in a quaternionless situation. The main images processing functions needed for this tasks will be discussed and faced through the AI and coupled with a developed tracking algorithm. Tests, comparison, tasks achievability with real and simulated images will be shown together with the used and trained Machine Learning (ML) models. In the end, foundations developments and keypoints ideas to develop a dual-purpose AI assisted autonomous Star Tracker (ST) for Space Surveillance and Tracking (SST)/ Attitude Navigation (AN) will be highlighted and presented.

Acknowledgments

Questo lavoro rappresenta il mio primo passo verso la Ricerca. E' stato un capitolo importante della mia vita perchè in tre anni succedono tante cose. E' stato un mettersi in gioco perché solo alla fine ti guardi indietro e tiri le somme: sei stato bravo o no? Personalmente credo di si, e posso leggerlo anche negli occhi di chi è entrato nella mia vita in questi anni e di chi già conoscevo. Il percorso mi ha messo alla prova, fatto crescere ed è stato anche pieno di ripensamenti... un po' come la montagna in cui spesso vado e alla quale sono stato meravigliosamente accanto in questi anni.

Un primo ringraziamento lo rivolgo a Melissa. Grazie per l'amore, la pazienza che hai nei miei periodi più tosti e il coraggio che mi dai quando ci lanciamo in mezzo a tante avventure. Grazie in particolar modo per la revisione a puntino che hai fatto di questo lavoro. Ringrazio anche il mio amico Emanuele, punto di riferimento, di sfogo e confronto che sei da molti anni. Grazie anche a te per tutto il sostegno e le soddisfazioni che mi dai. Ringrazio la mia amica Manana, che mi metti sempre il sorriso ogni volta che ci vediamo. Un ringraziamento speciale anche al mio amico Pampa, è stato bello condividere gli anni di Università, dalla triennale fino al dottorato al tuo fianco, sia sui banchi che sul posto di lavoro. Grazie per le belle chiacchierate, spunti e condivisioni sia umane che tecniche (avrò sempre da imparare da te non c'è verso!). Grazie alla mia famiglia che, in mezzo a tutte le difficoltà, rappresenta sempre un punto di riferimento. Grazie Mamma, Papà, Valerio, Damiano ed Edoardo. Grazie anche a tutti gli zii, zie, cugini e cugine. Voglio ringraziare anche tutti gli amici del gruppo di arrampicata e montagna con cui ho condiviso molti km e guadagnato le mie prime vette sia in orizzontale che in verticale, con la roccia o con la neve. La montagna ci aspetta e di sfide ce ne saranno altre! Grazie al mio amico Giuliano, ancora dopo tre anni top-pizzettaro dei Castelli Romani e non solo. Sono felice delle belle chiacchierate che facciamo in pizzeria sull'Astrofisica ed il suo futuro. Voglio ringraziare il Prof Curti, i ragazzi dell' ARCALab e della Scuola di Ingegneria Aerospaziale. Questi quattro anni passati insieme a voi mi hanno regalato tanto e fatto crescere: umanamente, professionalmente e tecnologicamente. Grazie in particolar modo a Ivan, Salim, Gilberto, Gabriele, Francesco, Andrea, Simone, Federica, Daniele, Lucia, Flavia, David, Manuela e Fabrizio. Infine voglio ringraziare anche gli amici del XXXV/XXXVI ciclo con i quali sono cresciuto assieme e confrontato (più da remoto che dal vivo), in particolare ringrazio Piero e Giulia da Padova, Feliciana, Giulia, Alessandro, Umberto, Matteo e Mirko (vi ho promesso la fraschetta e l'avrete!).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Potential collisions between spacecrafts, spacecrafts and space debris are something that can't be neglected anymore. Space debris have been representing a huge concern for every human and robotic mission since the 90's when Space Shuttle program was a reality and the first International Space Station (ISS) module was deployed. Every year the space environment is becoming more and more crowded due to the multitude of deployed spacecrafts both from space agencies/governments and private companies (Figure 1.1). Together with the increasing presence of active platforms in space there is an associated increase of artificial debris. These are largely composed of:

- inactive satellites;
- space objects intentionally released as part of missions;
- rocket parts;
- frozen propellant from propulsion systems;
- fragments from space objects due to collisions and explosions.

As a result, there is a growing risk of collision for operational satellites and other space objects which may either impair the functionality of the active spacecrafts or result in catastrophic failure of the mission or in a catastrophic impact with the Earth.

In 2019, Space Debris Office (SDO) of the European Space Agency (ESA) estimated the number of RSOs larger than 10 cm at 34 000 units, with 128 million objects under 1 cm. United States Space Surveillance Network (US SSN) tracks more than 27,000 pieces of orbital debris[21]. According to it there are approximately 23,000 pieces of debris larger than 10 cm, half a million pieces of debris the size of 1 centimeter or larger and approximately 100 million pieces of debris of 1 millimeter and larger. There is even smaller micrometer-sized debris.

The greatest part of debris is concentrated in Low Earth Orbit (LEO) region (Figure 1.2). This portion of space offers many scientific and business opportunities and nowadays is being interested by mega constellation projects such as SpaceX's Starlink system. Even in the Geostationary Earth Orbit (GEO) there is a non negligible portion of debris. Reducing the risks of collisions is crucial to protect

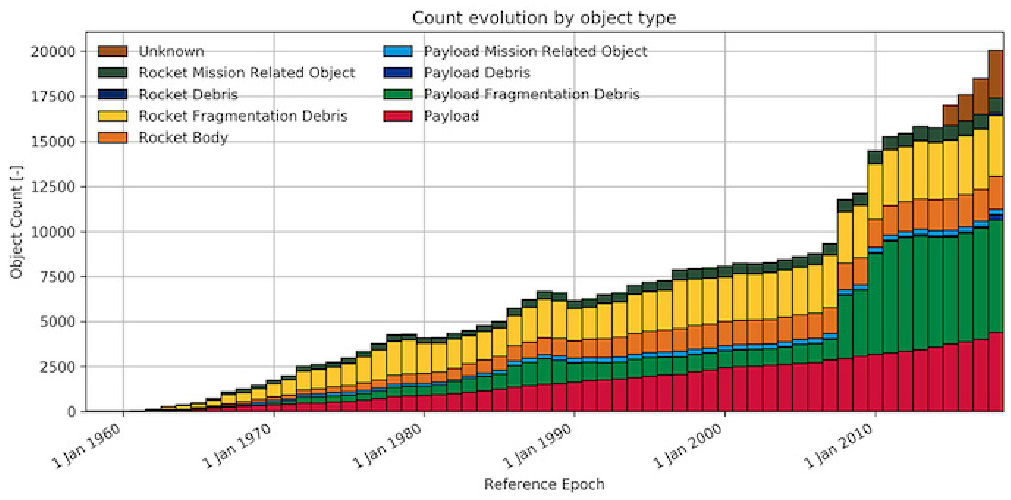


Figure 1.1. Space Debris classes' units time evolution (debris' size $\geq 5\,cm$) [1, 2].

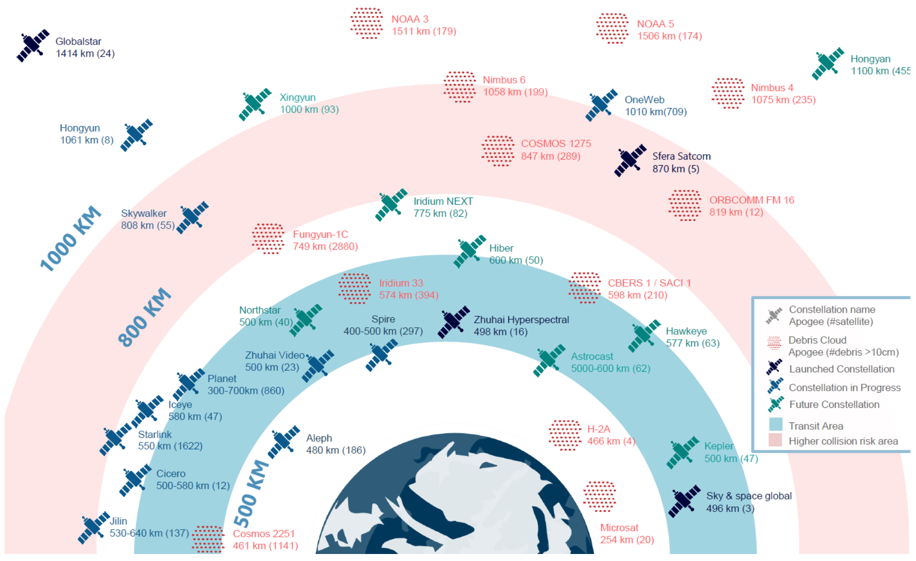


Figure 1.2. LEO main constellations and debris clouds (credits to Euro-consult).

the precious satellites which populate this region (telecommunication, meteorology, military missions).

Nowadays the main techniques to monitor this risk are ground based: Radar and Telescopes. The former can work 24/7 without any weather condition limitation but with a high power consumption (the higher/smaller the debris is and greater will be the power consumption needed for radar detection). The latter have a reduced power consumption (just camera power supply and engines for telescopes maneuvers) but their main limitation is due to the weather conditions, light pollution and night sky operations. Moreover, both these ground based solutions suffer of lack for suitable and easy accessible areas to build such these ground stations. By this, recent ideas of building a constellation of satellites carrying on payloads for debris monitoring are taking place. In particular, the idea of using already existing on-board electro-optical payloads is thrilling because this would reduce the actual cost of such a mission. This pushed the idea of using attitude electro-optical payloads known as "star sensors" or "Star Trackers" for this kind of purpose. These sensors are used to retrieve the satellite orientation w.r.t. a desired star-based reference frame. Anyway, they could constitute a dual-purpose payload if a suitable algorithm for RSOs data extraction from images were developed. This is the rationale behind this work, and it will be implemented through the use of AI. After an introduction to Space Navigation (SN), SST and ML, the main image processing challenges will be discussed and solved in an AI fashion. An image processing based algorithm will be combined with an own made tracking algorithm both for RSOs tracking and HAR determination purposes. The work of a real payload for Space Based Space Surveillance (SBSS) mission based on traditional algorithms will be presented in its advances during these three years. In the end, future plans for the development of an AI-based autonomous ST will be shown, highlighting the key areas and decisions of how developing each modules for RSOs tracking and AN.

This work fits in the space debris field, because it describes the development and design of an AI assisted space surveillance algorithms capable of tracking RSOs. This algorithm aims to be incorporated within existing attitude navigation sensors which are called star sensors. After an introduction about the space surveillance (Chapter 2) panorama and AI techniques (Chapter 3) with particular concern to the space applications, the operative problems are faced and solved. Starting with the Image Processing problem (Chapter 4), the images segmentation (Section 4.1) and consecutive streaks detection and tracking (Section 4.2) have been solved and analyzed with an encouraging algorithm. Then this software has been applied to the HARs estimation problem in Chapter 5, showing suitable performances for the knowledge of the spacecraft spin motion without the use of on board gyroscopes. This dual-purpose algorithm has been put within a preliminary design of an autonomous AI based next generation ST for space surveillance and space navigation purposes (Chapter 6). The last Chapter 7 is devoted to the illustration of my parallel and close related activity as Co-PI of a space based space surveillance mission: Star sensor image Processing for orbiting Objects deTection (SPOT) project. In the end, Conclusions occur to summarize the main findings of this research activity together with the description of possible further investigations.

Chapter 2

Space Navigation & SST

2.1 Space Navigation

Space Navigation refers to the whole bunch of engineering disciplines and activities related to achieving the knowledge of the "state of a space system". In particular for spacecrafts it can be differentiated in two parts:

- orbital navigation;
- AN.

The former aims to solve the problem of where the spacecraft is. By the knowledge of the spacecraft center of mass position is possible to predict its future position thanks to dynamics models. The need of spacecraft orbital navigation comes from the need of planning future maneuvers, pointing at desired directions to chase a target or to provide a service, avoiding collisions with other spacecrafts (both in a Formation Flying (FF) mission or in a LEO context) etc. The latter concerns the relative orientation between two reference frames: a desired reference frame and the spacecraft reference frame. The need of a precise AN system comes from the need of thruster orientation prior a spacecraft maneuver, the need of precisely pointing of an instrument towards a target (antenna, Light Detection and Ranging o Laser Imaging Detection and Ranging (LIDAR), optical payloads), the need of maximize the on-board generated power by pointing the solar panels towards the Sun etc.

Within this work we will focus more attention on some subjects of AN rather than orbital or trajectory navigation even if the use of images from on-board optical payloads could be used beyond the attitude knowledge purposes.

2.1.1 Space Attitude Navigation

The state of the system in this engineering science is represented by a collection of parameters which can let us retrieve the orientation of a body (spacecraft) with respect to a desired reference frame. Generally for description and visualization purposes a set of Spacecraft's "Euler Angles" (ex. Roll, Pitch ,Yaw presented in Figure 2.1) is used while in an operative scenario the set of four parameters named "quaternions" is more used (Figure 2.2). The aim of space AN is to retrieve this information by measuring some quantities and developing methods to achieve the attitude knowledge. In general a space AN system is made of

- sensors;
- processing units;
- algorithms.

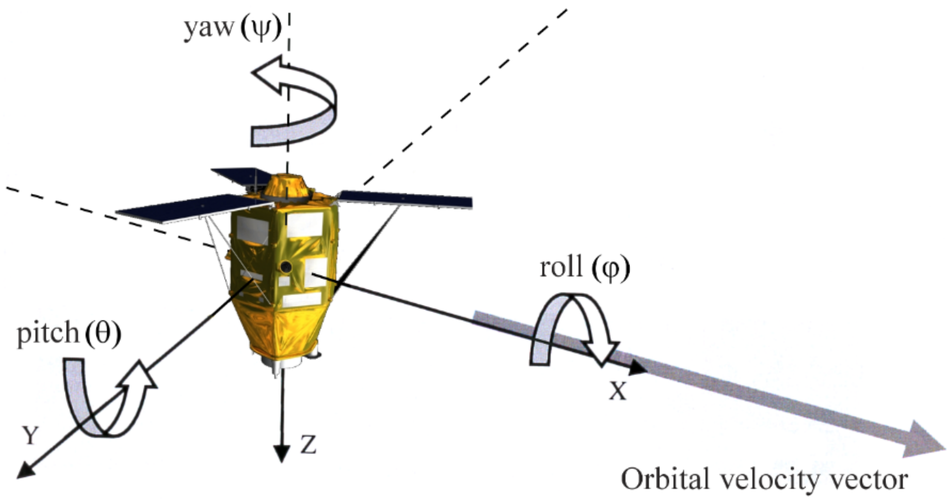


Figure 2.1. Spacecraft's Euler angles Yaw, Pitch and Roll definition [3].

Sensors are devices which can measure quantities after a proper calibration procedure. The nature of the measured quantities varies according to the nature and type of the sensor:

- Stars position in the FOV for ST;
- Angular velocities for gyroscopes;
- Angle above the horizon for Earth Sensors (ES);
- Sun's direction w.r.t. image plane for Sun Sensors (SS).

In every sensor, then, there is a suitable algorithm which can transform the collection of measured quantities in other more useful data to retrieve part or the total attitude knowledge thanks to algorithms which implement noise signal filtering and theoretical method for the measured quantities manipulation. All of these algorithms need to be implemented on the embedded system, the so called "Processing Units" (PUs). These last are often made of a micro-controller for low power operations or by coupling a micro-controller and Field Programmable Gate Arrays (FPGAs) in more powerful applications which need real time or quasi-real time operations (example the image processing step in star sensors). During this work the space AN will be partially faced, in particular the use of ST will be considered and the following section will deeply describe the working of these electro-optical sensors.

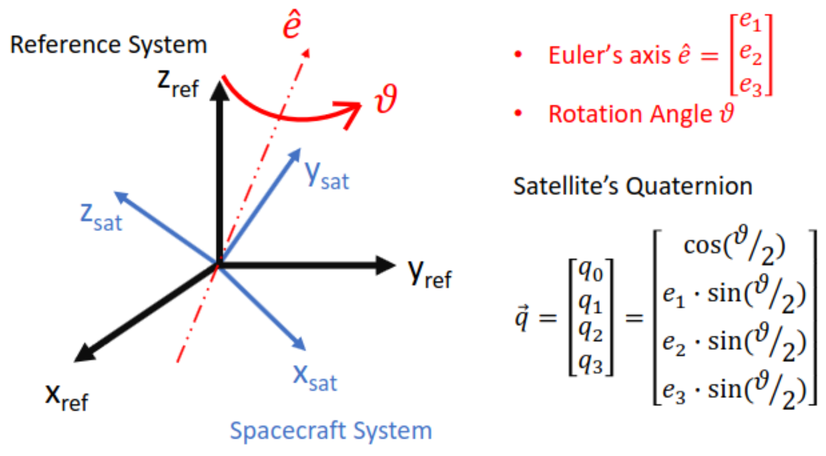

Reference System
$\hat{e}$
Z_{ref}
ϑ
Y_{sat}
Z_{sat}
Y_{ref}
X_{ref}
X_{sat}
Spacecraft System
Euler's axis $\hat{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}$
Rotation Angle ϑ
Satellite's Quaternion
$\vec{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\vartheta/2) \\ e_1 \cdot \sin(\vartheta/2) \\ e_2 \cdot \sin(\vartheta/2) \\ e_3 \cdot \sin(\vartheta/2) \end{bmatrix}$

Figure 2.2. Spacecraft's system, Euler rotation elements and quaternion definition.

2.1.2 Star Sensors

Star sensors or STs are electro-optical devices capable to retrieve the platform's quaternion through photos of the night sky (Figure 2.3). They are composed of:

- camera: sensor and Optical Head (OH);
- PU.

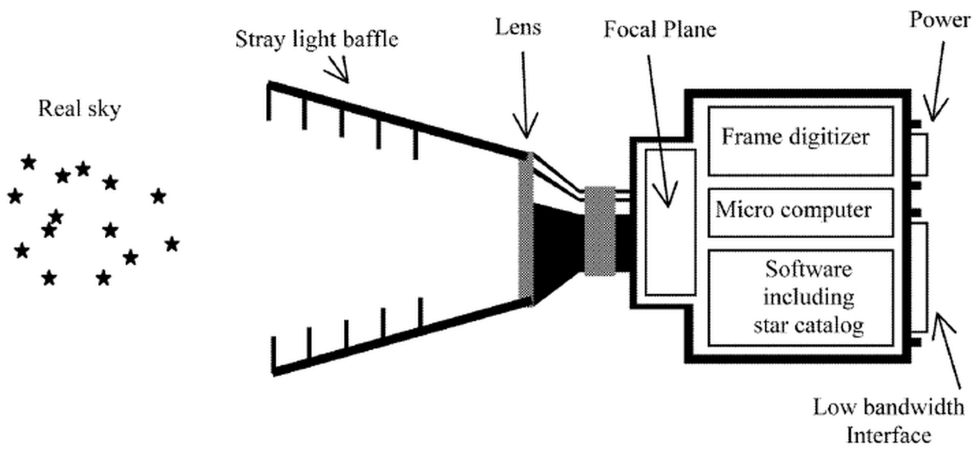


Figure 2.3. Star sensor's main parts [4].

The sensor is hosted and protected within a structure which constitutes the camera body. While the photo is being acquired, sensor collects photons through a system of lenses. These are contained within an OH which is protected by a conical structure named as "baffle". The baffle is used to reduce the straylight condition to which the camera is subjected when pointing at the sky while having a strong lightsource in proximity.

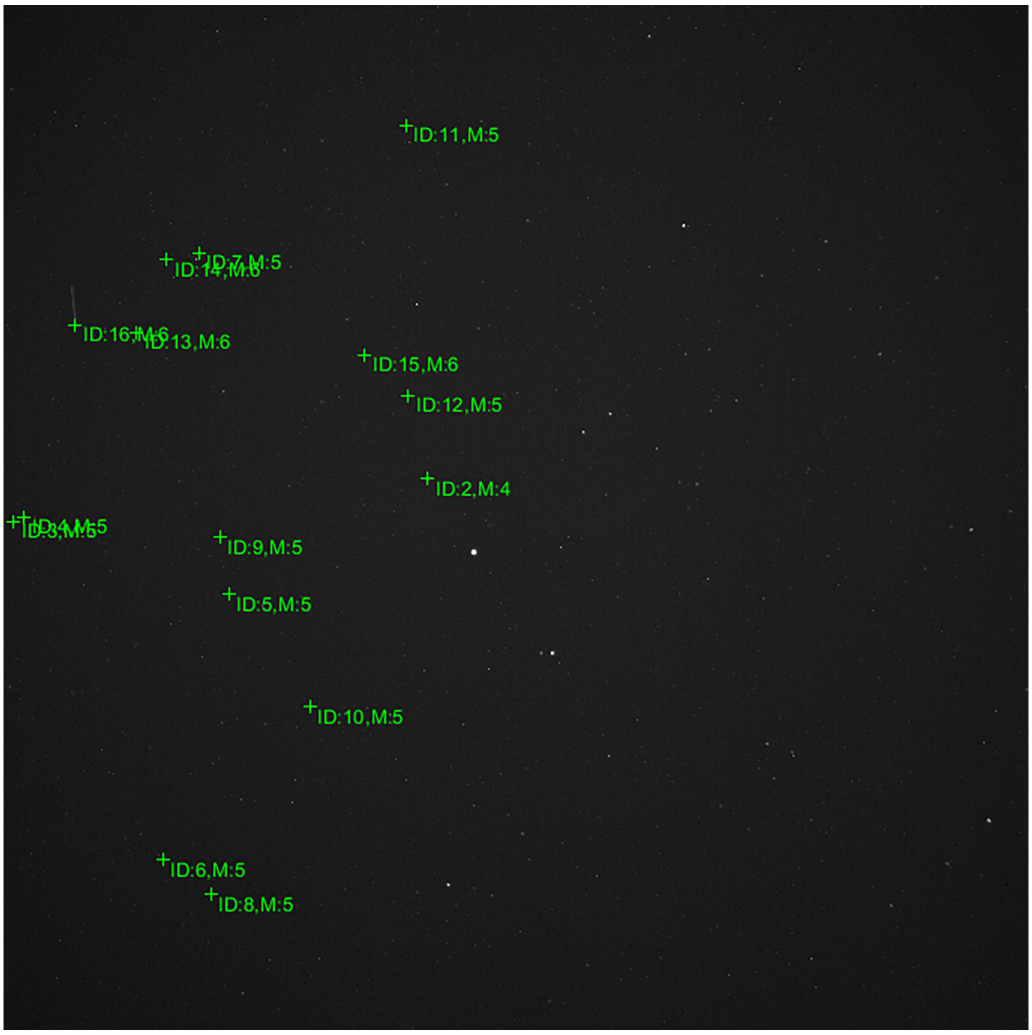


Figure 2.4. ARCSEC SAGITTA Star sensor image with brightest objects' centroids (green spots). Each centroid has its identifier (ID) and associated magnitude value (M).

Star sensor is used to measure the stars positions within the camera reference frame (Figure 2.4).

For each group of pixel associated to the same star the magnitude of the star and the center of light distribution (centroid) is computed.

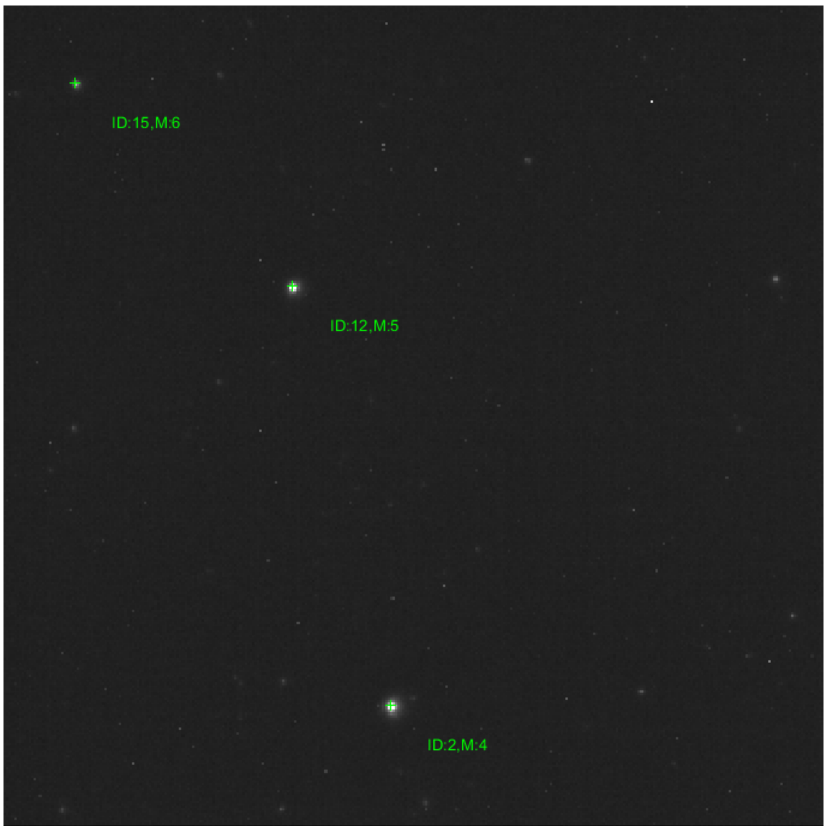


Figure 2.5. Stars'centroids (detail of 2.4).

Through the "pin-hole" model the centroids coordinates in the image plane reference frame are then converted into the Camera Reference Frame (CRF) star unit vector 2.6. Actually, the pin-hole model is a set of mathematical relations which takes into account the camera's parameters (sensor size, pixel size, focal length) and relates any point position on the image plane with its associated unit vector in the CRF.

The stars' unit vectors information is then used to retrieve the attitude of the sensors. By the knowledge of how the sensor is mounted onto the platform is then possible to know the platform quaternion (and thus the spacecraft orientation with respect to a specific reference frame).

Quaternion Computation

Stars unit vectors information is the output of the star sensors after the image processing module. This output is then given to the attitude processing chain .

This processing chain differs if an a priori quaternion estimate is available or not.

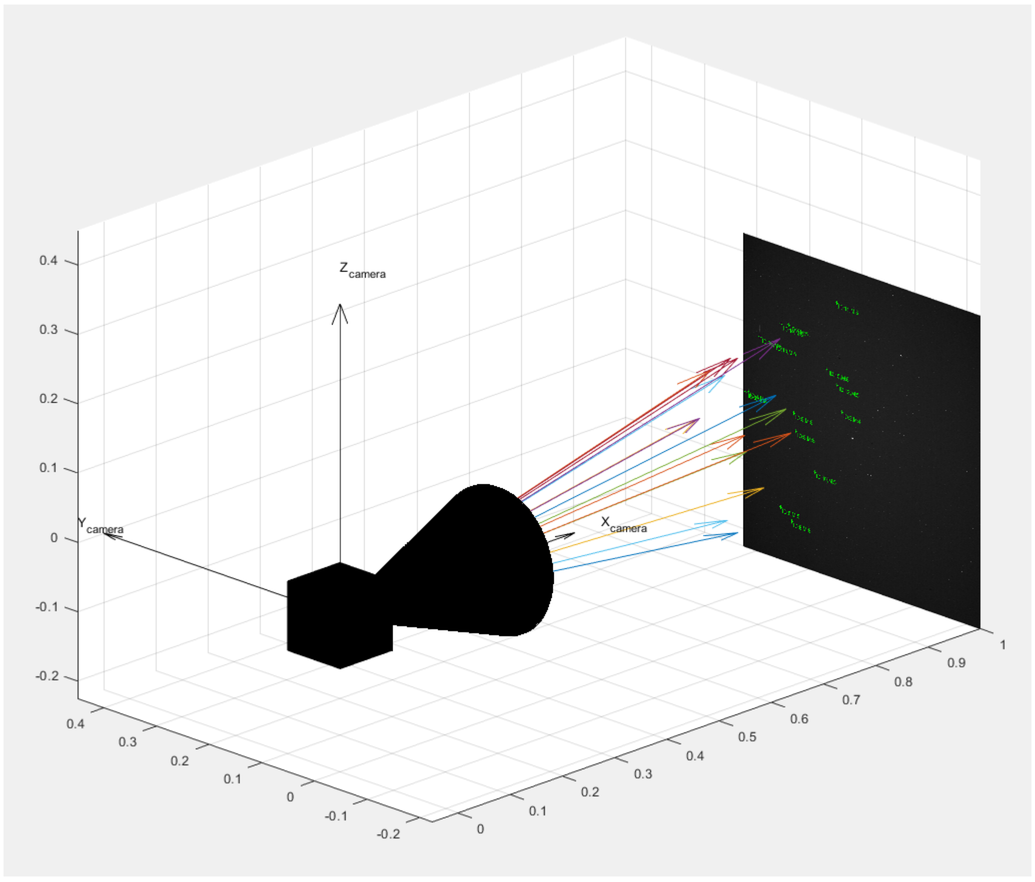


Figure 2.6. CRF and centroids associated unit vectors.

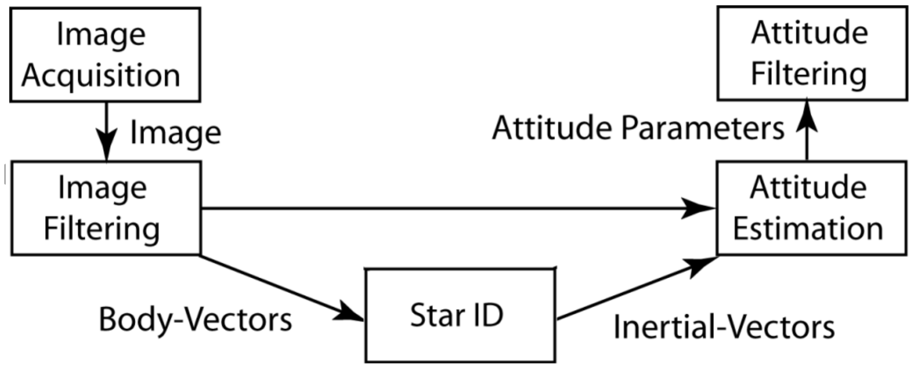


Figure 2.7. Star sensors' algorithm chain scheme [5].

When the quaternion info is not available and the stars unit vectors in the CRF are expressed, the Lost In Space Algorithm (LISA) is performed. By the unit vectors distribution is indeed possible to perform a research in an on-board star catalogue in the star sensors' memory. Then a star's features or patterns based research is started and the process repeats until most of the detected stars in the FOV are identified ("Star ID" module in 2.7). Several traditional algorithms have been developed to solve this problem like TRIAD [22], Planar Triangle algorithm [23], Multi-Poles algorithm [24] etc... AI solutions have been developed and proposed for solving this task [16] but some time is still needed to see these algorithms on a real space mission due to a long validation process. After identification, a first solution for the quaternion is found through the Attitude Determination (AD) routine. In case of an a priori quaternion availability, the process of identifying visible stars is not needed because with the a priori quaternion info they are known and tracked by the tracking algorithm. In the ST "Tracking mode" the attitude is updated on varying of the star field and the "Star ID" is bypassed. Moreover, under this tracking mode it is even possible to retrieve an angular velocity estimate based on the finite differences of the stars position in the images sequence.

2.2 Space Surveillance and Tracking

To monitor space debris phenomena, Near Earth Objects (NEOs) and prevent potential collisions, space agencies/governments and privates from all over the world have been increasing funds for the development of SST systems in the framework of their Space Situational Awareness (SSA) programmes.

SST system is capable of tracking and performing orbit determination to catalogue both RSOs and NEOs. It generates data that can be used to predict hazards to operational spacecraft or to infrastructure on ground. Such as for example a potential satellites-debris collision or a potential collision with a reentering object (inactive spacecraft, rocket body or NEO).

ESA states [25] that:

"SST system can be considered as a 'processing pipeline' to process observation data acquired by sensors – the telescopes, radars or laser-ranging stations – and provide derived applications and services, typically comprising collision warnings."

The main core of a SST system is the object catalogue, which contains up-to-date orbit information for all objects over a certain size threshold and info concerning their physical properties. To reduce at minimum the size of this catalogue, a correlation check is performed. Here the orbit of the detected object is determined and then it is correlated with the orbits of the catalogued objects. In this way it is possible to verify if a new object is detected or to update a catalogued object's orbit with new sensor's data.

From the catalogue info, several services can be provided by the SST:

- Conjunction predictions;
- Fragmentation detection;
- Reentry prediction & Impact analysis.

Conjunction predictions are related to conjunction events (close approaches between two objects) that allow to issue warnings for potential collisions. The second service is related to the fragment analysis to determine when, where and who did collide or explode. The third and last service computes orbital lifetimes of reentering objects and predicts the reentry trajectory to assess potential on ground collisions with sensible targets.

2.2.1 SST Panorama in USA and Europe

Several SST systems have been built or are being built all around the world to track space debris and NEOs. They create, maintain their own RSOs catalogues and can cooperate, share data upon international agreements. Here, an overview of the world SST panorama is provided by showing the USA and European SST systems organizations, both for space debris and NEOs monitoring. In the USA, space debris surveillance is performed through the US SSN which is operated by the United States Space Force (USSF) under the responsibility of the United States Space Command (US SPACECOM). The conjunction, fragmentation, objects reentry analyses and predictions are performed by the 18^{th} Space Control Squadron located at the Vandenberg Space Force Base, California [21]. The sensors network is constituted by more than 30 ground-based and worldwide distributed radars and telescopes. Moreover, six dedicated satellites are part of this network too.

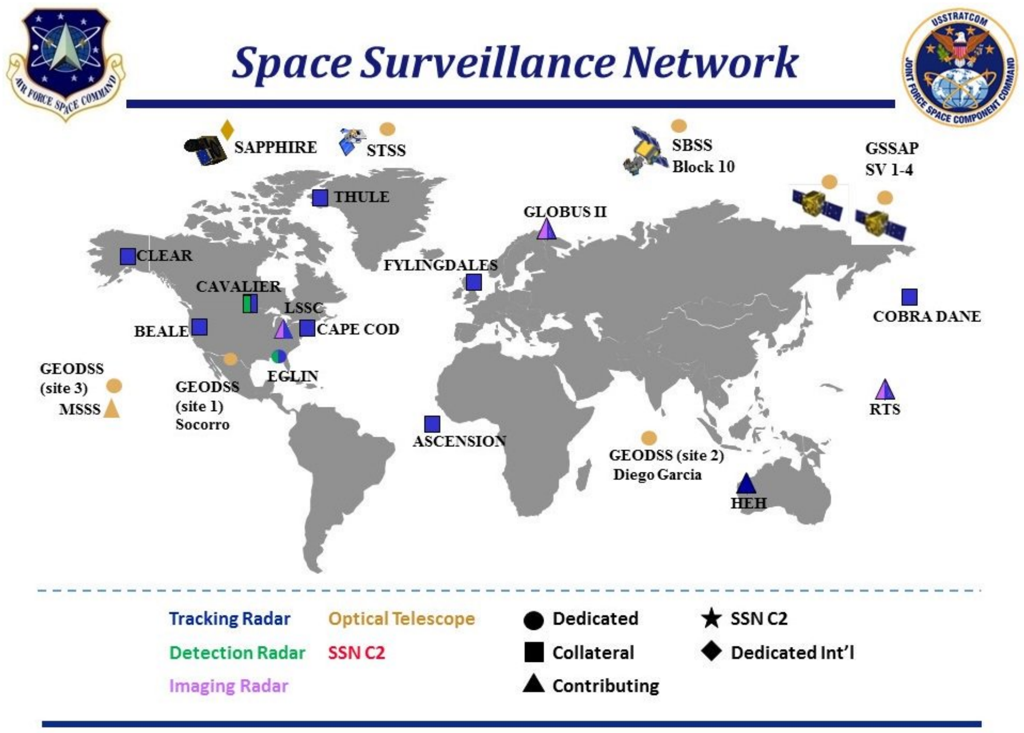


Figure 2.8. US SSN worldwide distribution (credits to USSF).

US SSN's sensors are separated in three categories:

- Dedicated sensors;

- Collateral sensors;
- Auxiliary sensors.

Both the dedicated and collateral sensors are operated by the US SPACECOM, but while the former have a primary objective to acquire SSN data, the latter obtain these data as a secondary objective. The auxiliary sensors are not operated by the US SPACECOM.

NEOs are tracked by telescopes and radar systems worldwide. Each of these ground centers submits observations to the Minor Planet Center (MPC) and risk of impact analyses are performed with these data by the center for NEO studies of Jet Propulsion Laboratories (JPL) and by ESA's NEO Coordination Center (NEO CC).

In Europe, ESA is focusing research and developing technologies for SST systems. In the 2013-16 period their ground prototypes were developed and deployed. Since 2017, ESA is continuing to develop SST systems, conducting additional qualifications of national assets (radars, optical telescopes and laser-ranging systems, SST data processing and application, SSA-NEO infrastructures [25]).

Another recent born European reality is the European Union SST. It was established in 2014 by the European Parliament. France, Germany, Italy, Poland, Portugal, Romania and Spain are the European Union (EU) state members of this SST Consortium. Since 2016 the EU SST has been cooperating with the EU Satellite Centre (SatCen) to develop a European SST capability.

The SST services range from in orbit fragmentation characterization, conjunction analysis and re-entry prediction and related risk assessment[26]. Research & Development programs are being fulfilled to improve SST capabilities. The areas concerned by these programs are: sensors, processing and service function activities. Their main researched subareas are:

- Development of ground telescopes, radars, lasers to catalogue objects in all the earth orbital regions and fuse their measurements;
- AI techniques application for improving detection and characterisation of RSOs;
- Radio Frequency (RF) sensors to better characterize active objects' maneuvers;
- Development of space-based surveillance systems to complement those on the ground;
- Improvement of autonomy in sensor tasking;
- Improvement of autonomy for satellites to anticipate and act against a collision risk;
- Development active debris removal techniques and in-orbit proximity operation algorithms.

2.2.2 Space Based SST Systems

Besides these SSA challenges, a recent one is related to the development of space based SST systems. In particular with reference to [27], it can be seen that in Period 3 (2017-2020) of the ESA's SSA programme, the development of space based sensors appears as one of the main research areas. This topic is one of the most recent challenges in the SSA field and initil studies from ESA-Airbus have provided a concept. This concept uses a platform in a sun-synchronous dusk/dawn LEO orbit to scan the entire GEO daily. This will be achieved with an on-board telescope with an aperture of about 20 cm that is sufficient for SST tasks. Airbus has been in charge of the feasibility study, design and development of a SBSS mission [6]. In Figure 2.9 the analysed orbit can be seen in the bottom figures. It is the red one shown in two orthogonal projections. This sun-synchronous dusk/dawn orbit lets the observing satellite scan the whole GEO belt (green circle/line) periodically with a low FOV and with a suitable low phase angle (achieved through an anti-sun pointing strategy). The scanning strategy of the GEO belt is shown in the top part of the figure where several fences appear. This step-and-stare strategy is achieved by pointing the telescope through the platform attitude control actions.

The same SBSS demonstrator can contribute to improve the statistical knowledge on the sub-catalogue small-size (down to 1 mm) space debris population in LEO too. Having a look at Figure 2.10 it is possible to see the sensor pointing at a known LEO object within a tracking region. In this way the capability of the sensor can be assessed via the tracking of known objects by collecting measurements.

This SBSS design is the first step for the ESA's *Optical In-Situ Monitoring* project which aims to design and integrate a breadboard of a space-based space debris camera and to develop and test its end-to-end processing chain. Other existing or ended SBSS missions are the Canadian NEOS Sat and Sapphire ones. The former is for NEOs space based surveillance while the latter is purely oriented to space debris and satellites detection. Both of them use a suitable telescope to achieve this scope. Other two missions were performed by US: the Mid-course Space eXperiment (MSX), equipped with the Space Based Visible (SBV) sensor and the US SSN SBSS mission.

A detailed study for the on-board detection of SOs is performed by D. Spiller et al [7]. Here the idea is to take advantage of a star sensors equipped platform to detect debris and spacecraft. The proposal of a suitable software architecture to achieve this task and send to ground useful data for orbit determination purposes is provided. Through the development of a High Fidelity ST Simulator (HFSTS), the algorithms could be tested to analyse RSOs' detection limits in terms of max magnitude, relative distance and dimensions simulating a common ST.

Considering Figure 2.11, through the HFSTS it is possible to generate images similar to ST. Real noise sources, rolling shutter effect and stars Point Spread Functions (PSF) are simulated together with objects streaks and magnitude according to the considered phase angle and object's properties and relative distance. The images are used as input for the processing chain:

- Pre-processing;
- Clustering;

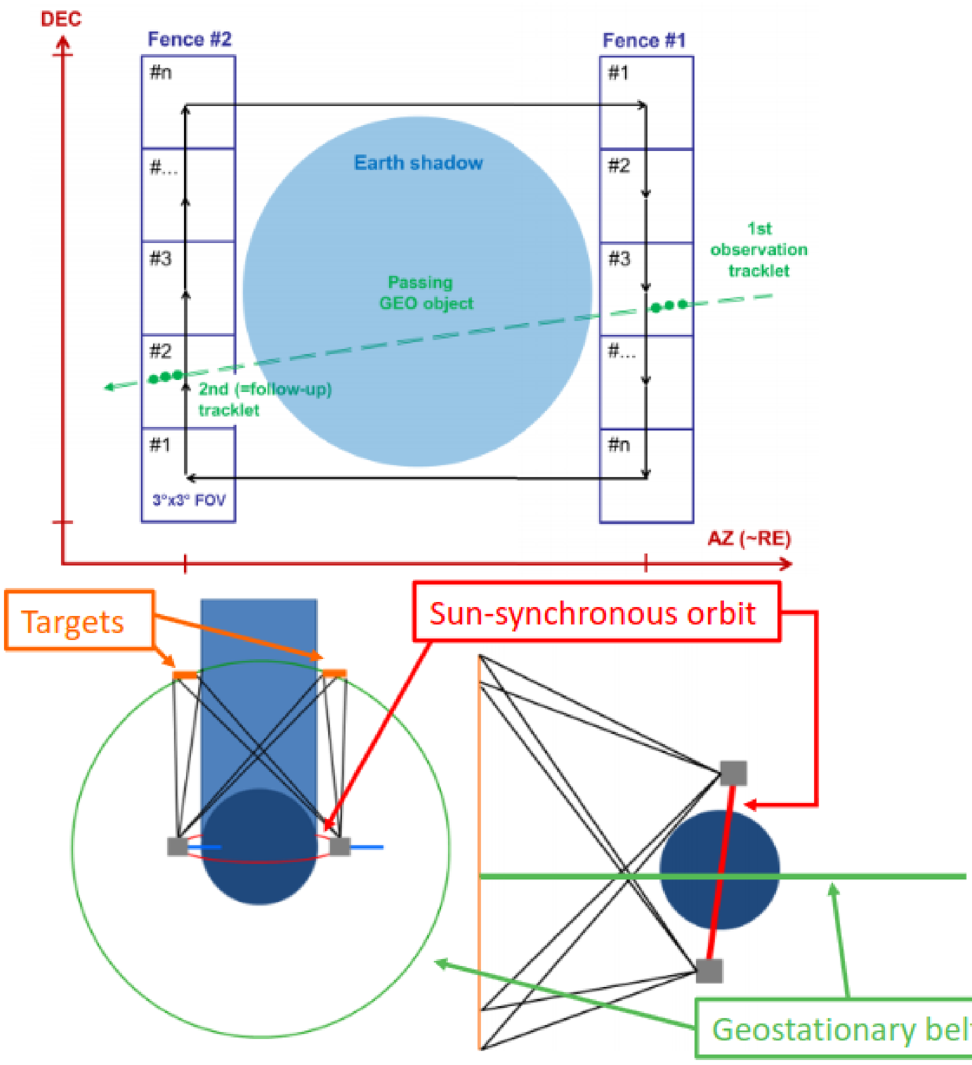


Figure 2.9. GEO observation strategy [6], the observing platform is on the sun-synchronous orbit.

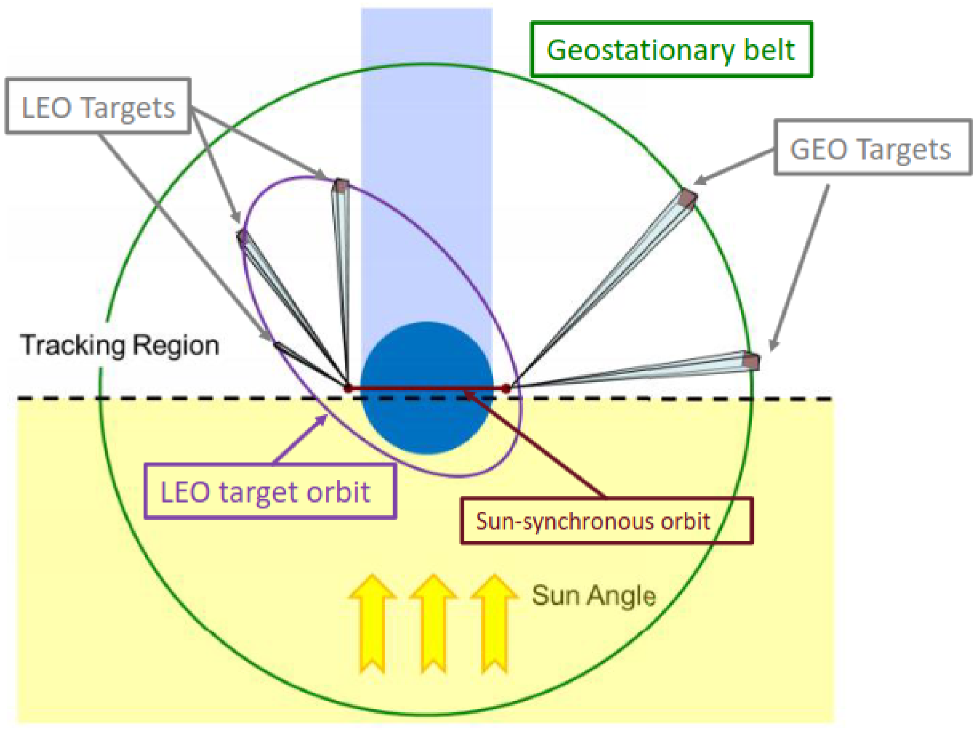


Figure 2.10. LEO observation strategy [6], the observing platform is on the sun-synchronous orbit.

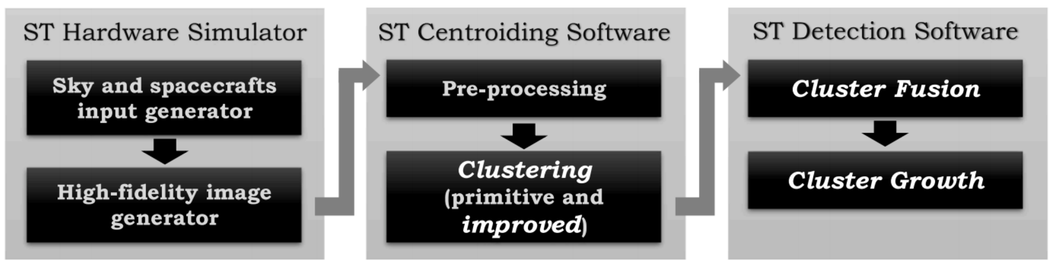


Figure 2.11. SPOT high level architecture [7].

- Cluster Fusion;
- Cluster Growth.

First module purpose is the image segmentation, identifying the over threshold pixels and under threshold ones. Its output is the segmented image where every pixel carries with it the associated energy value. Then the Clustering algorithm groups the over threshold pixels through two steps: "Primitive Clustering" and "Improved Clustering". This is achieved via several hard coded filters and in the end the centroids, energies and dimensions of all the clusters are computed. "Cluster Fusion" is needed to merge clusters related to a couple of consecutive images. It performs cluster association just at an image couple level. The cluster growth instead is capable of associating clusters to their respective objects in a continuous flow of images from the objects appearance inside the FOV untill their disappearance.

This work is not just focused on the objects info extraction from images. It aims also at carrying out a feasibility analysis in terms of objects detection capability is performed. A simulation campaign has been performed: for every combination of objects' magnitude, relative distance and observation direction fifty scenarios were analyzed. For each of them, different initial conditions were used for initializing ten fictitious objects inside the FOV. In this way it was possible to evaluate several performance indices among which the *Percentage of Tracked Objects* is shown in Figure 2.12.

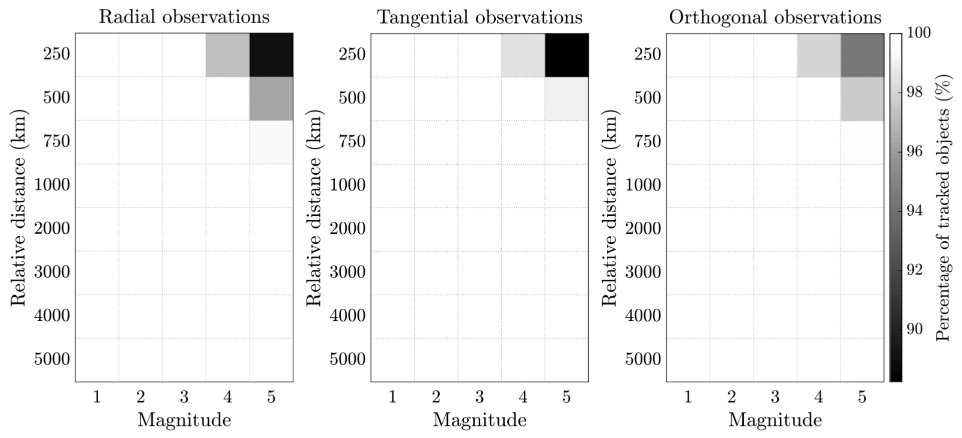


Figure 2.12. SPOT feasibility campaign results [7].

This index is defined as the number of tracked objects over the total initialized one in a single test case. Considered magnitudes reach the maximum limit of 5 that is a representative value for the common available ST. Relative distances reach the maximum limit of 5000 km because only objects greater than 10 m can be tracked at greater distances (we remember that the target is detecting smaller space debris). Results show that all the objects are almost tracked ranging from 250 km to 5000 km for all the magnitude intervals. This simulation campaign was conducted not considering all the possible delays and processing times for the image processing and other algorithms and the results are related to the maximum performance indices values.

This work is part of the development process of the SPOT project. This project is funded by Italian Space Agency and developed by School of Aerospace Engineering, Sapienza University of Rome. The SPOT mission, which will perform its In Orbit Validation (IOV) mission within 2024, is different from other ones that have been cited previously because it aims to use STs and not just telescopes. This can increase the availability of suitable platforms to integrate this technology and could lead to a cost effective LEO, MEO (Mean Earth Orbit) and GEO constellations for SST purposes.

As stated previously the on-board segment aims to RSOs info extractions. This project is being developed together with the design of a ground module which is devoted to a first orbit determination step and the creation of a orbit catalogue. Then this info can be used for conjunctions predictions and orbit to orbit correlations.

2.3 On-Board Electro-Optical Sensors

Electro-optical sensors are the main actors of this research together with the AI. This section provides an overview on the practical use of electro-optical sensors. Here the main features of these sensors will be shown and described.

For *camera* it is meant an electro-optical device made of:

- A *camera body*: it contains the full electronics: sensor, power supply system, cooling system and processing units. It is sometimes covered by electronic interfaces for connectivity, parameters controls and a structure to protect what is inside it. The sensor is generally of two types: Charged Coupled Device CCD or Complementary Metal-Oxide Semiconductor (CMOS).

- An *Optical Head*: it is the equipment containing the lenses system and all the mechanical systems concerning lenses controlled motion and focus adjustment. It generally has a cylindrical shape and can be fixed (for fixed focal length systems) or with variable longitudinal length (zoom).

Camera Body

It is mainly devoted to the sensor protection: from air dusts, water, shocks. The sensor is a rectangular grid of basic electronic units (CCD or CMOS). It has a Width (W) and Height (H) and a size which is given by multiplying W with H. W and H can be expressed in pixels or millimeters. The conversion factor is given by the physical length of every single pixel which is called pixel size (1-22 μm of typical range.). Another feature concerning the sensor array is the form factor: a ratio which quantifies the exact proportions between H ad W.

The camera used in my objects detection works has the following values:

- W = 960 pixels

- H = 640 pixels

- pixel size = 24.06 μm

- Image size =$960 \times 640 = 0.6144$ Mpixel

- Form factor = 3:2

From the previous features it can be retrieved that the physical sensor dimensions are 23.1 mm large and 15.4 mm of height with a diagonal of 27.8 mm.

Optical Head (OH)

This device contains the whole optical system. OH features together with camera body ones are responsible for the global camera features. Moreover, by changing the OH while mantaining the same camera body, different global features can lead to a different use of the same sensor (let's think about a Reflex camera with a wide view or zoom OH). On the external side of OHs, parameters control interfaces can be found such as: focus, system aperture, focal length etc. OH main features are:

- focal length (f_0): it is the focal length of the lens. It is measured in mm and it can be fixed or variable (zoom);
- Aperture diameter (d): it is the diameter of the OH aperture;
- f number: ratio of focal length with the aperture diameter $f_{number} = \frac{f_0}{d}$.

Camera

Camera body and OH coupling is the *camera.* It has performances which are function of camera body ad OH features. The most important camera performance is the *FOV*. *FOV (θ_{FOV})* is the angular portion of scene which is visible through the camera. For circular sensors it is a unique value while for rectangular sensor both Vertical (VFOV) and horizontal (HFOV) can be defined. When the shape is rectangular and just a FOV value is provided, the diagonal FOV is given.

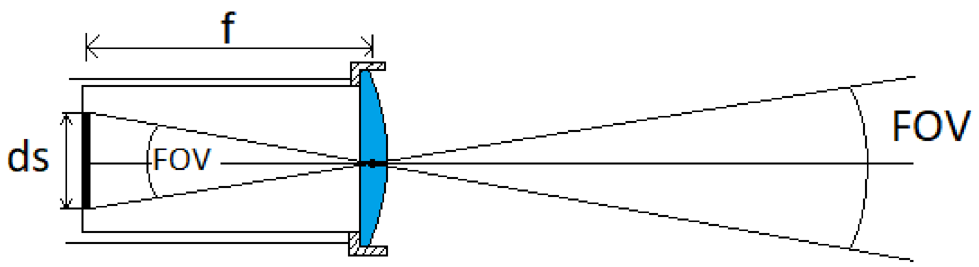


Figure 2.13. FOV computation scheme.

By Figure 2.13 it is possible to retrieve a trigonometric formula which links FOV, f_0 and sensor dimension (D).

$$\theta_{FOV} = 2 \arctan \left(\frac{D}{2\, f_0} \right) \tag{2.1}$$

As a consequence of 2.1:

- If f_0 increases, the FOV decreases. As a matter of fact, with a zoom objective it can be noted the FOV restriction as the focal length is being increased. This is the reason why zooms have a variable and low FOV if compared to wide angle OHs;
- Considering the same OH, the use of a larger sensor makes the FOV increase.

Another important concept to measure the performance is the Istantaneous FOV (IFOV). It is a crucial when the aim of an on-board camera is Earth Observation (EO) or attidude determination.

The camera used for objects detection purposes along this paper uses an OH with variable focal length. Its features are:

- f_0 range: 18-105 mm;
- θ_{FOV} range: 77.3° - 15.6°;
- d, variable aperture: f/3.5-22.0.

Chapter 3

AI: How It Has Changed the Space and How It Is Involved in This Research

In this chapter a brief history and introduction of the AI techniques with particular regard to Deep Learning (DL) is faced. Then, an overview of the DL impact in the space field will be covered. In the end, a description section about the machine learning models applied in this work follows.

3.1 A Short Introduction of AI Techniques & Deep Learning

We are experiencing a flourishing period for the AI. This is proved by the large number of publications and applications of this technology. Nothing so surprising if we think that since the 50's and 60's a not negligible effort was made in the field in terms of study and research. In particular the DL field of AI started in 50's but before being extensively applied many years had to be waited due both to the lack of computational/hardware resources and a lack of huge training datasets for the models. A first boost to DL was given in the 80's when the rediscovery of the "Back-propagation" algorithm for the training the modern Neural Networks (NNs) was made. This brought, in 1989, to the first application for handwritten digits classification performed by *LeNet*. This model was developed by Bell Labs and largerly used by the United States Postal Service (USPS) [28]. Then NNs world had to wait again for at least 20 years when the first DL competition started. As a matter of fact, with the ImageNet challenge the rise and popularity of Convolutional Neural Networks (CNNs) took place and by the 2015 the problem of multiclassification on images was considered solved with an achieved accuracy of 94.5 % [29]. The 2010-2020 decade saw the rise of the CNNs which rapidly invaded every branch of the scientific community where images were used, processed and information extracted via classical algorithms. The CNNs have become so efficient, flexible and important for computer vision applications that nowadays it is impossible to exclude a CNNs approach for problems having images as input. This last decade was the most suitable for this DL revolution because of the huge amount of materials for dataset creation, the hardware and software advances. Actually the development of fast, burly parallel chips called Graphical Processing Units (GPUs) occured with the videogames industry growth. Here a more powerful hardware was needed to handle high rendering scenarios. In particular, the development of powerful NVIDIA GPUs together with the Compute Unified Device Architecture (CUDA) [30] software made this hardware the most suitable for boosting the CNNs training and inference, achieving the near real time or sometimes, real time capability.

3.1.1 AI & Deep Learning

AI is the scientific field where artificial intelligence agents interact with the surrounding environment, performing their tasks and learning from outside as human beings do normally.

In particular, related to the *learning process*, the AI sub-field of ML focuses on the possibility of making an artificial machine to learn tasks. Actually, from an algorithm point of view, classical approach for solving problems is the implementation of a program which receives input data and contains all the instructions to provide desired outputs. This is not true anymore with ML algorithms where they receive input and output data and learn the specific set of rules which link these two sets. Learning process is achieved via training of the machine with a suitable couple of input and output: the dataset. The more representative of the task the dataset is, the more the machine will learn it well. Three sub-fields of ML can be mentioned:

- Supervised Learning;
- Unsupervised Learning;
- Reinforcement Learning.

The first sub-field is related to making a machine learn through the provision of both input and related outputs untill the machine achieves sufficient accuracy to perform its task. The second one is related to achieve learning of a task by a machine via the only provision of input data. This is an interesting sub-field of ML because it finds several applications as the case of clustering algorithms. In the last case an intelligent agent, the machine, interacts with the surrounding environment and at each step of the process a reward function is given to make it learn a specific policy/task.

In the end, DL comes: It is ML with the use of NNs which are architectures of interconnected layers which are composed of neurons. These are the basic units of NN models and perform linear combination of their input parameters. By this operation each layer performs operations and provides in output a representation of its input parameters. A stack of interconnected layers performs several representations untill arriving to NN's output. *Deep* in DL stands for this idea of successive levels of representation. The applications of NNs are a lot nowadays as well as the variety of DL models. They are used for various tasks such as image processing, classification and detection of objects, clustering algorithms, text and speech recognition, regressions etc.

3.1.2 Convolutional Neural Networks

CNNs are a particular class of NNs which are heavily applied in computer visions problems like image recognition and image processing. Their name is derived by the convolution operation which is actuated by one or more layers inside the network itself. In a rough approximation, they are stacks of convolution and maxpooling layers. They are capable of learning high and low-level features of an image regardless of their position and orientation. A great advantage of a CNN is that the weights to be learned are shared between the layers, minimizing the needed memory storage (interesting fact for on-board applications).

The convolutional layer works in the following way: a convolution operation is performed by sliding a small window (typically 3×3 pixels) over height and width of the image and over its color channels (Figure 3.2). The window creates a patch feature map, which is multiplied with a learned weights matrix called convolution kernel. This operation could cause image size reduction, which is handled by applying padding, that is to say adding border pixels in order to make the output size the same as the input (Figure 3.3). The neurons contained in the network layers are activated through an activation function: ReLU (Rectified Linear Unit), softmax and sigmoid are typically used. These activation functions introduce non linearities in the NNs and make the models easy to handle non linear problems. ReLU provides an output value equal to the input value if the input is positive, otherwise ReLU provides 0 as output (Figure 3.1).

$$y = f_{ReLU}(x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$$

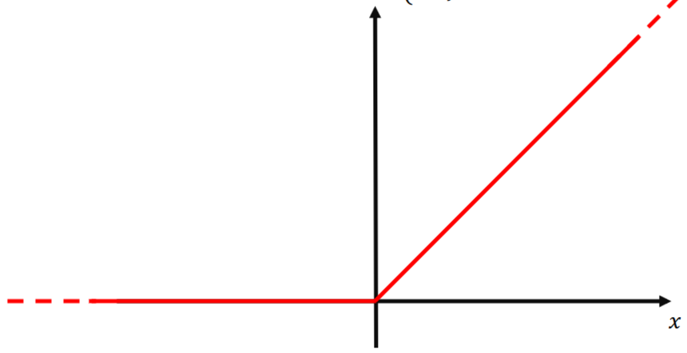


Figure 3.1. ReLU activation function mathematical expression and plot.

The pooling layer has the aim to down-sample the image in order to optimize the learning of features. In particular, the max pooling layer takes a portion of the image (typically a 2×2 window) and substitutes it with the maximum pixel value (Figure 3.4).

The performance of a CNN is represented by values of loss and accuracy, which are computed during training, validation and test processes. It is desirable to obtain similar accuracies between the training and validation phase to avoid overfitting. Avoiding the overfitting will mean to have generalized model performances against never seen data. There are different ways to overcome this issue. These include L2 weight regularization [31], batch normalization [32], data augmentation [33] and

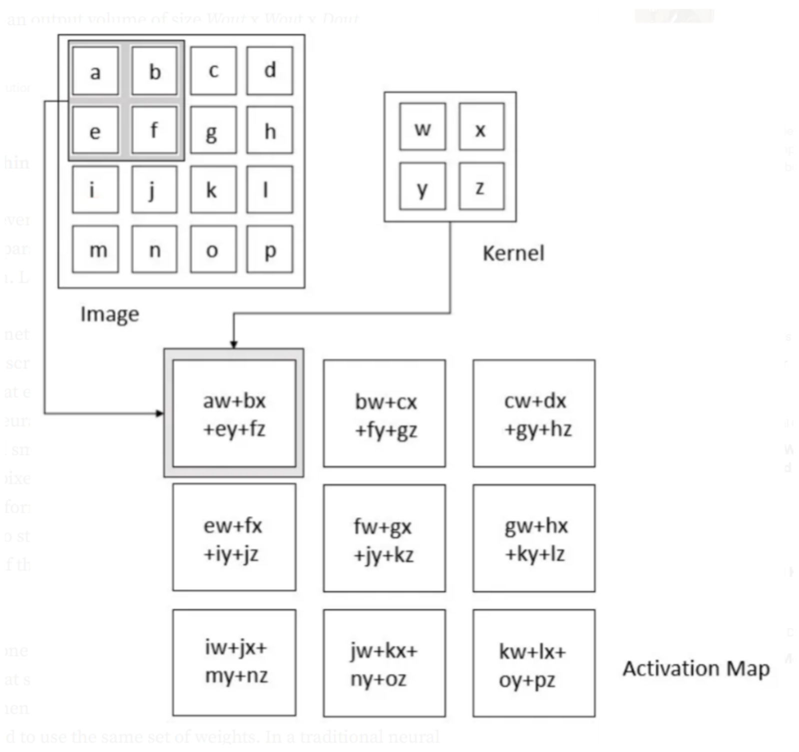


Figure 3.2. Convolution Operation [8].

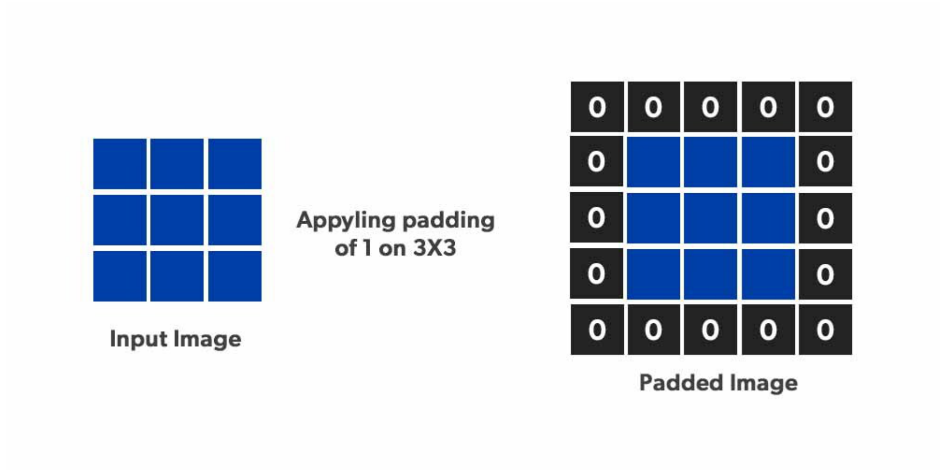


Figure 3.3. Padding Operation [9]

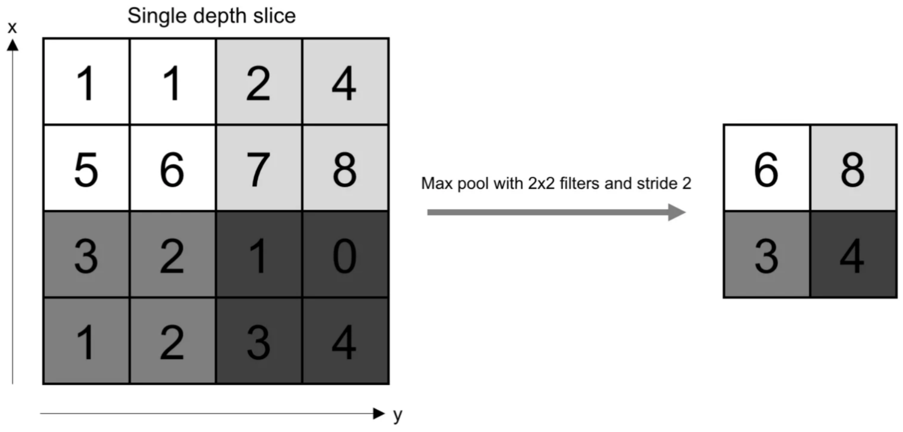


Figure 3.4. Maxpooling Operation [10]

dropout [34].

3.2 AI and Space: Research and Existing Applications

AI and in particular ML has achieved many successful tasks in the last decade. This has brought a huge interest in it, especially in the space industry and research. Anyway being ML capabilities discovered and tested recently, some time must be waited before founding them massively used in real space missions (robotic and human). This is both due to:

- The strong constraints on technology reliability to keep the risk of failure in a space mission really low;
- models developed within the NN are not human readable and have been impossible to replicate thus far [35].

Anyway, recently space companies and research teams have started to think about AI possible uses in the next future space missions. Some areas of application have been found:

- EO: AI and ML in particular are capable of analyzing autonomously a large amount of data. For example images coming from satellites for meteorological purposes, naval traffic, fires detection, agricultural monitoring etc. They need to be preprocessed, classified and complex information need to be extracted with repetitive processes that can be automated through AI. Even for non image data AI can be applied for signal anomaly detections or noise smoothing;
- Space exploration: Deep Space Probes (DSPs) and Rovers. These application can find help by AI in some cases as autonomous trajectory adjustment for collision avoidance purposes (DSP with a plate or asteroid, rover with craters or rock along the path). An examples of space missions involving AI are:
  - the Hera mission [35] for autonomous navigation application;

  - Mars latest rover for the intelligent data transmission where on-board software removes human scheduling errors.

- SN in FF missions and large constellation. In this application there is often the need of not having members of the group to collide each other and maintain a specific relative geometry. AI can be used in this kind of process to take decisions on how to maneuver spacecrafts in order to get the required group configuration while optimizing the propellant consumption;

- Spacecraft health state monitoring: AI can be used for monitoring the correct working of subsystems on-board and their performance degradation over time. SatGuard system, actually, is designed to extend the life of satellites deployed in orbit. Through the AI, "big data", DL, and ML technologies, the system is able to carry out anomalies and performance degradation detection. SatGuard's development is lead by Israeli Aerospace Industries (IAI) [36]. The main concern in space mission operation is to ensure the health and safety of the spacecraft. In the worst case the interruption of spacecraft functionality can lead to the loss of a mission but it often results in compromised mission objectives. Methods of system health monitoring are challenging especially in latest complex satellites and the available time to observe and interact with any given spacecraft is limited if compared to ground-based systems. This is due to the availability and bandwidth of their connection to ground, the availability of staff, communication latencies, and power budgets . These are the reason why every space-crafts need a minimum level of autonomy during their missions [37];

- Space telecommunication: data handling and transferring, coverage problem and its impact over the data transfer strategy suffer from several factors: limited on-board resources, satellites high speed, high complexity of models for terrestrial networks, the great heterogeneity between the satellite layers (GEO, MEO, LEO), the aerial layers (unmanned aerial vehicles etc.). Space Telecommunication offer a wide and challenging environment for AI research and applications. Some of them are related to satellite communication like beam hopping, telemetry mining, ionospheric scintillation detecting, and remote sensing. Moreover the Space Air Ground Integrated Networks (SAGINs) are an application where satellite and non satellite networks are integrated with AI to increase service flexibility. Table 3.1 illustrates the application of AI algorithms to solve different satellite communication problems.

3.2.1 RSOs Maneuver Detection and Estimate

Great concern exists for understanding and estimating performed maneuvers by active RSOs. This is important because a maneuvering spacecraft changes its orbital parameters and its future position. This could lead to include it as a new catalogue object or to associate it with a different one because of unsuccessful correlation procedure. This has to be avoided both to contain objects catalogue dimensions but also to avoid false collision warnings due to propagation of obsolete objects.

AI algorithm	Satellite communication application
SVM	Network traffic forecasting, channel modeling, telemetry mining, ionospheric scintillation detecting, managing interference, and remote sensing.
Decision Trees	Channel modeling, ionospheric scintillation, detecting, and remote sensing.
CNN	Channel modeling, remote sensing, space-air-ground integrating, handoff optimization, and carrier signal detection.
RNN	Anti-jamming, telemetry mining behavior modeling, and handoff optimization.
Auto Encoder (AE)	Managing interference.
Reinforcement Learning (RL)	Beam hopping, anti-jamming, managing interference, behavior modeling, space-air-ground integrating, and energy managing.

Table 3.1. Some of AI application in the Space Communication frame [20].

In this framework, R. Linares and R. Furfaro's [38] research aimed to use the inverse Reinforcement Learning to understand RSOs' behaviour in terms of maneuver policy to track them and predict their future position in order to avoid possible collisions. Another interesting work was done by R. Abay et al. to detect RSOs' maneuvers as orbit evolution anomalies using a developed Generative Adversarial Network (GAN) Figure 3.5. This method was then applied to a real existing GEO

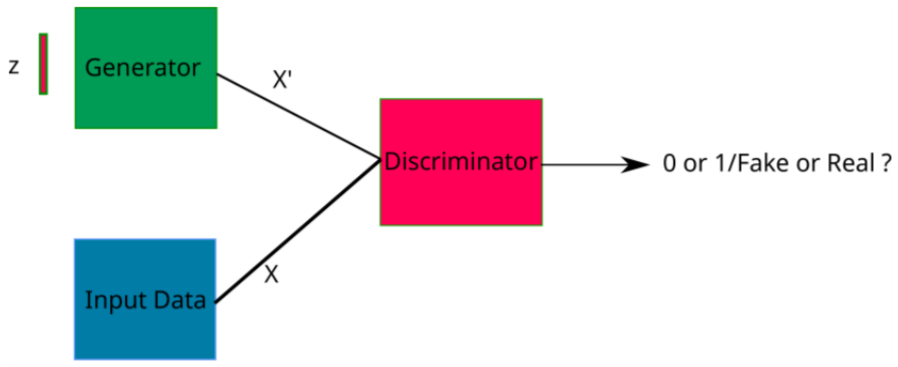


Figure 3.5. GAN typical architecture [11].

satellite and the network was able to provide good results in detecting the part of the station keeping maneuvers of the Optus 10 GEO satellite [11]. Previous works investigated the problem via the application of AI techniques but it is also possible to apply classical algebraic approaches and achieving good results as A. Pastor et al. did in their work [39]. Here two novel and operationally feasible methods, based on optimal control approach and weighted non least squares method were proposed. They were used to solve both the track-to-orbit problem for detecting and estimating a single burn maneuver and orbit-to-orbit problem to do the same with a multiple burns maneuver.

Despite the same context where the papers were born, Linares and Furfaro's one points to achieve something more significant for SSA than simple maneuver info because it aims to identify the maneuvering strategy which can lead to longer term predictions of the spacecraft location. Actually, for an active object a useful information to provide to the user through the catalogue could be the status of the spacecraft: Station keeping, Loss control due to engine failure, Transfer orbit and so on. It must be pointed out that the capability of estimating a performed maneuver of a satellite is still a useful information for SST purposes because it can be used to update RSO's mass and thus its ballistic coefficient, improving in such a way the characterization of the object itself. Moreover, the Abay and Pastor works make (explicitly or not) the assumption of being capable in impulsive burns maneuvers estimation, while Linares and Furfaro's work does not show the need of making hypotheses on the maneuver nature (low thrust or impulsive burn). This makes AI approaches more flexible with respect to the maneuver nature.

3.2.2 RSOs Shape/Properties Estimate

Another recent challenge in the SSA field is increasing the quality of RSO's information in terms of shape, reflective properties and angular rate. This is needed to improve the estimates of the acting forces like radiation pressure and aerodynamic drag. Actually, from a better knowledge of the acting forces, a better RSO's position can be predicted and a more reliable conjunction analysis can be performed. To face this problem, one possible approach is the use of RSO's light curves. By collecting RSO's frames from the on ground telescopes, it is possible to extract the object magnitude time evolution during the observation interval. From this input data, several approaches can be applied to estimate the shape, optical properties and angular rate of the RSO.

One recent work from R. Furfaro and R. Linares [12] shows that it is possible to use a CNN to invert a light curve for RSO's shape detection. In particular, four

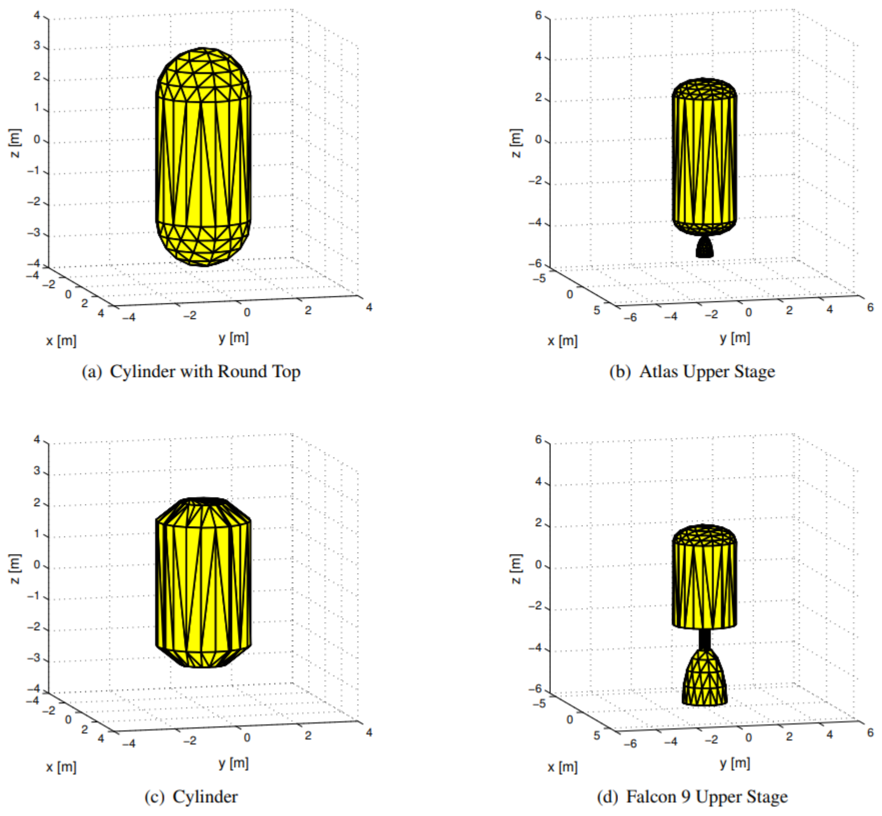


Figure 3.6. Four models of rocket bodies [12].

models of rocket upper stages (Figure 3.6) are modeled in terms of geometrical, dynamical and optical properties and then are used to generate 200000 light curves for the CNN training and validation step. In this work a simple CNN architecture made of three convolutional layers plus two fully connected was designed and used for solving a classification problem. Then trained network (Figure 3.7) performs well in recognizing RSO's shapes from the photo metric input data, achieving a validation accuracy of $92,2\%$.

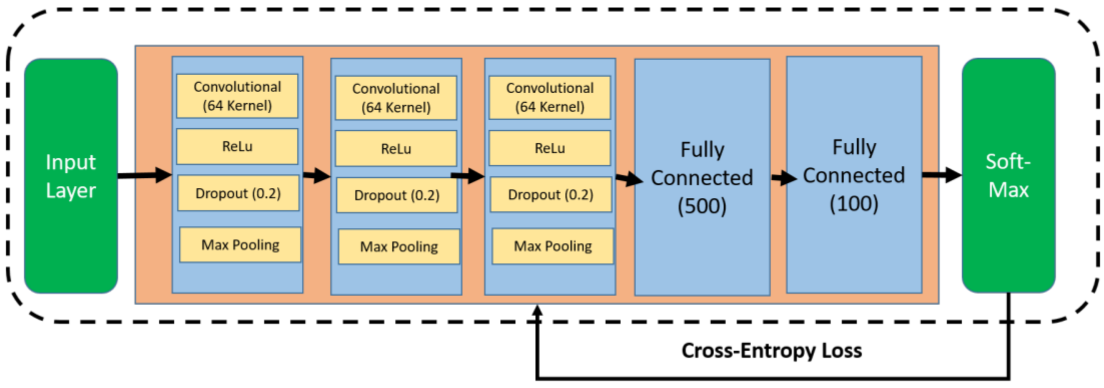


Figure 3.7. CNN architecture used in [12] for light curve inversion problem.

B. K. Bradley and P. Axelrad [40] tried to retrieve the shape of RSO's in the GEO belt using the Light Curve Inversion (LCI) technique developed by M. Kaasalainen and J. Torppa[41] for solving the problem with asteroids. This technique, which is different from the AI approach used in the previous work, seems to perform well for several objects like upper stage rocket bodies (Figure 3.8), 1U and 3U Cubesats while it under-performs with box-wing satellites and High Area to Mass Ratio (HAMR) objects.

Comparing these shape estimation approaches, it must be pointed out that both the CNN method and the classical LCI technique are tested against simulated data. Moreover, the solution of the LCI problem performed via NN is not affected by the ill-posed nature of the problem itself [12], if a well realized dataset is provided for the CNN training.

Advances in R. Linares, R. Furfaro et al.[42] show that the CNN significantly outperforms traditional ML techniques on a 3 class real data classification, achieving an accuracy of 75.4%. The authors argued that the many layer configuration of the CNN enables it to learn more complex decision boundaries, which suggests that the simulated light curve data does not effectively encapsulate the complexities present in the real light curve data. This represents a step forward compared to previous works [12, 40] because of the use of real light curves and their consequences over algorithms performances (improvement in accuracy).

A possible solution to this problem was proposed by J. Allworth, L. Windrim et al. [13] where the same data-driven method for the classification of light curve measurements of RSOs based on a DL approach was used. The original approach followed here is the use of transfer learning to make an existing trained CNN model achieve better performances when applied to real light curve for shape retrieval task.

Actually, authors use a pre-trained network over a huge simulated light curve dataset and then its weights are used to initialise a separate NN where the last layers are fine tuned with the training on smaller real light curve dataset (Figure 3.9). In this work both the models are tested against the same real light curve test set and the fine-tuned network shows an improvement in accuracy equal to 5%.

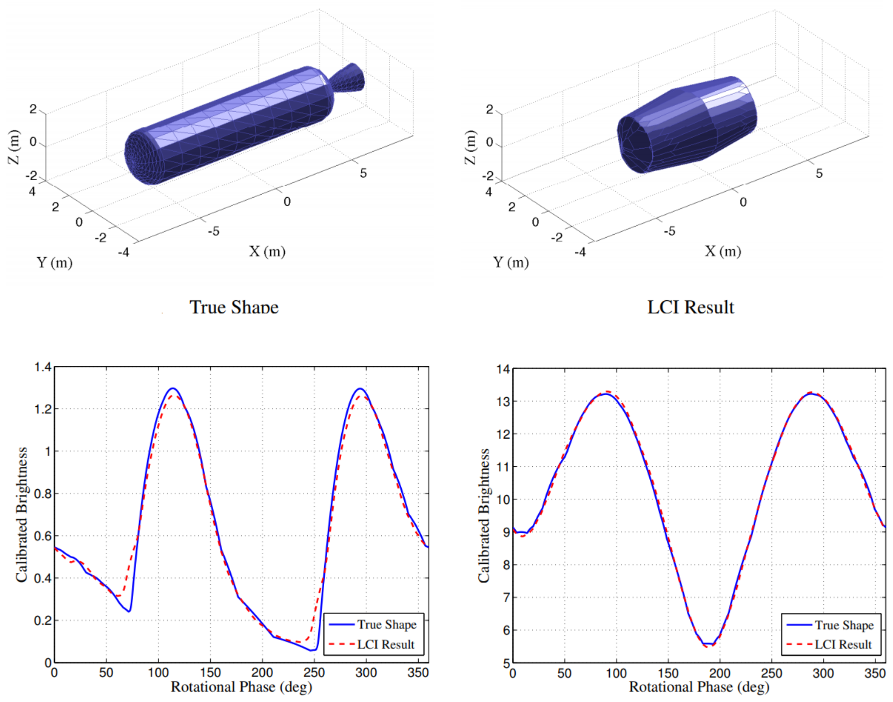


Figure 3.8. LCI problem applied to Centaur upper stage in [40]. Below the real and reconstructed shapes' Light Curves are compared for two different rotation axes.

3.2.3 AI Applied to SN

In the SN field, the main AI algorithms used are:

1. Fuzzy Control (FC);
2. NNs;
3. Evolutionary (Heuristic) optimization algorithms (such as Genetic Algorithms (GA));
4. Adaptive Neuro-fuzzy Inference Systems (ANFIS).

These different methods require different approaches. Actually, NNs need a training phase before being used for inference. The great concern is not just selecting or designing the proper model architecture for the desired purpose but, providing good and large-enough datasets. The problem of providing large and well calibrated datasets can be easily faced through the usage of simulations and computer implemented models to generate the suitable amount of input and output training samples. With a suitable training strategy and achieved generalized performances, NNs based algorithms could efficiently replace Guidance Navigation and Controls (GNC) algorithms if the inputs and outputs of GNC algorithms are used to train the AI algorithm. Their use would bring two main advantages:

1. The NNs faster (or input size independent [43]) execution time in-orbit when compared to non-AI GNC algorithms;

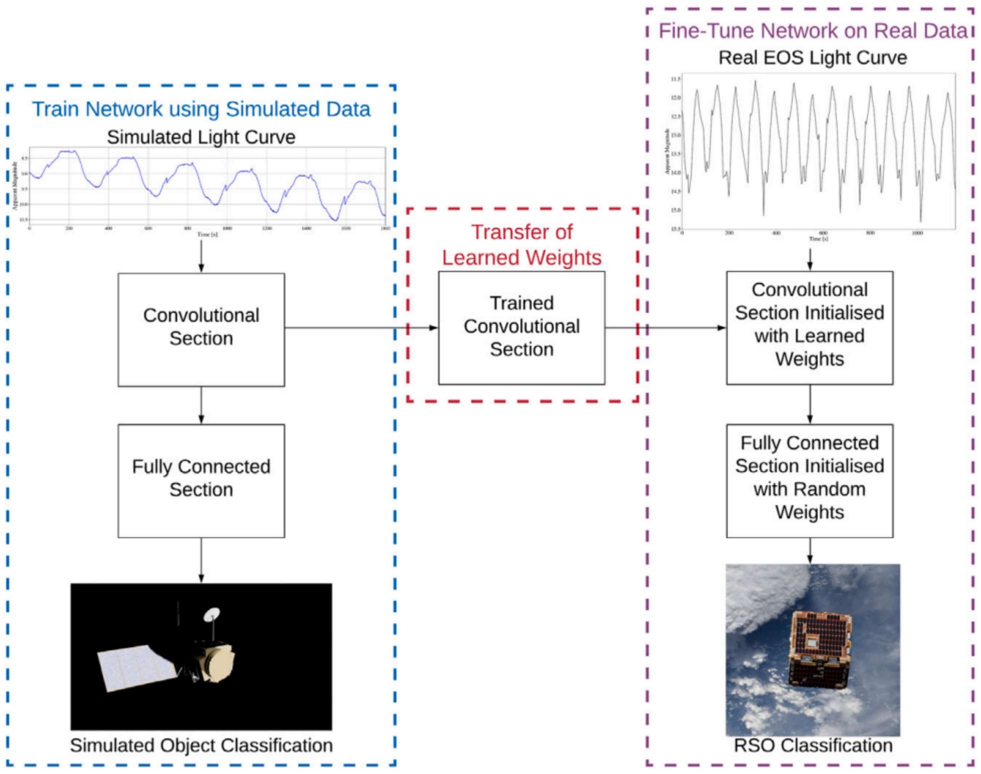

Train Network using Simulated Data
Simulated Light Curve
Convolutional Section
Fully Connected Section
Simulated Object Classification
Transfer of Learned Weights
Trained Convolutional Section
Fine-Tune Network on Real Data
Real EOS Light Curve
Convolutional Section Initialised with Learned Weights
Fully Connected Section Initialised with Random Weights
RSO Classification

Figure 3.9. Fine tuning approach used in [13] to increase the CNN classification accuracy including real data in the learning process.

2. NNs based algorithms could maintain the same accuracy levels of current GNC traditional algorithms [44].

In [14] an Extended Kalman Filter (EKF) coupled with NN (RBFNN) is utilized. The additive disturbance and nonlinearities are modelled through a Radial Basis Function NN (RBFNN). As the knowledge of additive disturbances improves, the EKF process noise covariance matrix is adaptively adjusted (Figure 3.10).

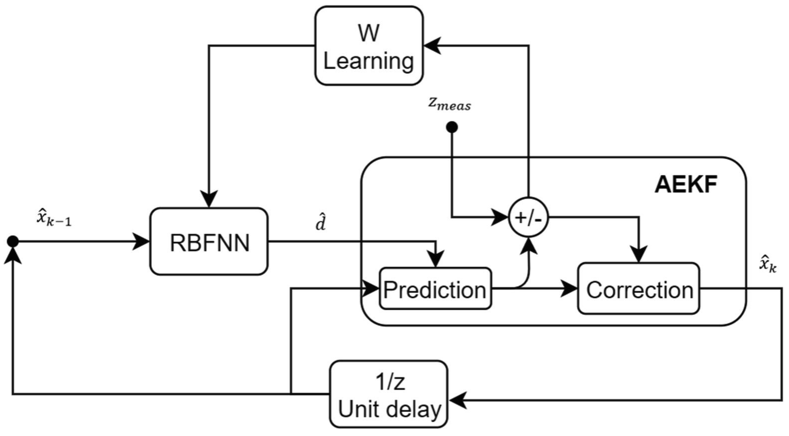


Figure 3.10. Navigation algorithm scheme from [14].

In [15] a novel AI-based algorithm solves the monocular pose estimation for close-proximity operations around an uncooperative spacecraft. It combines a Single HourGlass NN (SHGNN) for feature detection with a Covariant Efficient Procrustes Perspective-n-Points (CEPPnP) solver and a Multiplicative Extended Kalman Filter (MEKF) (Figure 3.11).

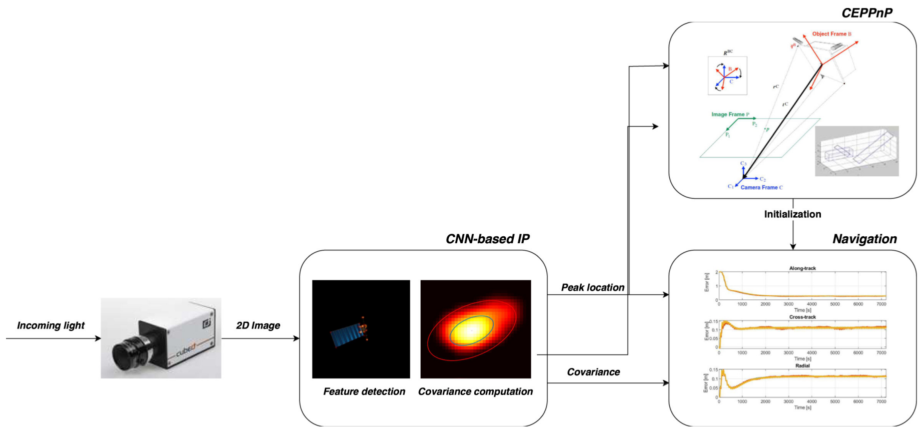


Figure 3.11. Navigation algorithm scheme from [15].

In [45] GA are used to optimize the learning function of a NN through the GA-NN merging. This led to a solution very close to the global optimum solution. Moreover, GA could be used to optimize the parameters of NN to achieve certain goals.

In the AN branch, the Lost In Space (LIS) is a heavy AI faced problem. Actually, from [46] a survey of LISAs is shown and described. The problem of LIS is often faced through ML by formulating the star identification process as a classification problem. For each star an encoded string representing one class is generated. When a star field image is provided, then a star is selected and the encoded string is generated. In the end a NN is used as classifier to identify the star [16].

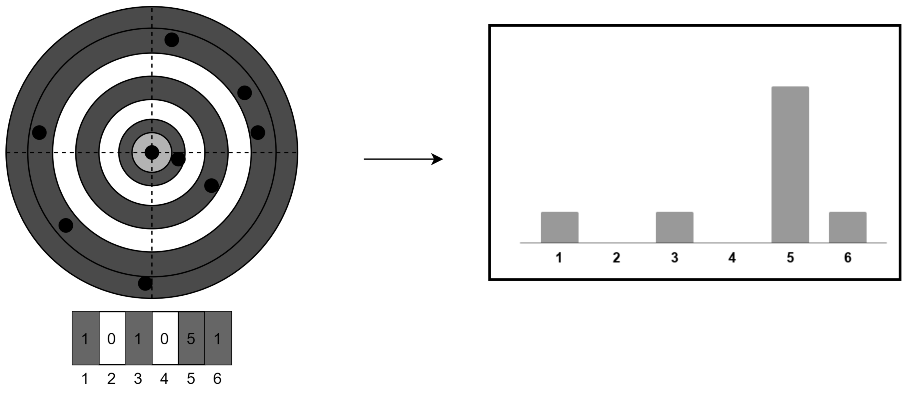


Figure 3.12. Star Pattern is being encoded in a numeric string by dividing the radial distance in intervals [16].

3.3 Machine Learning Models for Images Segmentation and Objects Detection in this work

This section is devoted to give an overview and to describe the three NNs which are used in this research activity: U-Net, YOLOv3 and YOLOv4. The U-Net is applied to solve the problem of night sky images segmentation for brightest objects information extraction. The YOLOs are used to localize and recognize the brighter streaks on the dark background. Effects and results of these models are shown in the following chapters.

3.3.1 U-Net

The U-Net is a fully convolutional neural network (FCN) which was first used for segmentation of biomedical images and was later adapted to space-based applications such as crater detection [47]. The network architecture in this paper is chosen to be a slightly modified version of the original one as shown in Figure 3.13. In this Figure the numbers on the left of each block represent the input tensor sizes (512, 256, 128 etc...) while the numbers below each block represent the number of filters in that block (1, 64, 64 for the three starting blocks). Arrows colours vary with the blocks' tasks: blue for convolution operation, red for MaxPooling, green for the UpConvolution operation, grey for the copy operation and brown for the layers concatenation.

The U-Net has a symmetric encoder-decoder structure: the encoding, down-sampling path is a stacked sequence of two ReLU 3×3 convolutional layers followed by a 2×2 max pooling layer. At each level, the number of filters doubles, reaching its maximum value at the bottom.

The decoding up-sampling path contains 2×2 up-convolutional layers concatenated with layers coming from the corresponding down-sampling level in order to preserve already learned features. The concatenated layers are followed by dropout and a sequence of two 3×3 convolutional layers up to the output layer in which a final sigmoid-activated 1×1 convolution produces data ready to be binarized. The present U-Net will be fed with 512×512 images and their corresponding masks.

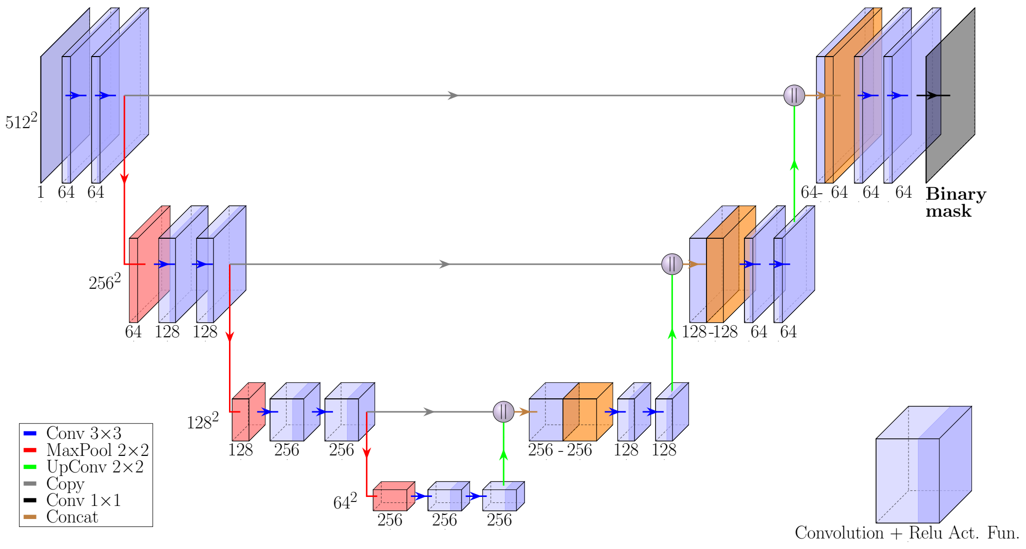


Figure 3.13. U-Net model with input tensor sizes and number of filters [17].

Configuration parameters used for this network to prevent overfitting are:

- Learning Rate: It is an optimizer's parameter that fixes the step size at each iteration during minimization of the loss function;
- Regularization Factor: It is a parameter needed for the l2 regularization based on penalization of the cost function;
- Dropout Rate: It regulates the percentage of inactive network elements in the dropout layers during training and validation process;
- Kernel initializer: It sets the weight initialization method; in this case, it is set on he_normal [48].

In this work, Adam [49] was chosen as the optimizer and Binary Crossentropy [50] was chosen as the loss function.

3.3.2 YOLOv3: You Only Look Once, Version 3

The You Only Look Once, Version 3 (YOLOv3) is a real-time object detection algorithm that detects objects in images and videos. The YOLOv1 was created in 2015, YOLOv2 in 2016 while YOLOv3 in 2018 [51]. YOLOv3 is an improved version of YOLOv1 and YOLOv2 and is implemented using the Keras or OpenCV deep learning libraries. YOLOv3 performs Objects Detection (OD) which is composed of two steps: Objects Localization (OL) and Objects Classification (OC). The former aims to localize and thus estimate the position and extension of the object within an image while the latter aims to specify the class (or kind) of localized objects. In this way, OD means knowledge of position and type of objects at the same time.

YOLOv3 is a CNN capable of performing real-time OD using a structure which is shown in Figure 3.14.

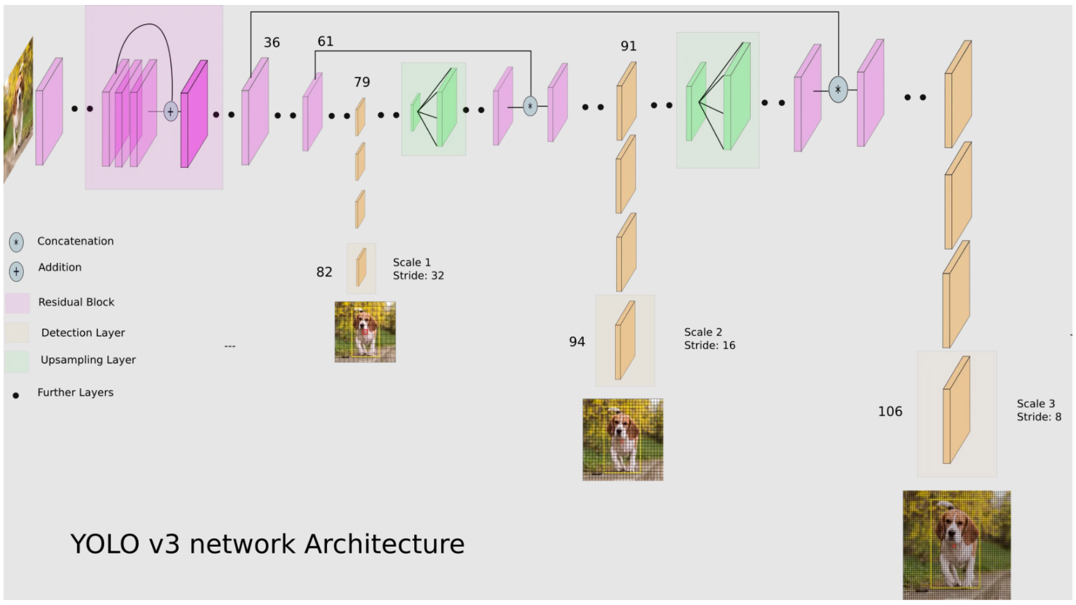


Figure 3.14. YOLOv3 architecture [18].

The first part of YOLOv3 is composed of a Darknet 53 network who was previously trained on the ImageNet dataset [52]. Darknet 53 is represented in the picture (magenta) in the left side of Figure 3.14 and is composed of 53 convolution layers. For the task of detection, 53 more layers are stacked onto it, giving a total amount of 106 fully convolutional layers, underlying architecture for YOLOv3 . The detection layers are the 82^{nd}, 94^{th} and 106^{th} ones (right part of Figure 3.14, vertical structures) and they are given the same input image but with different sizes due to different spatial scale for OD to be performed.

The input image size for the 82^{nd} layer is scaled w.r.t. the network input by a factor of 32. The 94^{th} input is scaled by a factor of 16 and the 106^{th} input image has a scale factor of 8. In this way the detection is performed at three different scales into the same image: the 82^{nd} layer is responsible for wide objects detections while the 106^{th} is responsible for small objects detections. At each of the 3 prediction layers, the network performs detection using 1×1 convolution. This has as consequence that features maps and prediction maps have the same size and for this reason the algorithm was called YOLO: You Only Look Once.

How Does OD Work?

If we consider a network input image of 416 $\times$ 416, the 82^{nd} layer input will have a size of 13 $\times$ 13. On every single pixel (or *cell*) of this input a detection kernel will be applied. Its effect will be the production of a feature map whose size is 13 $\times$ 13 $\times D$, where the *Depth* "D" is given by the formula:

$$D = b \times (5 + c) \tag{3.1}$$

which contains the number of classes "c" and the number of predicted bounding boxes "b" which is equal to 3 for the considered network. The fixed number 5 stands for the five values associated to the box: two coordinates, two box sizes and an Objectiveness score which represents presence/absence of any object. The cell containing the center of the object that cell belongs to is thus responsible for detecting the object. In the framework of this thesis activity the detection problem will require just a single class and thus the Depth will be equal to 18.

To predict Bounding Boxes (BBs), "anchor boxes" are used. YOLOv3 uses predefined BBs called anchors or priors which are used to calculate real width and real height for predicted BBs. An amount of 9 anchor boxes are used, 3 anchor boxes for each input scale. This means that at each detection layer every grid scale of feature map can predict 3 bounding boxes using 3 anchor boxes (this is the reason why b is equal to 3 in the Equation 3.1). To calculate these anchors K-Means Clustering is applied in YOLOv3 and after this computation the real BBs prediction comes using a log-space transform:

$$\begin{cases} b_x &= \sigma(t_x) \, + \, c_x \\ b_y &= \sigma(t_y) \, + \, c_y \\ b_w &= p_w \, e^{t_w} \\ b_h &= p_h \, e^{t_h} \end{cases} \tag{3.2}$$

where (c_x, c_y) are the cell's top left corner of the anchor box, (t_x, t_y) are the NN predicted box center, (t_w, t_h) are the NN predicted box dimensions and (p_w, p_h) are the anchor box dimensions (width and height). With these value and equations is possible to compute the BB's center coordinates (b_x, b_y) and dimensions (b_w, b_h). After this operation, the total amount of predicted BBs for the three detection layers (considering a 416 $\times$ 416) is: $(13 \times 13 \times 3) + (26 \times 26 \times 3) + (52 \times 52 \times 3) = 10647$. All these redundant bounding boxes from 10,647 boxes are then suppressed using Non-Max Suppression (NMS) algorithm. NMS filters the redundant boxes, leaving just the desired and final ones. The algorithm can be summarized in this way briefly:

1. The proposal with highest confidence score is removed from list of predicted BBs and is added to the final list;

2. The final candidate list is compared with all the proposals in the predicted BBs list. If the comparison index is greater than a threshold, then proposal is removed from predicted BBs list;

3. Again the predicted BBs proposal with the highest confidence from the remaining proposals is removed and added in the final list;

4. With this new final list proposal, once again the comparison index is computed with all the predicted BBs proposals. All the predicted BBs proposals with a comparison index higher than the treshold are removed;
5. This process is repeated until there are no more predicted BBs proposals.

As comparison index, the Intersection Over Union (IOU) is used. It is defined as the ratio between the intersection area and total area between two areas.

$$IOU = \frac{Intersection\ Area}{Total\ Area} \tag{3.3}$$

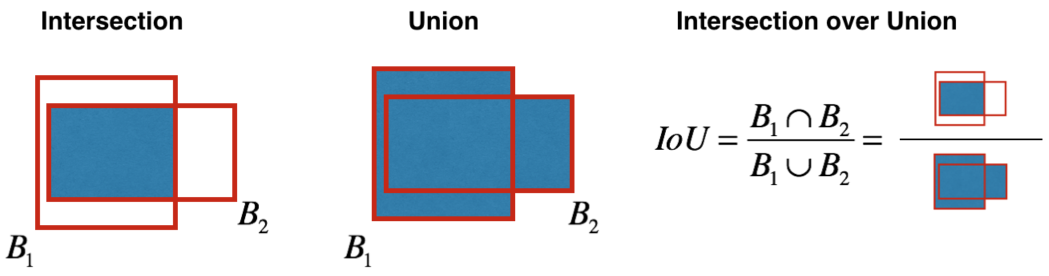


Figure 3.15. IOU graphical explanation [19].

3.3.3 YOLOv4, What's New?

The YOLOv4 is a one-stage detector (Figure 3.16) with several added components developed by Alexey Bochkovskiy et al. [53]. Both YOLOv3 and v4 are object detection models, and in the v4 there is a significant increase in the Average Precision (AP), at the same frame rate, w.r.t its predecessor (Figure 3.17).

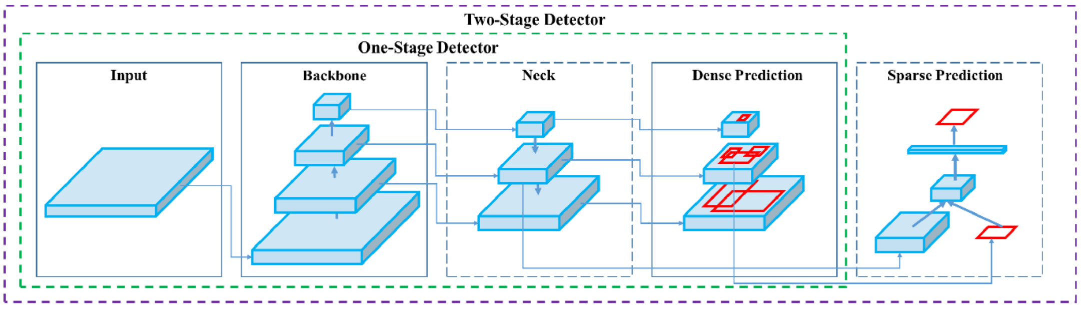


Figure 3.16. Object Detector models structure [53]. YOLOv4 is one stage detector w.r.t. the classical two stage detectors like Faster-RCNNs (Region based CNNs)[54], Mask-RCNNs [55] etc..

As every objects detector model, the YOLOv4 is made of these blocks too:

- **Input**: Image, patches, Pyramid;
- **Backbone**: It is a deep learning model that constitutes the feature extractor and which is trained on ImageNet. Generally, the backbone is a classification

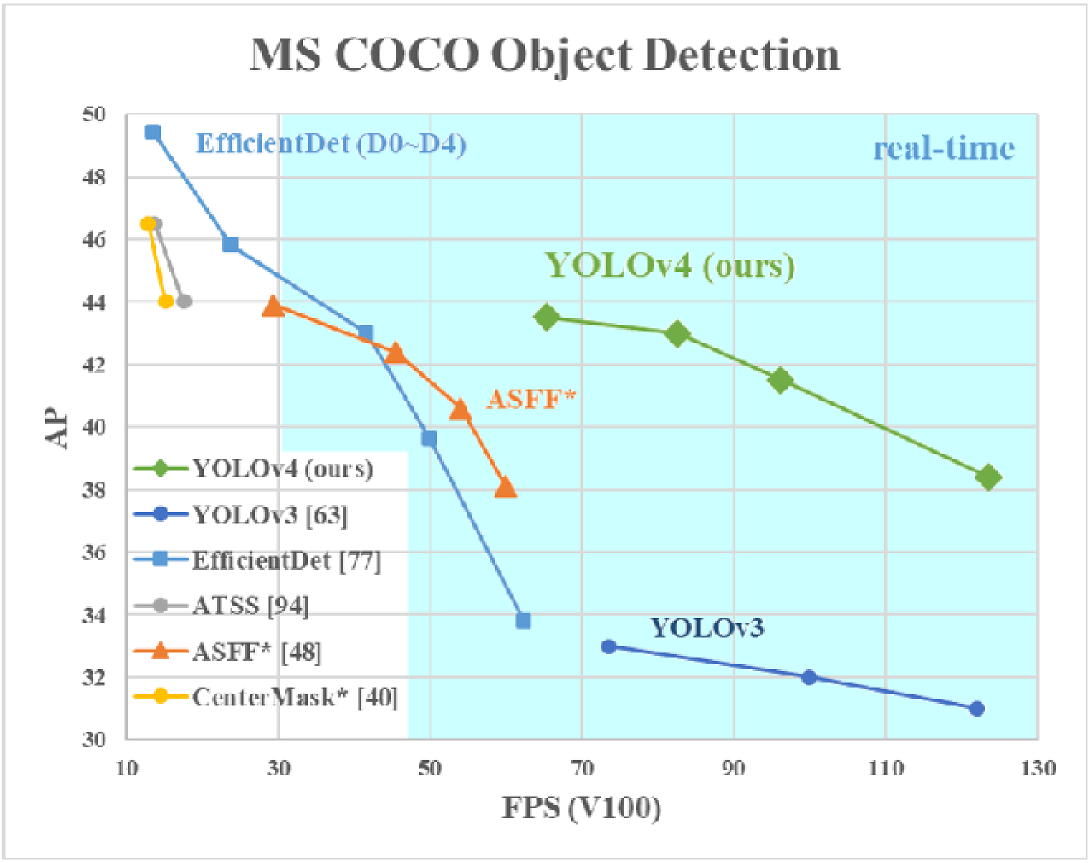


Figure 3.17. Object Detector models comparison over Microsoft Common Objects in Context (MS COCO) Dataset [56]: AP vs Frame Per Second (FPS) [53]. YOLOv4 significantly outperforms the v3 in terms of AP.

models like the VGG16 (Visual Geometry Group 16)[57], ResNet [58], or DenseNet [59];

- **Neck**: It is made of methods which collect feature maps from different stages of the backbone. Basically, it is a feature aggregator. Its existence is common but not mandatory. Usually, a neck is composed of several bottom-up paths and several topdown paths [53];
- **Head**: It is the real object detector. It localizes and classifies the object. Several models can be used as head and they are divided according to the usage or not of the anchors:
  - Anchor Based: RPN (Region Proposal Networks) [54], SSD (Single Shot multibox Detector) [60], YOLO [18], RetinaNet [61];
  - Anchor Free: CornerNet [62], CenterNet [63], MatrixNet [64], FCOS (Fully Convolutional One-Stage object detection) [65].

This is the general structure of a DL object detector, moreover YOLOv4 does contain a set of methods which can enhance the network performance. They are divided into two sets:

- **Bag of specials (BoS)**: They are just a group of "plugin modules" and post-processing methods that increase the inference cost by a small amount but can significantly improve the accuracy. The plugin modules are used for enhancing certain attributes in a model, such as enlarging receptive field, introducing attention mechanism, or strengthening feature integration capability, etc., and post-processing is a method for screening model prediction results [53]. Common modules that can be used to enhance receptive field are SPP (Spatial Pyramid Pooling) [66], ASPP [67], and RFB (Receptive Field Block) [68].
- **Bag of Freebies (BoF)**: They are a set of methods that only change the training strategy or only increase the training cost (nothing to do with inference). For example the data augmentation, regularization, class label smoothing are some of them.

YOLOv4 Architecture

Previously the standard structure of one stage or two stages objects detectors have been described and shown underlining the various possibilities in the models choices for every part of it: BackBone, Neck and Head. The original configuration of YOLOv4 is:

- **Backbone**: CSPDarkNet53;
- **Neck**: SSP [66] + PANet [69];
- **Head**: YOLOv3 (YOLO layers are the 139^{th}, 150^{th} and the 161^{st}).

As BoS and BoF used for this architector, they are:

- **Backbone**:

  - BoS: Mish activation, Cross-stage partial connections (CSP), Multi- input weighted residual connections (MiWRC);
  - BoF: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing.

- **Head**:
  - BoS: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS;
  - BoF: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyper-parameters, Random training shapes.

This is the architecture that has been used in my PhD research activity and the benefits this network has brought will be shown in the next chapters.

Chapter 4

AI Based On-Board Image Processing

This chapter is devoted to investigate and face the main core of my PhD thesis: the problem of Image processing through the use of AI. Image processing is a mandatory step to achieve tasks as complex information extraction from images, the RSOs position over the time in my case. Because of this, it is needed to design and to develop an algorithm capable of tracking RSOs over the time. To reach this scope through the use of AI, I faced the problem of night sky objects segmentation (Section 4.1). This was done with the use of the U-Net CNN to get a segmented (binarized) image. With a binarized image the RSOs appear as clear streaks on dark background. Thus, it is possible to train a YOLOv3 model to detect streaks on the night sky to achieve the objects detection capabilities (Sections 4.2 and 4.2.2). At this point streaks' positions (centroids) can be computed.

If the image segmentation and streaks detection process is applied for each frame of an images sequence, it is possible to know the streaks centroids history at each frame. The history (or the whole collection of centroids) knowledge is used by a developed Tracking algorithm (Section 4.2.2) to gather centroids which belong to the same object. In this way, the RSOs tracking or clear streaks tracking on the dark sky background is possible.

4.1 Image Segmentation

In the framework of on-board autonomous star sensors algorithms, star segmentation and accurate centroid estimation are critical problems to face. The number of actual detected stars and the relative centroids' estimation accuracy have a huge impact on the success of star-identification-based attitude determination routines [70, 71]. Taking into account that star sensor images generally contain several noise sources (stray light noise, single point noise, single events upsets (SEUs) and so on), good initial processing of image data is mandatory. In this way, it is possible to improve the percentage of success in the star identification process [16, 72] and the possibility of retrieving accurate attitude information from images. A good image segmentation process is also needed for RSOs detection. In this case the image noise and the weak nature of an RSO's streak can often lead to an incorrect detection of the object and loss of precious information for the orbit determination modules.

The process of image segmentation is needed to highlight the useful image information which is geometrically encoded in the image pixels. Normally, not all the pixels in an image are needed, and a process of selection and exclusion is mandatory to focus attention on just a part of the image. Image segmentation faces and solves this problem bringing as a result a binarized image which is often called Mask. Through the convolution of this Mask with the original image, it is possible to extract photometric and geometrical information about the target pixels. First of all we focus on the geometrical information, the photometric information instead will be discussed in the next sections/paragraphs.

Regarding image segmentation, in the last decades several traditional approaches have been proposed. They are classified according to the element they take into account to discern foreground and background pixels: image histograms, detection of image gradients, detection of object edges and complexity analysis techniques [70].

Among them, commonly used methods are the following:

- Iterative threshold method [73]: It performs an iterative optimization. An initial threshold is set, and then the algorithm improves the estimation at every step with a suitable improvement strategy. The strategy should be fast enough in convergence and should improve the quality of the segmented image at each step.

- Local threshold methods: They are based on the selection of an initial threshold value plus a margin. For a generic examined pixel, its value is compared with a reference one. This reference value is updated continuously and takes into account the local energy value of the surrounding pixels. The local energy value is computed through a criterion which characterizes the segmentation method. A local threshold method is described in [74], and it is based on a moving streak average, while in [75], the approach uses rectangular areas and the local contrast level.

- Otsu's algorithm [76]: It performs a more refined approach where the intraclass variance of the foreground and background pixels is considered as a cost index and where the selected threshold comes from its maximization.

- Niblack's method [77]: It is a localized thresholding algorithm in which the threshold is varied across the image as a function of the local intensity average and standard deviation.

Each reported traditional method presents limitations. Iterative threshold methods rely on all the energy levels in the global image, and these constitute their limit: a simple strong stray-light scenario can badly affect the predictions of these methods. Moreover, when the histogram of a star image is unimodal, the traditional iterative threshold method is time-consuming and cannot reach a suitable segmentation. A local threshold method would fail in the same way if the difference between the star signal and the local noise intensity is lower than the margin. The Otsu method, instead, can only provide satisfactory results for thresholding if the histogram is not unimodal or close to unimodal [78]. AI could be an appealing and useful tool to face this challenging problem due to the recent development and application of AI in image processing.

Star segmentation is a task that has many aspects in common with the segmentation of dim small targets, whose primary goal is to enhance the contrast between the target itself and the background, regardless of its nature; segmentation must be performed in every possible situation (noise, blurring, angular motion of the camera and/or motion of the target).

All these issues also occur in ST images. Dim small target segmentation has been studied for both synthetic aperture radar (SAR) and optical images. Jin et al. [79] proposed the application of a lightweight patch-to-pixel (P2P) CNN for ship detection in PolSAR (Polarimetric Synthetic Aperture Radar) images. Their approach prefers the use of dilated convolutional layers rather than conventional ones in order to expand the receptive field without adding any model parameter. Zhao and Jia [80] employed a more general-purpose target segmentation using a CNN on infrared images, testing it on both real and synthetic images; however, their study suffers from the necessity for a large amount of training data to be fed to the network. Fan et al. [81] overcame the issue of large training data by using the Modified National Institute of Standards and Technology (MNIST) [82] database and simulating images which have similar properties to the long-range infrared images. Nasrabadi [83] proposed an autonomous target recognition in Forward Looking InfraRed (FLIR) images based on different deep CNN architectures and thresholding. However, this approach suffers from a lack of reliability in a real-world scenario. Shi and Wang [84] based their studies on the use of a CNN and a denoising autoencoder network by treating the small targets as background noise and therefore transforming the segmentation task into a denoising task.

The use of the famous biomedical-based U-Net for small target segmentation tasks has not been broadly investigated as of this work. Tong et al. [85] proposed an enhanced symmetric attention U-Net, which employs information extracted in the same layer to focus on the target and cross-layer information to learn higher level features. Xue et al. [86, 87] investigated a more specialized version of the small target segmentation problem by going deeper into the problem of segmenting star images by proposing StarNet. This network is particularly complex due to the fact that its training is carried out in a multistage approach, an aspect that affects the training time in a negative way. In addition to this issue, StarNet is pre-trained on weights taken from the first three stages of VGG16 [57], whose initial input size must be a three-channel image. This forced choice translates into a computationally heavy process that affects the prediction time and could require top choice GPUs. All these aspects seem to make real-time image processing unfeasible. The U-Net proposed in this work tries to overcome this problem by working with mono-channel images by exploiting a simpler network architecture and training strategy.

What I did facing the night sky image segmentation algorithm was the development of an AI-based image segmentation module of a proposed ST-based payload for attitude determination (and lately this was used by me in the RSOs detection framework). This algorithm was then proposed as software to be integrated with a hardware to create a payload for on-board optical sensor AI-based applications for small satellite missions. The choice of proposing an AI-based segmentation algorithm guarantees a high segmentation quality of ST images against a variety of noise scenarios without the need for intensive calibration activity. Actually, it represents a robust processing solution for commercial and cheap STs which can be used in small platform missions.

In the next sections the design of an AI-based star and RSO segmentation algorithm will be shown. It is capable of filtering the faintest objects from background in several Signal-to-Noise-Ratio (SNR) level scenarios (stray light noise, ghost noise, SEUs). In particular this algorithm aims to provide information about the brightest objects within the FOV in order to facilitate star identification and object detection and reduce the memory storage burden of a ST for successive operations. By brightest objects, I mean the ones that stand out most inside an image. Here, the formulation of the segmentation problem is similar to the creation of a saliency map. In the predicted mask, just the most salient objects will appear. This philosophy seems to be suitable to detect stars and objects in low SNR environments as strong stray light noise that could affect the star sensor image. The algorithm design aims to have few modules to reduce the algorithm size, complexity and hardware implementation difficulties in order to provide a reliable star detection product. In the IOV proposal section I will describe an integrated hardware and software payload (design, optical head specifications, component choice and validation mission) for a small satellite application.

The ML world was investigated to take advantage of the NNs capability of learning specific tasks, performing well against unpredictable situations and reducing algorithm complexity for segmentation purposes; this could avoid the problem of algorithm re-calibration to successfully process images in different noise conditions. Moreover, the CNNs are considered for this kind of task. They have the advantage of processing the image directly without iterative steps and without a local approach that would be power and time-consuming. All of these features are designed along with the capability of resolving objects in strong stray-light environments if a suitable dataset for training is available.

In the process of designing and validating such an algorithm several steps have been completed:

- Creation of a balanced dataset of real and simulated star images for training, validation and testing of the CNN block.
- Design of an AI-based brightest objects detection algorithm capable of providing the most significant information for star identification and objects tracking algorithms.
- Comparison of our proposed AI-based algorithms with traditional segmentation algorithms.
- Validation of the AI algorithm both with ST simulated images and real images.
- Proposal of a an electro-optical payload for brightest objects segmentation in the onboard night sky images and a validation mission on a small platform.

Next sections are organized as follows: firstly, I show the creation and description of the proposed dataset (Section 4.1.1). Secondly, the segmentation-clustering algorithm scheme is presented (Section 4.1.2). Then, a description of the traditional algorithms used for comparison follows, along with a description of the considered performance indices (Section 4.1.2). Results of U-Net's training, validation and testing are then reported and compared with traditional algorithms (Section 4.1.3). They are followed by a discussion part about the obtained results. In the end, a summary of the achievements is presented (Section 4.1.4).

4.1.1 Dataset Creation and Description

The dataset created and used for the U-Net training phase is publicly provided at [88]. It is composed of 5600 squared monochromatic jpeg images and 5600 associated jpeg masks. All of them have a size of 512 × 512 px and a bit depth of 8. Masks' pixels can have one of two possible values: 0 for background and 255 for foreground pixels. The samples for the training and validation phases are the first 5300 ones. They are generated from a batch of 600 images to which 180°clockwise rotation, 90° clockwise rotation, added noise, increased stray light, blur effects, increased luminosity, contrast and further modifications have been applied. This original batch is made of 300 samples coming from several night sky acquisition campaigns I personally led on the field and 300 samples which I randomly acquired using Stellarium [89] software in night sky conditions considering different FOV and attitudes (Figure 4.1). In the acquisition campaign I led, several kinds of cameras and objectives were used: ZWO ASI 120MM-S camera with default optics, reflex Nikon D3100 equipped with a Nikkor 18-105 mm and a ProLine PL16803 camera coupled with an Officina Stellare RiFast 400 telescope.

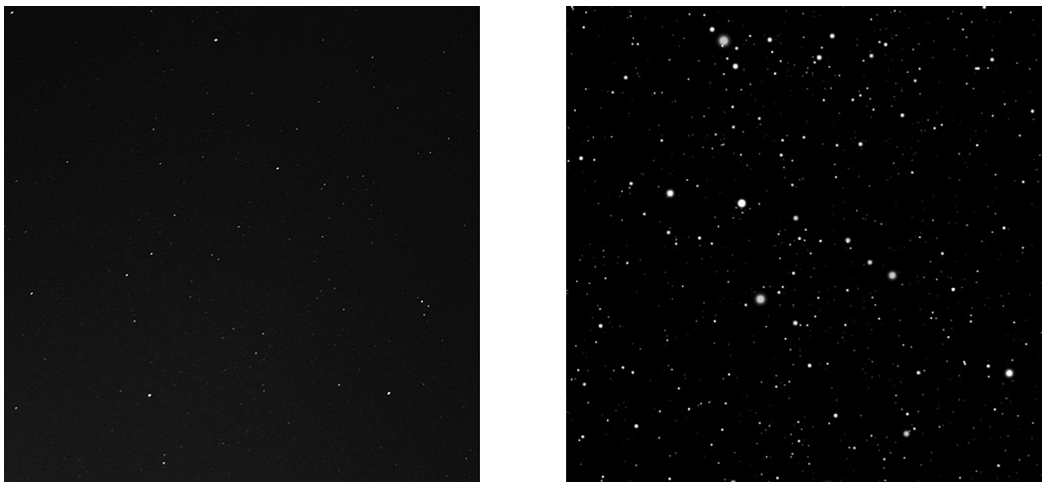


Figure 4.1. Dataset samples: real night sky image (on the left) and Stellarium-generated night sky (on the right).

Last 300 dataset images were obtained with a HFSTS. It is used to simulate realistic night sky images, simulating all the physical, functional and geometrical characteristics of the star sensor, along with all the instrumental and environmental noises [90, 91, 92, 93, 24]. Simulator images were used only in the test phase to provide independent samples the network has never been trained on to monitor the generalization capability of the trained network.

The dataset contains images of night sky where stars are the main actors but where planets, light pollution caused stray light noise, SEUs, clouds, airplanes' streaks, satellites, comets, nebulae and galaxies appear as well. Masks were obtained using specific thresholds for different groups of images from different campaigns, sensors and software. In particular, mask creation was carried out using Adobe Photoshop CC 2019 with the scope of obtaining just the most salient objects in the FOV and filtering all the noise and undesired objects cited above. An example of a dataset image and mask is provided in Figure 4.2, where it is clearly visible that just the brightest points appear inside the mask. Input images and mask preprocessing is performed in order to organize these data in two float 4D tensors for NN training and testing. Every image is converted into floating point arrays and normalized using its maximum value to carry out an image adaptive normalization process; every most significant image pixel will have an associated value close to 1. This will help the network detect the most significant pixels and make the learning process easier for the net by constraining the interval of signal values between 0 and 1. Moreover, this normalization process improves the algorithm capability of working with images that have different range of energy levels. The same process applies for mask data vectorization and normalization, with the only difference being that the normalizing factor is constant and equal to 255 for every mask.

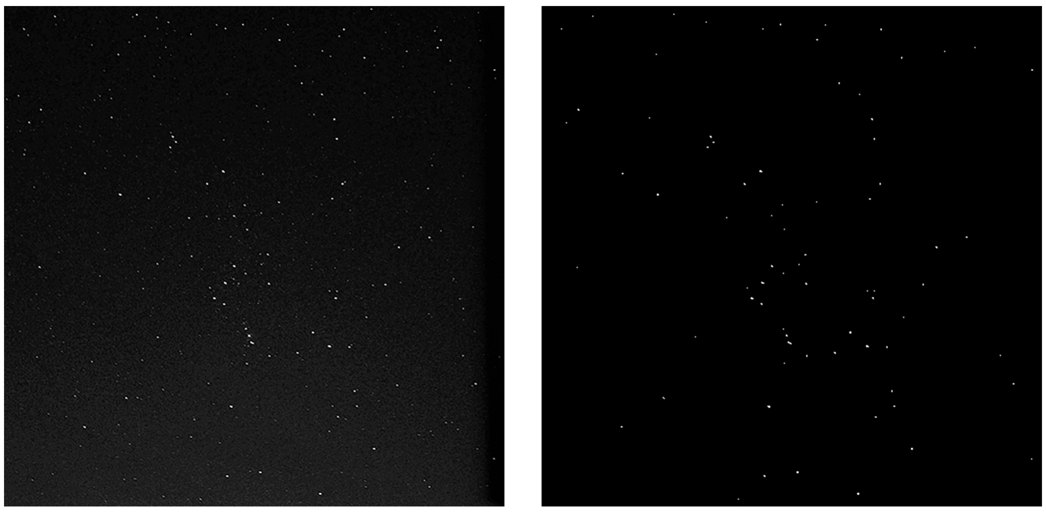


Figure 4.2. Dataset samples: real night sky image (on the left) and relative mask (on the right).

It must be pointed out that the dataset does not contain only point-like stars but also images in which appear as streaks due to the angular speed of the sensor. Considered angular speeds are within the $[0°/s, 1°/s]$. This allows the proposed AI-based algorithm to work properly even when the star sensor is rotating with respect to the fixed stars or when a sidereal pointing of the camera occurs together with RSOs passages.

Concerning the noise, 5300 images for training and validation are generated from a batch of 300 images. This batch is composed of several acquisitions performed with different sensors and so, different noise conditions. In less-noise-corrupted images, an added source of uniform noise was added using Adobe Photoshop 2019

(Filter->Noise->Add Noise->Uniform 6.3%). For different stray light conditions, +0.5, +1 and +1.5 stop in image exposure were added randomly. The used sensors' noise level characterization can't be provided due to the different kind of sensors, calibration level used and the unavailability of them in my department laboratory. For the last 300 test images, it is possible to characterize the noise:

- Shot Noise: Poisson probability model and proportional to square root of the detected signal;
- ReadOut Noise (RON): Normal distribution with mean 0 $e-$ and standard deviation of 82 $e-$;
- Dark Current (DC): Constant value of 550 $\frac{e-}{px \times sec}$;
- Dark Signal Non-Uniformity (DSNU): Normal distribution with 1 mean value and a standard deviation of 0.065;
- Photo Response Non-Uniformity (PRNU): Normal distribution with 1 mean value and a standard deviation of 0.01;
- Stray Light: 30000 $\frac{e-}{sec}$.

4.1.2 Algorithm Design and Configuration

In this section, the proposed AI-based algorithm's scheme called Brightest Objects Sky Segmentation (BOSS) is described and shown (Figure 4.3).

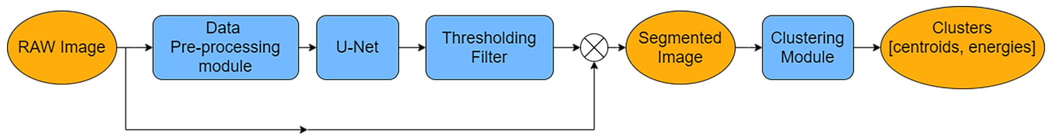


Figure 4.3. BOSS algorithm scheme: data are in orange, and modules are in light blue.

The monochromatic raw image from the camera arrives at the preprocessing module in which it will be resized, normalized and vectorized into a floating point array. This module will be tailored according to specific sensors' bit depth, while the size must be the one the NN is trained on. Then, the array is processed by the trained U-Net which gives its output prediction. This is a 2D array in which every pixel has an associated value of probability p of being foreground. If $p \geq p_{min}$, the value is rounded up to 1. After the thresholding filter, a binary image is obtained, and its element product with the original image will provide an array with just active pixels' energy values. This output will be used by the clustering algorithm to detect clusters and compute their centroids and total energies. The clustering module developed within this work takes all the active pixels in the segmented image and organizes them in a list. Then, it associates the first pixel in the list to the first cluster and starts to verify if the next pixels are part of the same cluster or not through a distance-based criterion. Whenever the next active pixel does not belong to the already identified clusters, then a new cluster is identified.

The distance criterion uses the Euclidean norm and the condition to assess the membership of two active pixels ($\mathbf{p}_i$ and $\mathbf{p}_j$) to the same cluster as expressed by Equation (4.1).

$$\|\mathbf{p}_i - \mathbf{p}_j\| \leq \sqrt{2} \tag{4.1}$$

Then, the clustering algorithm then first performs a filtering action with a minimum and maximum dimension and then a sorting of all the clusters. A minimum dimension filter is needed to avoid a possible noise signal in the clustering outputs (ex. hot pixels), while the maximum dimension filter is needed to avoid great detected clusters due to non-stars objects.

The sorting operation is then performed in descending order and considers the first N clusters in output from the previous filtering actions. It is based on a combination (Equation (4.2)) of the clusters' dimension ($dim_{cluster}$), energy ($E_{cluster}$) and maximum energy value ($max(E(\mathbf{p}_i))$) over the cluster because this sorting index proved to be suitable to select the best clusters for star pattern recognition purposes with the conducted tests.

$$Sorting\,Index = \frac{E_{cluster}}{dim_{cluster}} \times max(E(\mathbf{p}_i)) \;\; with\,i\,over\,cluster \tag{4.2}$$

N is computed by applying a 1.25 factor to the average number of stars which depends on the FOV and cut-off magnitude of the selected sensor [71]. The maximum number of clusters in output is fixed a priori to process just the most significant ones, and the 25% margin factor is selected to take into account possible uncertainties of the average number of stars formula.

The U-Net is capable of resolving background and foreground pixels after training, validation and test phases, but it still gives continuous values that have to be discretized. This is the reason why the thresholding filter and the selection of a reasonable value for p_{min} will be described in the next sections in order to obtain the best correspondence between algorithm prediction and targeted masks. Once the U-Net is trained and p_{min} selected, the BOSS algorithm will be compared with traditional image segmentation ones, both described in the next section in terms of segmentation performance.

Traditional Image Processing Algorithms Description

Several traditional algorithms for image segmentation were selected for comparison with the AI-based proposed algorithm. In the following, five algorithms are considered. They are based on Niblack's method, Otsu's method, the weighted iterative threshold approach (WIT) [70] and two local threshold (LT) approaches.

Niblack's Algorithm

Niblack's method is based on the computation of a threshold value T which is a function of the energy mean value and standard deviation computed over a rectangular window. Through its sliding over the whole image, it is then possible to identify which are the foreground and background pixels by comparing the window's pixels energy values with the local threshold.

$$T(x,y) = m(x,y) + k \times \sigma(x,y) \tag{4.3}$$

where k is a parameter which varies between -0.2 and -0.1, while the other parameter is the dimension of the window (supposed squared in this test with side d_{Nb}).

Otsu's Method

It is based on the gray level image's histogram. It aims to find the threshold value which minimizes the intraclass variance [94] of the thresholded black and white pixels. The main advantage of this method is its unparametric nature which avoids the need for configuring it according to the image noise level.

Weighted Iterative Threshold Approach

This method is an iterative way to compute the optimal threshold for a given image. At each step, the threshold is computed using the following formula:

$$T = \frac{(1+\delta) \times \mu_1 + (1-\delta) \times \mu_0}{2} \tag{4.4}$$

where μ_1 and μ_0 are, respectively, the average gray level values of foreground and background pixels. With this new threshold, new sets of foreground and background pixels can be computed with the following mean values, and the process repeats with the computation of a new threshold. When the difference between two successive thresholds is lower than a certain tolerance Δ, the process stops. This algorithm is dependent on a scalar parameter δ which can vary in the range $[-1.0, +1.0]$.

Local Threshold Approach Based on Rectangular Areas (LTA)

The algorithm used is from MATLAB. It scans the whole image and computes a local threshold. The used size of the window is given by the following formula:

$$Size = 2 \times floor\left(\frac{Image\,size}{16}\right) + 1 \tag{4.5}$$

Over this window, a Bradley's mean [95] is computed. By comparison between a pixel's energy value and this local mean, the classification of the pixel as background or foreground occurs. This process is carried out using a scalar parameter called sensitivity (S). It varies in the range [0,1] and indicates sensitivity towards thresholding more pixels as foreground.

Local Threshold Approach Based on a Moving Streak Average (LTS)

The last one, the segmentation algorithm [7], can be configured using a static or a dynamic approach [96] for the background noise estimation [94]. In this work, a dynamic approach based on a zigzag local thresholding with moving average is used [97]. In particular, a pixel is saved if its energy value is greater than the background noise, evaluated through a moving average line-by-line, plus a threshold τ_{pre}.

Comparison Indices

The performance comparison between the BOSS algorithm and other algorithms will be detailed in the results section. The compared algorithms will be tested with a set of gray scale images and relative masks. In order to assess the accuracy of the algorithm's predictions against test masks, the use of suitable indices is necessary. These indices are heavily used in the segmentation field to compare two image masks as the U-Net output and ground truth from the dataset actually are. These indices can also be used to compare predictions of any couple of segmentation algorithms. Before introducing them, the notion of true positive (TP), false positive (FP) and false negative (FN) must be given:

- TP: A counter that increases by one unit every time both the predicted pixel and the reference pixel belong to the foreground set;
- FP: A counter that increases by one unit every time the predicted pixel belongs to the foreground set while the corresponding mask pixel is a background pixel;
- FN: A counter that increases by one unit every time the predicted pixel belongs to the background set while the corresponding mask pixel is a foreground pixel.

A good prediction has background pixels and foreground ones almost in the same position inside an array as the associated ground truth mask. The goodness of the prediction can be assessed through the evaluation of Precision, Recall and F1 index.

$$Precision = \frac{TP}{TP + FP} \times 100 \tag{4.6}$$

Precision represents the percentage of the TP with respect to the sum of TP and FP . The lower the FP is, the higher the Precision is (with maximum value equal to 100% under ideal conditions).

$$Recall = \frac{TP}{TP + FN} \times 100 \tag{4.7}$$

Recall represents the percentage of the TP with respect to the sum of TP and FN . The lower FN is, the higher the Recall is (with maximum value equal to 100% under ideal conditions). Recall and Precision are similar but with a slight difference. This is due to the distinction of FP and FN. Every time an FP or FN occurs, there is a discrepancy between the prediction and the reference mask, but the difference between FP and FN helps us better understand the behaviour of the trained network.

Another index to be defined is F1:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.8}$$

F1 index combines the Precision and Recall and lets us understand when the best compromise between FP and FN occurs. It reaches its maximum value when the discrepancy between the prediction and the mask is minimized. Ideally, the maximum value of F1 is 100%, but the best prediction will be the one with the highest value of the F1 index.

4.1.3 Results

In this section, results of U-Net's training, validation and testing are shown, together with a reasonable choice for p_{min} and number of initial filters. Then, a comparison test of the BOSS algorithm with traditional ones is conducted against the same set of images.

U-Net Training and Test

The U-Net tested here is configured with the values shown in Table 4.1.

Table 4.1. U-Net configuration parameters.

Parameter	**Value**
Image size	512
Learning Rate	10^{-4}
Regularization Factor	10^{-5}
Dropout Rate	0.25
Kernel Size	3
Kernel initializer	$'he\_normal'$

Training and validation were performed in Tensorflow Keras [28] considering three epochs and a dimension of the training batch equal to three. This has been conducted for four different numbers of initial filters: 16, 32, 64 and 128. This choice was made to see how performance varies in terms of accuracy and loss as the network complexity increases, in order to select the minimum required number of filters while achieving satisfactory performance. Every trained network reached accuracies higher than 99.80% after the first epoch and remained constant for training, validation and testing. The same behavior applies for the final losses which are not higher than 0.016. Moreover, no overfitting phenomenon occurred.

The training, validation and tests were performed on the following workstation:

- CPU: AMD Ryzen Threadripper PRO 3975WX 32 Cores 3.50 GHz
- RAM: 128 GB
- GPU: Nvidia Quadro RTX 5000.

Tuning of Thresholding Filter

From the previous phase, it seems that every model has learned its task, but their output is not yet a binary mask. To achieve this, a thresholding operation has to be performed to select a suitable value of p_{min}. Its tuning is conducted in a range of values from 0.01 to 0.9 and different models over the 300 images test set.

For every p_{min} and model, the values (over the whole test set) of Precision, Recall and F1 are computed and reported in Figure 4.4. It can be seen (Figure 4.5) that the network with 128 filters performs better in terms of output mask. The value around 69% for the F1 index means a minimized number of discrepancies between the BOSS algorithm's prediction and the mask. This value is the highest one if the best achieved F1 values are considered among all the networks. Because of this, a model with 128 filters and $p_{min} = 0.04$ was chosen for the BOSS algorithm design. A test with a 256 initial filters model was conducted, but performances did not show relevant improvements with respect to the 128 initial filters model, while the required storage memory and computational time increment was not negligible (and it is also undesired).

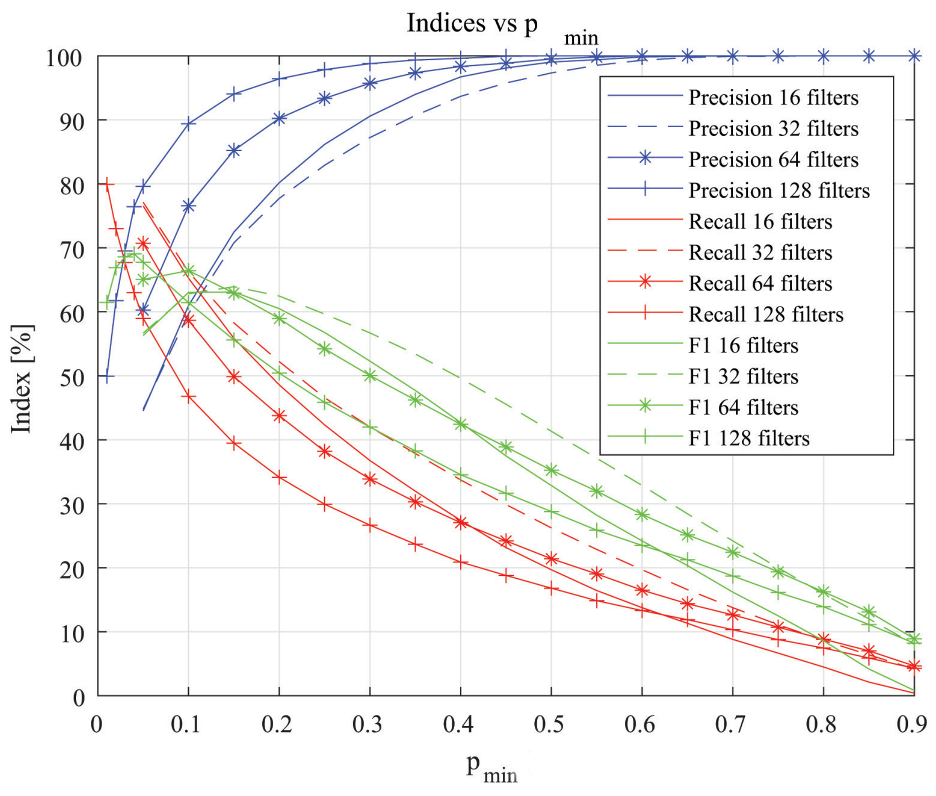


Figure 4.4. Performance curves vs. p_{min} and models.

Precision behaviour for low p_{min} values is due to an increasing number of FPs because as the p_{min} decreases, the number of FPs becomes greater if compared to TPs. For higher p_{min}, the number of FPs tends to 0, while TPs tend to a finite value, and the 100% Precision is achieved. This maximum value of Precision still does not mean that the prediction is good.

A similar discussion can be conducted for the Recall: as the p_{min} decreases, the risk of predicting FNs decreases with respect to TPs. This is the reason why Recall grows. For higher p_{min}, the increasing number of FNs causes the Recall value to fall towards 0%.

BOSS Algorithm Tests

In this section, examples of BOSS algorithm applications for stray light removal and ghost noise removal in night sky images are shown. The aim of a good segmentation

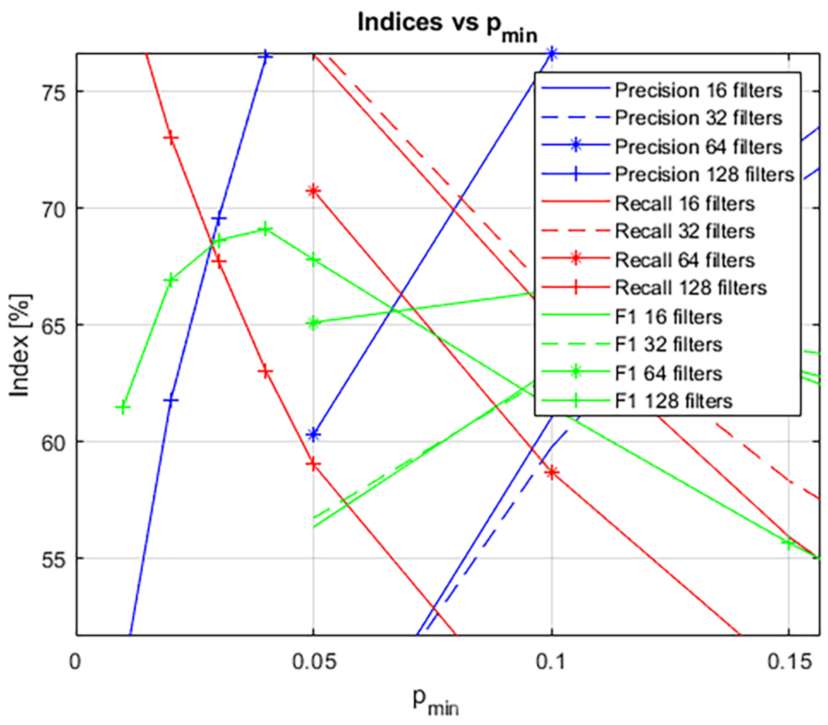


Figure 4.5. Details of performance curves vs. p_{min} and models.

in this case is to retrieve the point-like stars and streaks (RSOs) while removing background pixels that are often badly affected by several noise sources. By noise, we mean that part of the image signal that does not represent the target scene and must be filtered in a certain way. It corrupts the image actors, and if the SNR is higher than but close enough to 1, the useful signal could be filtered out as noise with the results of a loss of useful reference for further processing. The image noise is due to several causes briefly divided into two classes:

- Inner noise: It is due to the sensor components, electronics and realization technique.
- External noise: It is due to planets, the Sun, the Moon, the Earth, hosting platform structures and camera baffle together with lenses non-uniform reflection.

Sidereal Pointing Camera and Different Exposure Times

This test aims to investigate the BOSS algorithm behaviour against an increasing stray light noise source. Stray light noise is due to external light sources which cause a flare all over the image. It can be experienced when an optical-space-based sensor has a bore-sight direction close to the Sun, the Moon, or another high luminosity source. This stray light can have a uniform intensity or uniform gradient. If it is too strong, it can hide the useful signal associated to stars and RSOs and make them difficult or even impossible to be segmented and detected. A change in the stray light intensity during the mission may cause the need for continuous sensor calibration. This could be avoided with a calibration-less algorithm such as the U-Net-based proposal in this work. This noise can increase with the increase of the sensor exposure time as the amount of collected photons proportionally increases with time interval. In this test, images acquired with a Night Sky Simulator Facility (NSSF) (Figure 4.6) were used. This facility is made of a darkroom where a ZWO ASI 120MM-S camera points towards a screen. Here, Stellarium software is used to simulate the sky as seen by a specific camera with a selected FOV and optics (2.8 mm of focal length, f/1.4). All the simulated stars have a magnitude lower than 6.5 as shown in Figures 4.7 and 4.8. Using this software and a real camera, it is possible to simulate the working of a camera pointing at the night sky both for stars and RSO segmentation and detection for preliminary results. Images' noise sources are the real camera noises plus a further stray light source due to the screen technology (backlit screen).

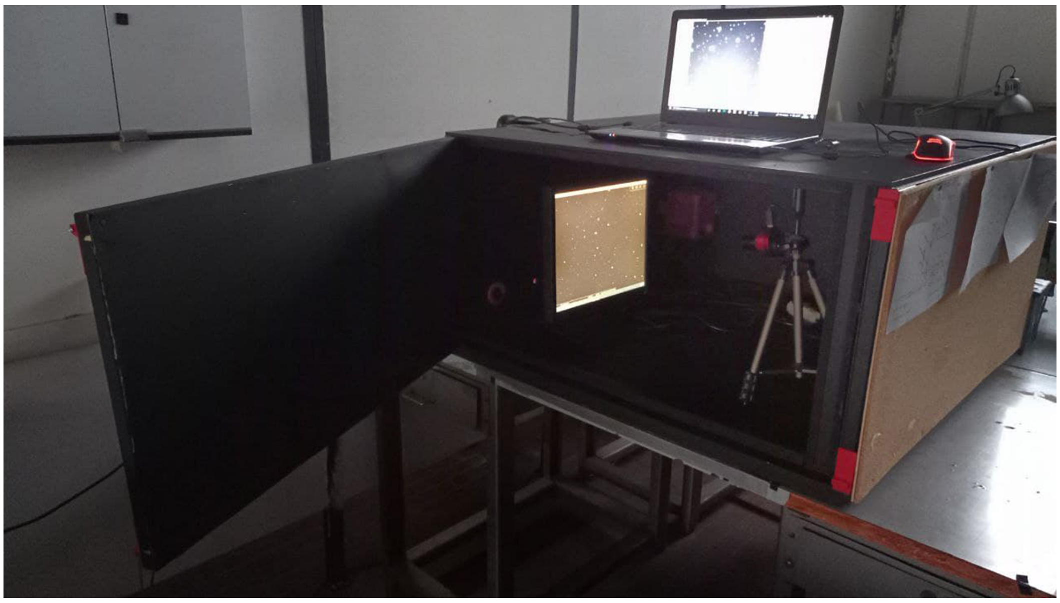


Figure 4.6. NSSF at ARCALab, School of Aerospace Engineering, Sapienza University of Rome.

A sidereal pointing was considered in this test just to see the effect of the image stray light noise increasing with the increase of the camera exposure time. Here 100, 200, 500 and 1000 ms of exposure were considered. The original images and segmented images are reported below (Figure 4.7). From this figure, it is possible to assess that the segmentation algorithm removes the stray light and correctly segments the stars. As the noise increases, the foreground pixels increase because of the increasing magnitude of the targets.

Ghost Noise Removal

This test aims to investigate the BOSS algorithm behaviour against a nonuniform gradient noise: ghost noise. This undesired noise source is due to camera lens reflections, spacecraft structure reflection and dust all over the camera lenses. The ghost noise is named in such a way due to its appearance: it is a vague and partially

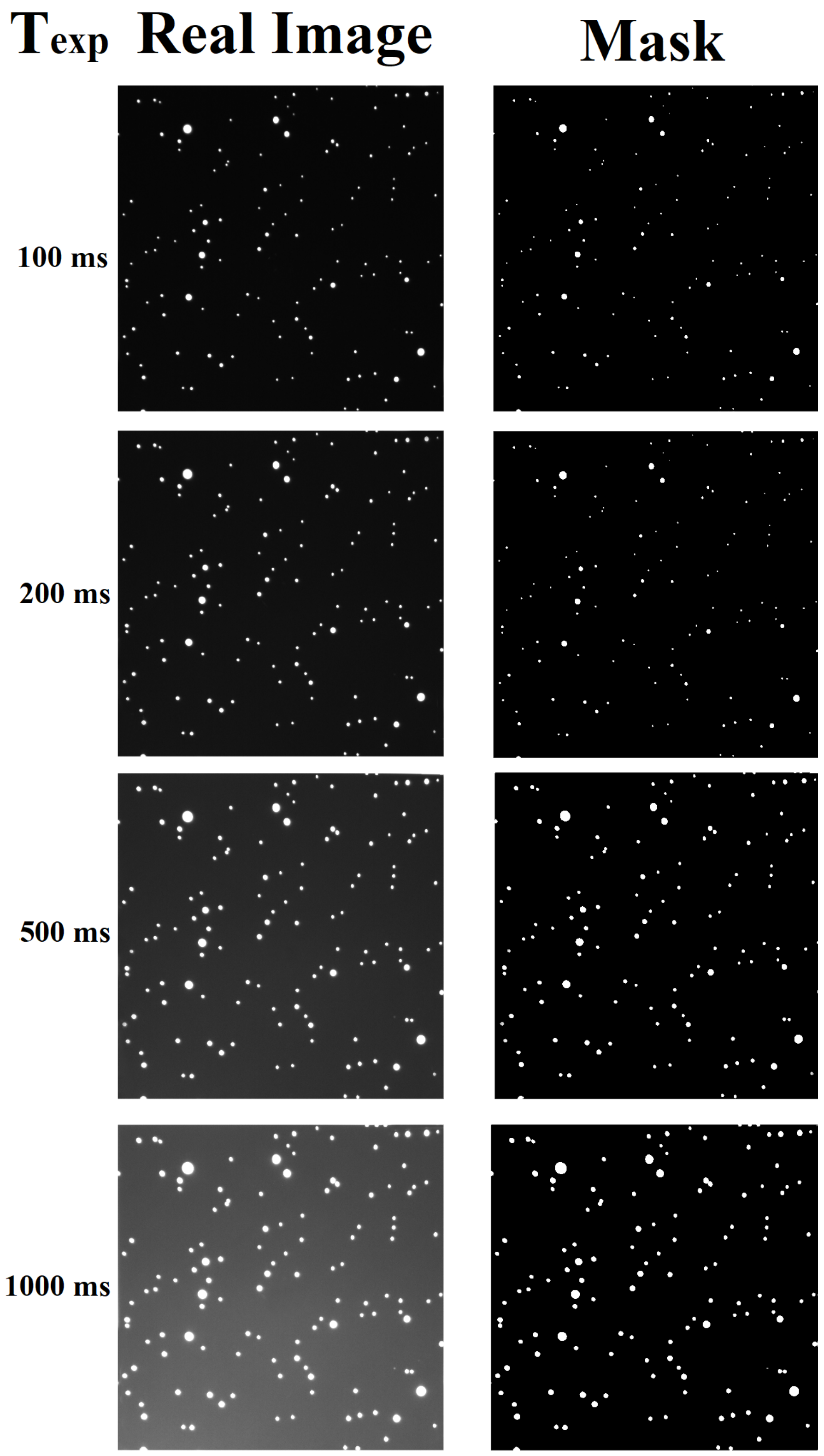


Figure 4.7. Exposure Time Effects Test: Images from NSSF (left), segmented images by the BOSS algorithm (right).

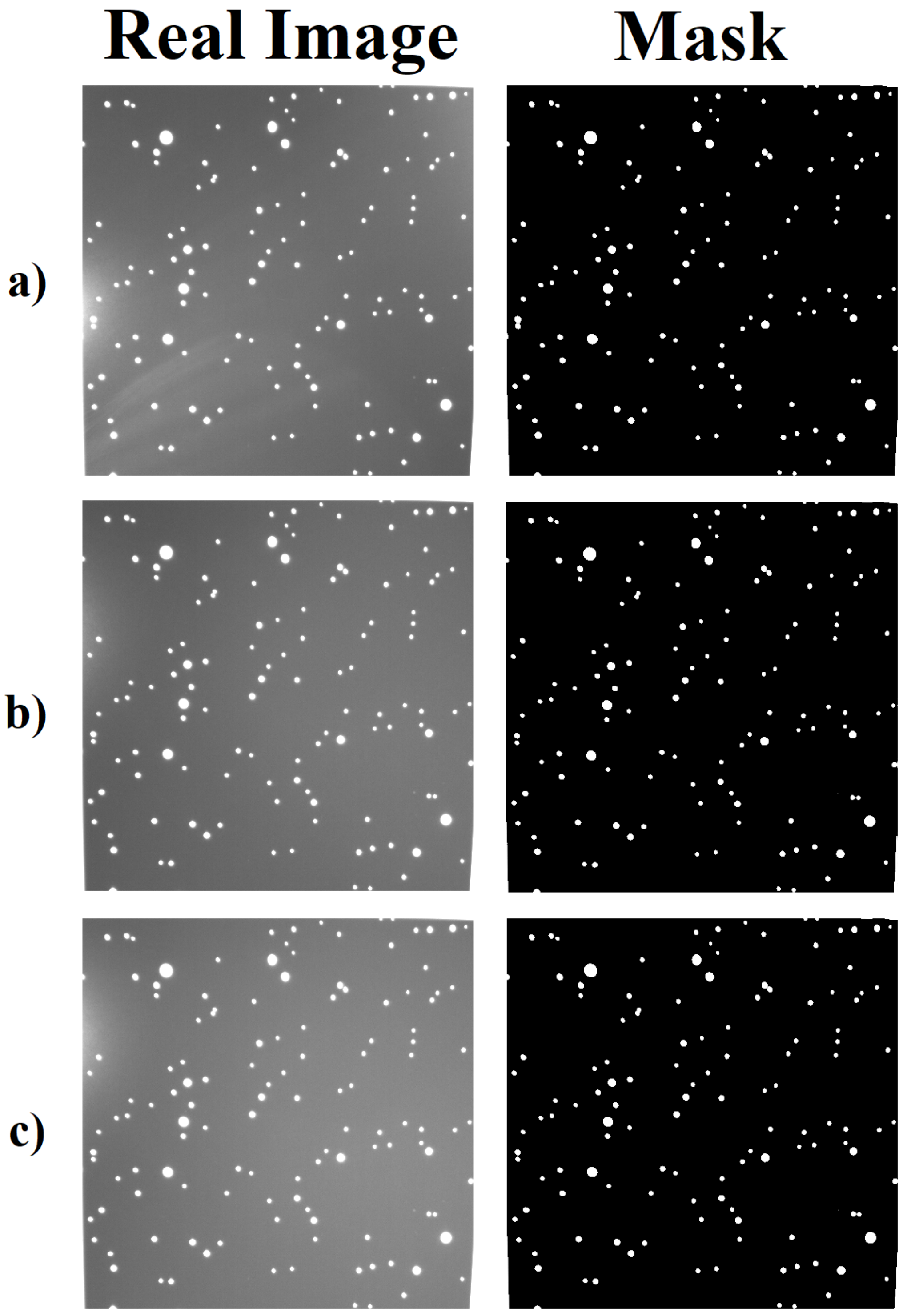


Figure 4.8. Ghost Noise Test: images from NSSF with ghost noise (left), segmented images by the BOSS algorithm (right); Three different ghost noise scenarios (**a**–**c**).

transparent shape superposed on: background, image's objects and a nonuniform gradient signal. This superposition may interest the whole image or part of it. If ghost noise is too strong, it can hide the useful signal associated with stars and RSOs and make them difficult or even impossible to be segmented and detected like a stray light. A change in the ghost noise intensity during the mission may cause the loss of information if no calibration is performed. This could be avoided with a calibration-less algorithm such as the U-Net-based proposal in this work. The results are shown with simulated images in Figure 4.8 and with real images in the next sections. This noise does not increase with the increase of the sensor exposure time, while the gradient level tends to be more uniform. Here, the NSSF has been used but with an added external flashlight source. The stray light and ghost noise were produced with three different positions of an external flashlight source which was moved at different heights on the left side of the camera and target screen inside the darkroom. Even in this case, it is possible to see a good segmentation quality provided by BOSS (Figure 4.8). A particular thing that can be noted is the white edges at the images' corners. They are not an algorithm error but a NSSF artifact due to a displayed undesired box in the Stellarium night sky images.

Real Image Test on BOSS Algorithm Output

In this section, a real night sky image is considered to test the BOSS algorithm against it. The aim of the test was the demonstration of the BOSS capability of providing a suitable segmentation and stars localization with a real image and real noise sources. The real image is shown in Figure 4.9 together with the segmented image. The camera used is a Nikon D3100 whose image has been cropped and resized to the U-Net input size of 512 pixels. The image contains, besides real sensor noises due to the electronics, a strong stray light source coming from the bottom and due to nearby city light pollution (Acquisition site: Tusculum/Frascati, Rome, Italy). Parameters of the camera are listed in Table 4.2. Clustering algorithms extracted the centroids of the eight brightest stars in the FOV and correctly localized them. They are shown in Figure 4.10 with stars names and visual magnitudes that have been reported in Table 4.3.

Table 4.2. Nikon D 3100 camera features. The image has been resized and binned to 512 px and 1:1 aspect ratio.

Feature	Value
Size	512 px
Aspect Ratio	1:1
focal length	30 mm
FOV	28.79°
pixel size	30.07 μm
Cut Off Magnitude	4.0
f-number	f/2.0

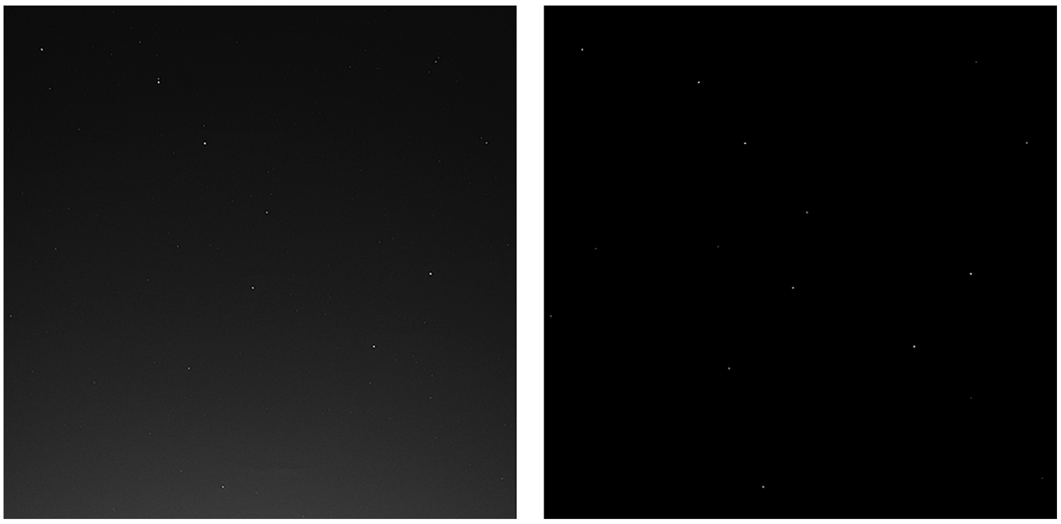


Figure 4.9. BOSS Ursa Major constellation segmentation: real Ursa Major image (left); BOSS binarized output image (right).

BOSS Algorithm Comparison

Here, a comparison between BOSS and traditional algorithms is conducted. The F1 index (Equation (4.8)) is chosen to compare the performance of the algorithms. Every algorithm was tested against the same test set of 300 simulated images [88]. Otsu's algorithm did not need to be configured, while LTA, WIT and LTS algorithms did. For these configurable algorithms, a tuning of their parameters was performed, and the F1 index was computed for every combination of their internal parameters.

Algorithms Comparison Procedure

The rationale behind the algorithm comparison is described in this section. The comparison dataset was composed of 300 images and 300 reference masks. Each reference mask represented the desired result of the segmentation process. Every reference mask was manually obtained using Adobe Photoshop 2019 (Image->Adjustments->Threshold), selecting a suitable threshold level because of the varying noise conditions over the 300-image test set.

Now, the purpose is to compare the output of the generic segmentation algorithm with the corresponding reference mask. The procedure was as follows:

- The output mask was compared with the reference mask pixel by pixel;
- The number of FP, FN and TP were updated during the mask comparison;
- FP, FN and TP values were used to compute the Precision, Recall and F1 values (these indices were described in the previous Section 4.1.2 *Comparison Indices*).

This process was repeated for all the 300 images, and a final averaged value for the F1 index was obtained for the considered algorithm.

This procedure was directly applied for the BOSS and Otsu algorithms because they do not need to be configured: Otsu does not need any configuration parameter,

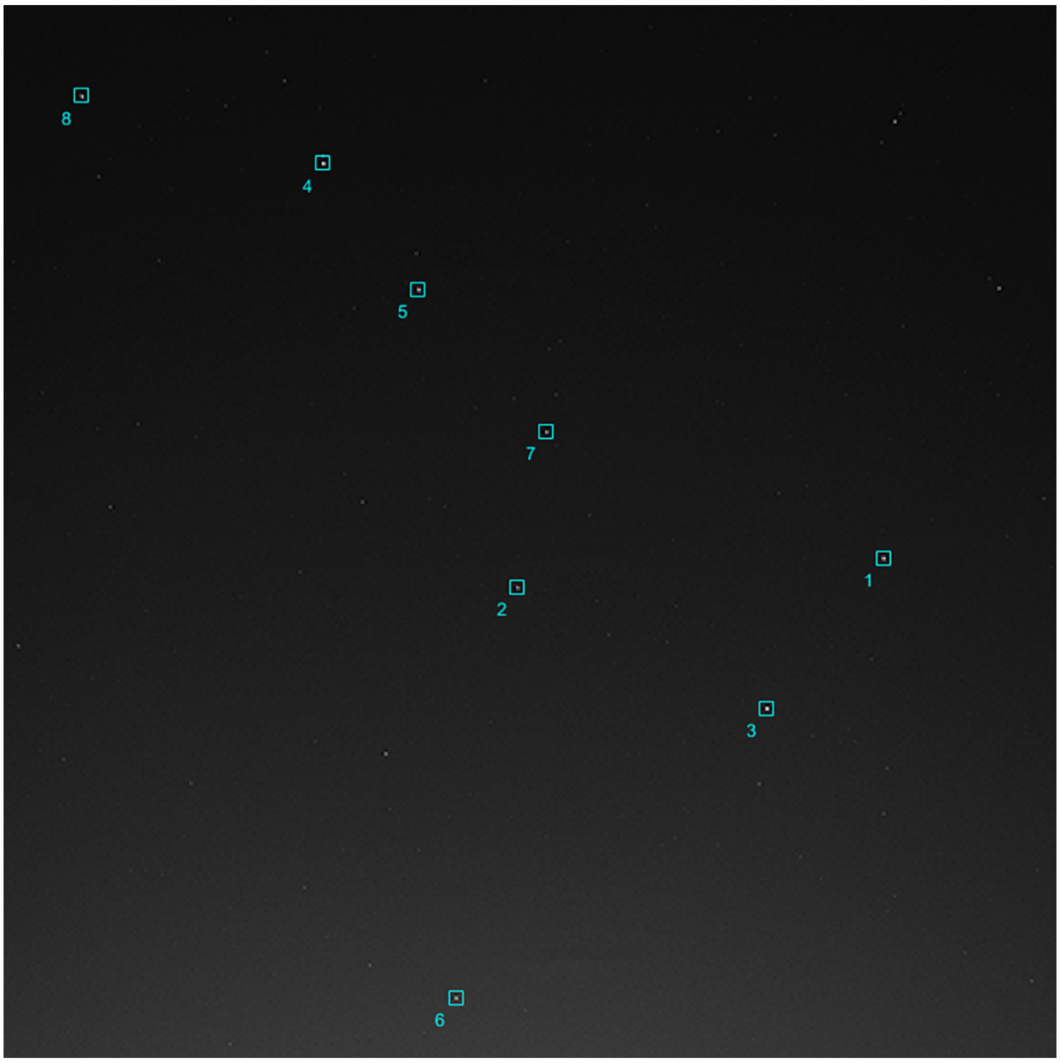


Figure 4.10. Ursa Major constellation stars localization performed by the BOSS algorithm.

and the BOSS algorithm has its fixed value of p_{min} which was frozen during its design in Section 4.1.3.

Niblack, LTA, WIT and LTS require an additional step before computing the final F1 value: the configuration parameters' optimization. Actually, all of them have at least one configuration parameter to be selected with a suitable criterion:

- Niblack's configuration parameters are k and d_{Nb};
- LTA's configuration parameter is the Sensitivity;
- WIT's configuration parameter is δ;
- LTS's configuration parameters are τ_{pre} and BKG_0.

By considering the generic configurable algorithm, the averaged F1 index was computed for every combination of the configuration parameters varying in their

Table 4.3. IDs, names and visual magnitudes of the segmented objects in Figure 4.10.

ID	Name	Magnitude
1	Dubhe	1.95
2	Phecda	2.42
3	Merak	2.34
4	Mizar	2.25
5	Alioth	1.75
6	Psi Ursae Majoris	3.16
7	Megrez	3.33
8	Alkaid	1.80

specific ranges. Actually, every selected value for the configuration parameter makes the algorithm to be more or less severe in terms of segmentation performance and changes the final F1 index value. Results of this process, using commonly used values for the parameters, have been collected for each configurable algorithm in Tables 4.4–4.7. In the end, the six averaged F1 values for Otsu, BOSS and the optimized algorithms can be obtained and were reported in Table 4.8.

Table 4.4. Niblack's algorithm tuning. F1 index vs. k and d_{Nb} . The first parameter varies along the rows from -0.2 to -0.1, while the second varies one along the columns from 1 to 10.

F1 (%)	d_{Nb}=1	d_{Nb}=3	d_{Nb}=5	d_{Nb}=7	d_{Nb}=10
$k = -0.2$	0.041	0.061	0.064	0.065	0.065
$k = -0.15$	0.041	0.062	0.065	0.066	0.067
$k = -0.1$	0.041	0.061	0.066	0.067	0.068

In these tables, the configuration parameters which maximize the F1 index were considered, and the maximum value of F1 is then reported in Table 4.8 for the final comparison. In this way, every configurable algorithm was optimized against the 300-image test set in order to make the comparison more challenging for the BOSS algorithm.

Comparison Results

Otsu's algorithms is the only traditional one which does not need any configuration of parameters. Its F1 score is 0.07 %.

The best achieved F1 values for every algorithm are summarized in Table 4.8 for a fast comparison.

Table 4.5. LTA algorithm tuning. F1 index vs. S . The Sensitivity varies along the columns from 0.1 to 1.0.

S	0.1	0.3	0.5	0.7	1.0
F1 (%)	79.63	68.97	3.62	0.04	0.04

Table 4.6. WIT algorithm tuning. F1 index vs. δ . Here, δ varies along the columns from -1.0 to $+1.0$.

δ	-1.0	-0.7	-0.5	-0.3	-0.1	0	$+0.1$	$+0.3$	$+0.5$	$+0.7$	$+1.0$
F1 (%)	0.07	1.19	3.35	19.35	37.43	50.86	27.17	16.96	12.08	7.58	3.92

Table 4.7. LTS algorithm tuning. F1 index vs. τ_{pre} and BKG_0 . The first threshold varies along the rows from 5 to 55, while the second one varies along the columns from 1000 to 2000.

F1 (%)	$BKG_0 = 1000$	$BKG_0 = 1500$	$BKG_0 = 2000$
τ_{pre}=5	13.39	13.39	13.39
τ_{pre}=15	72.75	72.75	72.75
τ_{pre}=25	60.85	60.85	60.85
τ_{pre}=35	46.55	46.55	46.55
τ_{pre}=45	35.15	35.15	35.15
τ_{pre}=55	26.92	26.92	26.92

Table 4.8. Summary of algorithms' best F1 scores.

Algorithm	Niblack	LTA	WIT	LTS	Otsu	BOSS
F1 (%)	0.068	79.63	50.86	72.75	0.07	69.05

From previous tables, we can see that the best F1 score was achieved by the LTA algorithm followed by the LTS one and the BOSS algorithm. Niblack and Otsu behaved worse than the others, while the WIT achieved high but not satisfying performances.

Both LTA and LTS behave better on the test set if compared to the BOSS algorithm. As a first impression, it would seem that the use of BOSS algorithm does not bring any advantage. However, the performances achieved by LTA and LTS were obtained via a tuning of their parameters, while the BOSS algorithm was not tuned after its design. LTA and LTS have to be calibrated every time the noise level changes inside the selected scenario to achieve the best segmentation quality output, while the BOSS algorithm does not need this calibration because it has been trained to segment well with several SNR levels. This consideration means that the 69% of F1 index is a generalized performance value, while the 79% and 72% values for the traditional algorithms are optimized and not generalized.

This would mean that a ST based on the BOSS algorithm would not require any calibration or segmentation performance degradation during its lifetime in orbit. Even if the sensor's noise increases with the increasing of the lifetime, the U-Net would be able to adapt itself to several levels of SNR. A stray light or a higher radiative region would not affect the quality of the segmentation product very much. With this consideration, the strength and meaning of the BOSS performance can be more appreciated and understood.

In Figures 4.11–4.13, it is possible to visually compare the segmentation algorithms output quality of these LT approaches and the BOSS algorithm to understand the limit of a traditional parameterized segmentation algorithm. Three real images of the night sky both from Earth and space were considered with different kinds and levels of noises. A discussion follows for each of them.

Considering the processing of these three figures:

- Figure 4.11: In the real image, two geostationary satellites (green boxes) on the bottom part of the image are clearly visible with the surrounding star field. This image was obtained by the telescope of the Italian Space Agency Matera Observatory. The noise that mostly appears in this image is a nonuniform flare all over the image stronger in the bottom part and weaker in the top. It is due both to telescope lenses and nearby city light pollution. It is an example of ghost noise. The noise sources associated with the sensor electronics are negligible due to the sensor cooling system which was keeping the camera at −20 °C. There are foreground horizontal and short streaks on the top left and top right edges of the image produced by the LTA algorithm. Moreover, there is some noise that has not been filtered in the whole image, together with the brightest and weakest objects. The same problem is present in LTS' prediction with a greater percentage of noise. Here, the best output quality is provided by the BOSS algorithm which correctly returns the most salient star-like objects, together with precious RSOs information. All the algorithms return the two small streaks of the geostationary. The stars in the background have a magnitude lower than 12. The used optics is an Officina Stellare RiFast 400 Telescope with a focal length of 1520 mm and an f-number of f/3.8.
- Figure 4.12: The real image is a detail of a frame from the EBEX mission star

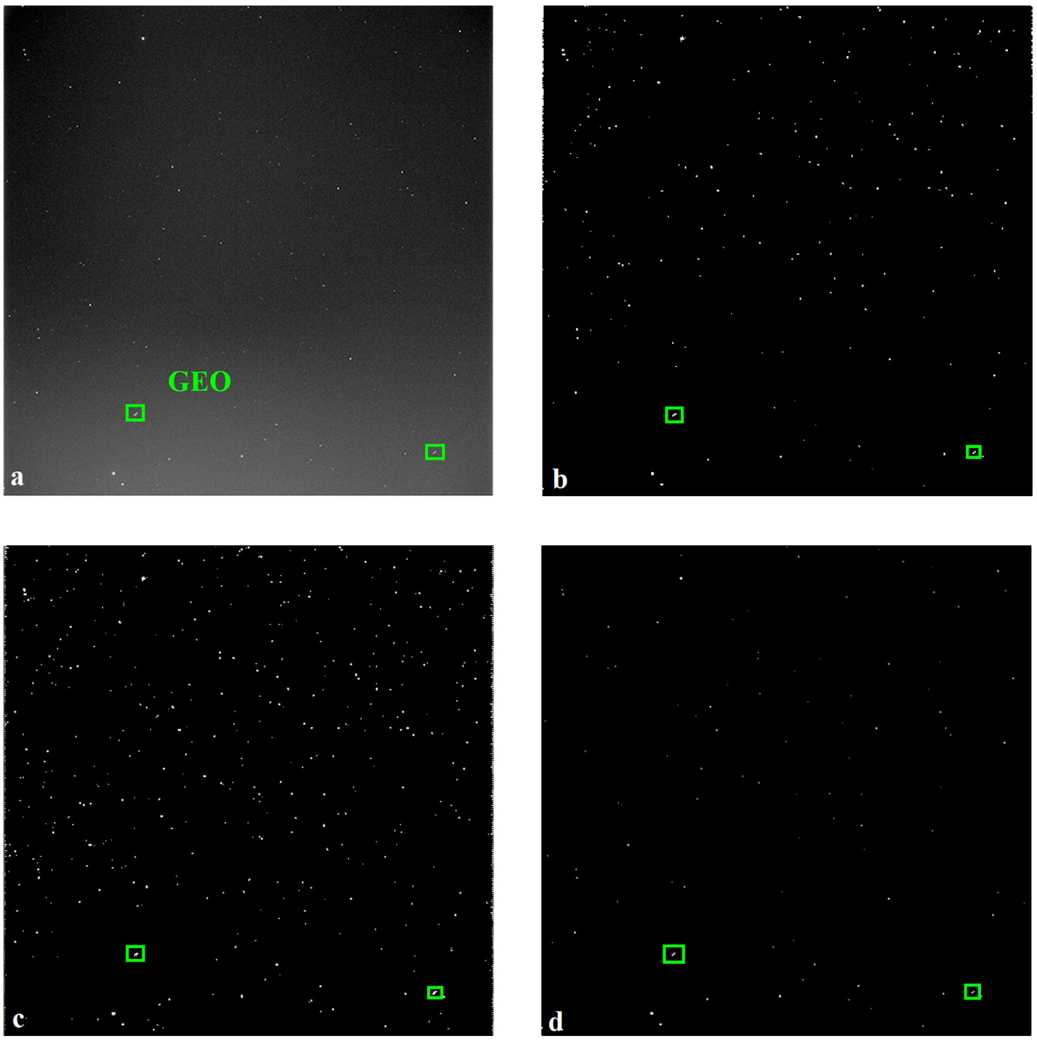


Figure 4.11. Italian Space Agency Matera Telescope's image segmentation: (**a**) real image; (**b**) LTA's prediction; (**c**) LTS' prediction; (**d**) the BOSS algorithm's prediction.

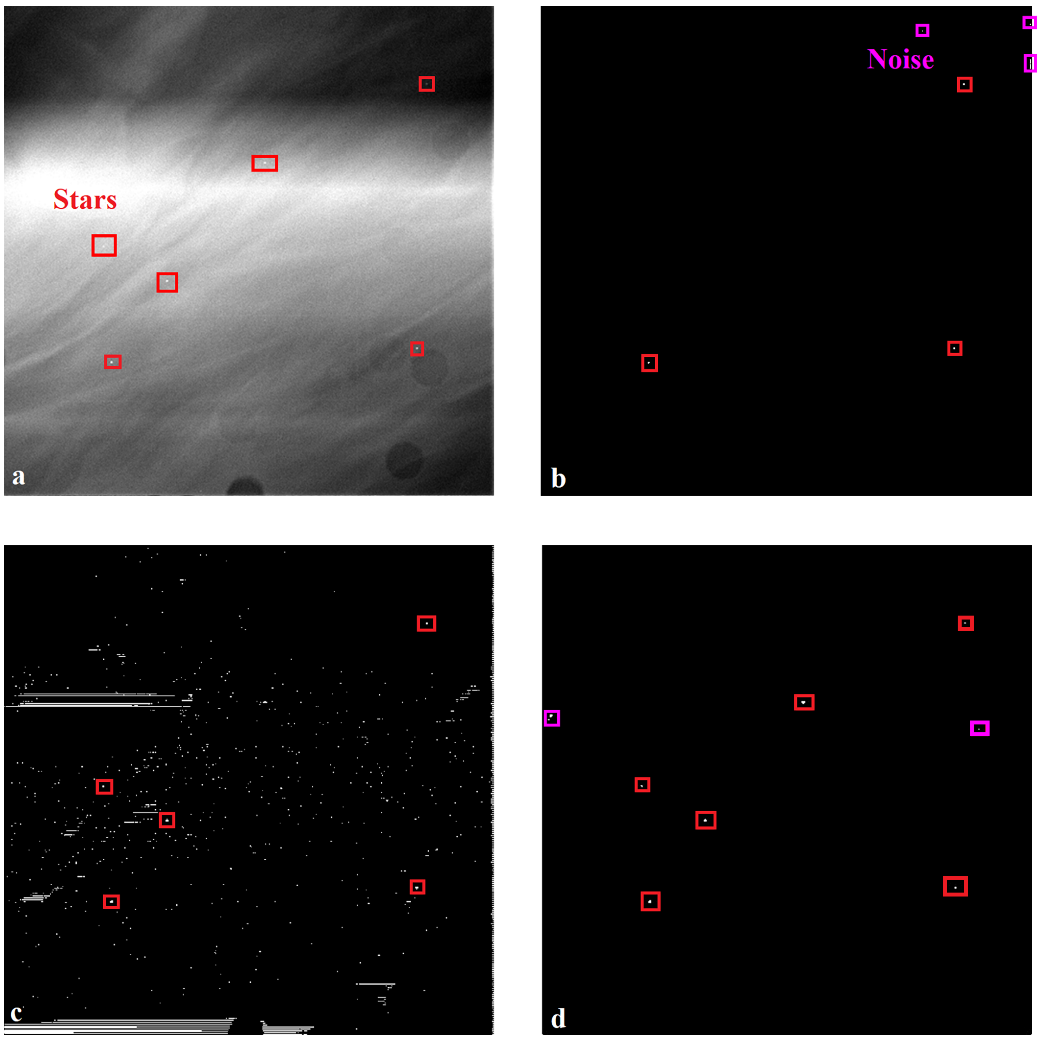


Figure 4.12. E and B EXperiment mission [98] star camera's image detail segmentation: (**a**) real image; (**b**) LTA's prediction; (**c**) LTS' prediction; (**d**) the BOSS algorithm's prediction.

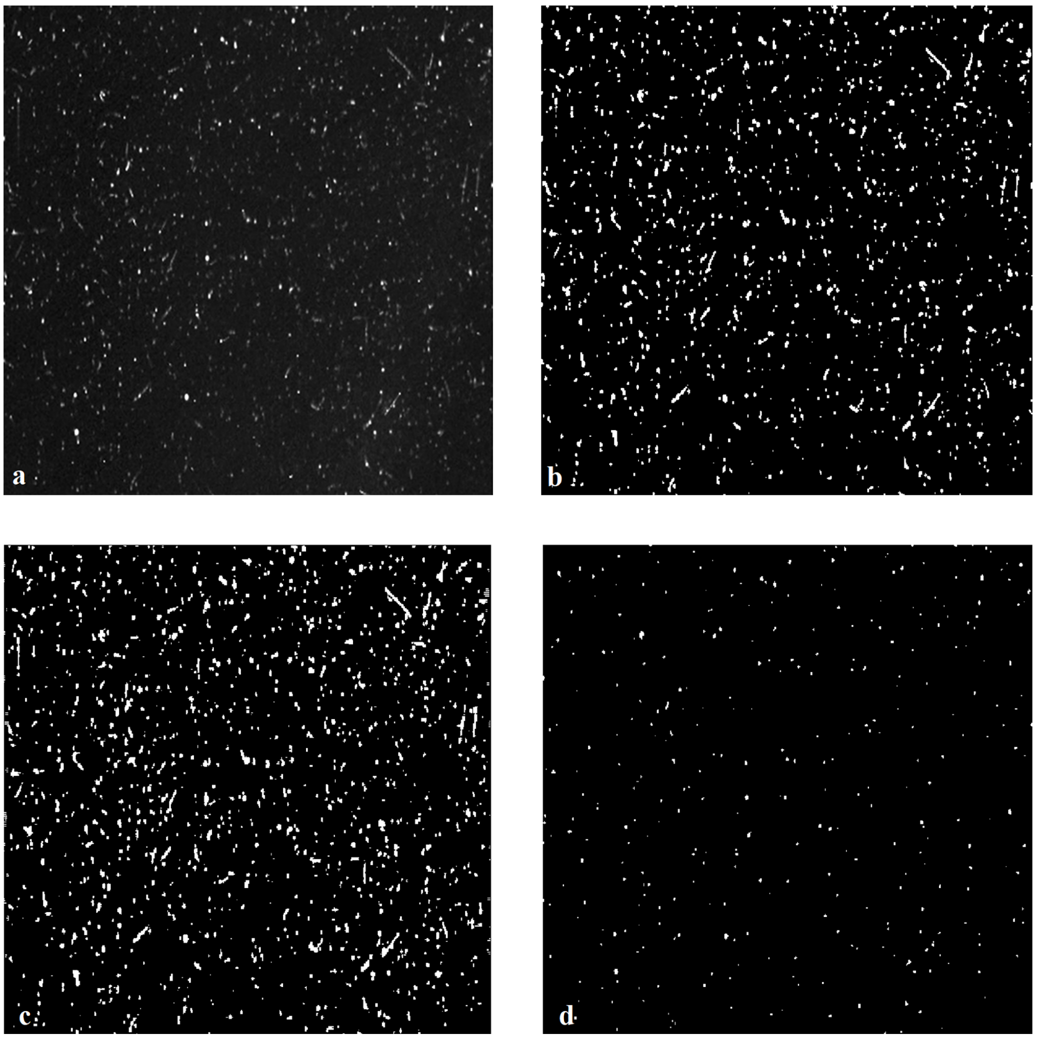


Figure 4.13. Juno Star Reference Unit's [99] image segmentation (Credits to NASA): (**a**) real image, (**b**) LTA's prediction; (**c**) LTS' prediction; (**d**) the BOSS algorithm's prediction.

camera [98]. The camera is mounted onto a stratospheric balloon carrying a telescope. It is pointed towards the above sky and contains a lot of noise. The dust over the sensor's lenses, mesosphere wind turbulence and stray light from the Antarctic continent's albedo are clearly visible and cause a strong ghost noise which covers the few stars in the background. Recognizing stars in this condition is important because star sensors are not used just in space where the absence of atmosphere avoids many noise sources, but also on ships or cruise missiles for navigation purposes. In this case being able to remove ghost noise due to clouds and other atmospheric effects is mandatory. The LTA (b) algorithm's prediction shows that few of the brightest stars have been correctly segmented with a not complete filtering action of the noise in the top right part of the image noise. The LTS behaves worst because the strong noise scenario does not allow it to correctly segment the image. In its predictions a lot of noise is still visible, with horizontal foreground streaks in addition to the correct segmented stars. This bad behavior is due to the algorithm's inability to adapt itself to different SNR conditions. Here, a re-calibration of the algorithm threshold would provide a better segmentation quality for the mask. In the last image (d), the BOSS algorithm prediction shows the segmentation of all the brightest stars (red boxes) with just a little portion of noise in the most corrupted region. Both the LTA and the BOSS algorithm show some problems with the strong ghost noise conditions but in different regions of the image and in different ways:

– BOSS algorithm seems to detect as stars the strong nonuniform brightest corrupted region due to the mesosphere winds in the image. The explanation for the BOSS algorithm is that the whitest spots in the ghost noise are recognized as possible embedded foreground objects;
– LTA seems to have an opposite problem with the darker region in the top right part: the gray level gradients in the darker localized region mislead the LTA algorithm during segmentation.

Among the two false positives cases, LTA behaviour could greatly affect the output mask quality because the gradient in the darker region of the image (generally the 99 % of these samples) increases the percentage of noise in the output mask (with possible negative effects both for attitude determination routines and RSO detection).

- Figure 4.13: This is a frame from a video [99] from Juno's Star Reference Unit (SRU) camera. The image contains stars and a huge number of SEU's crossing the sensor with different impact angles as the Juno spacecraft crosses Jupiter's high radiation polar regions. SEUs over images are caused by ionizing radiation which hits the sensor pixels. The more perpendicular to the sensor plane the ionizing radiation is, the more point-like the footprint of the high energy particle on the sensor is. This noise source can be reduced by shielding the electronics properly, but it cannot be removed. In the analysed image, many SEUs cross the sensor due to the high energetic region where the Juno spacecraft is. It is quite difficult to distinguish the white spots nature, but the purpose of this image processing is to show how the weakest elements on the detector (certainly SEUs under spacecraft inertial pointing condition) are removed. By a rapid inspection of the figures, it is possible to assess that LTA and LTS show similar segmentation outputs, and the SEUs which cross the detector almost tangentially are segmented as streaks. BOSS algorithm on the contrary, removes them. This is due to the dataset used to train the U-Net algorithm. The NN learned to detect just the brightest star-like objects, filtering less salient other ones. Here, the weak streaks of the SEU are removed, but this does not mean that all the star-like objects which appear in the BOSS prediction of Figure 4.13 are stars; they can be SEUs with a high impact angle.

These examples show that the BOSS algorithm has performances that are slightly lower but acceptable and generalized if compared to the calibrated traditional algorithms. Moreover, it has an intrinsic robustness to SEUs which a traditional algorithm does not normally have.

4.1.4 Summary of Findings for AI-based Night Sky Image Segmentation

In conclusion, this work has shown several aspects:

- The increment of initial filters in the U-Net increases the accuracy of the model predictions in terms of image segmentation quality and brightest objects detection;
- The BOSS algorithm is capable of achieving satisfying segmentation performance against different signal-to-noise scenarios. It does not need any re-calibration activity after its design in contrast to traditional LT segmentation algorithms;
- The comparison shows that the BOSS algorithm has segmentation performances which are comparable with respect to the optimized LT algorithms;
- The BOSS algorithm is able to properly remove the uniform stray light noise, most of the SEUs, weakest objects in an image and ghost noise. It is moreover able to provide a good product for star identification routines;
- A balanced CNN dataset for night sky images segmentation has been realized and publicly shared;
- The algorithm works fine, both with a simulated ST and with real images of the night sky (both from ground and in orbit platforms);
- Once trained with images from specific star sensors, the algorithm does not need to be calibrated again;
- The simple structure in the algorithm makes it simple to analyze, to implement and to validate;
- Early versions of this algorithm with a different training process proved to be suitable for RSOs detection when working with real images [100] (Section 4.2).

4.2 Objects Detection & Tracking

In the framework of SSA and Space Traffic Management (STM), the problems of RSOs detection and tracking to maintain and update ground catalogues, are of great interest. The planning of mega-constellations for space-based internet connectivity (SpaceX, OneWeb, Telesat and other companies) implicates the consequent significant increasing number of on-orbit active satellites and potential collisions with fragments across different space regions. This brings the *big sky* assumption[101] (i.e., collision risk is acceptably small with relatively few orbiting objects) to vanish because of the higher probability of close approaches and collisions. For these reasons, the importance of detection and tracking of RSOs has become a crucial aspect in the recent years. Nowadays the development and design of SBSS missions to increase RSOs monitoring coverage has led to the use of on-board optical sensors. Traditional filtering and image processing techniques are still being used[7] for RSOs detection. These techniques have the advantage of a long heritage due to their extensive application. On the contrary, their limitation is the need of complex filtering routines to perform easily human understandable tasks such as objects position extraction from images. For this reason the use of AI appears to be promising for space image data processing and space debris mitigation applications. In particular, the state-of-the-art shows great interest in AI applications to face space problems, to improve flexibility, autonomy and capability to handle failures and performance degradation while satisfying the reliability standards of a safety space system.

Recent works investigate the application of CNNs for RSOs detection: in [102] event based ground optical sensors are considered while in[103] point-like stars and streak-like stars scenarios are analysed with the consequent training of two different CNNs for object detection. A different approach based on semantic segmentation for dim small targets detection is exploited in [87], achieving satisfying performances at the price of a complex network's training and architecture. The advantages shown by CNNs in previous works could be combined with traditional hardcoded modules into a simple architecture for on-board RSOs detection applications through star sensors, adding also tracking capabilities to the detection task when RSOs position evolution occurs in a sequence of frames (Figure 4.14). Because of this, the development of a dedicated and reliable AI-based software to perform on-board RSOs detection and tracking is crucial for space based SSA activities because the collection of RSOs time evolution in a sequence of frames is essential for orbit determination (and thus SSA) purposes. A preliminary step in this direction has been taken in the present PhD activity. Actually, the previous image segmentation algorithm has been coupled with a further CNN to extract the RSOs position evolution within the on-board sensor FOV during a sequence of frames. The developed software aims to have as few modules as possible to reduce the algorithm size, complexity and hardware implementation difficulties in order to provide a reliable product. The RSOs Detection and Tracking module realization has been possible through several steps:

- Creation of a dataset of simulated ST images;
- Training, Validation and Test of the YOLO network based RSOs detection module;

- Design of the Tracking module to collect the history of each RSOs during their motion in the FOV in terms of centroids coordinates.

In the end, performed tests and obtained results from the developed algorithm are presented within this section. Real and simulated images have been selected and processed for validation and the performance in RSOs detection and tracking over multiple passages have been evaluated in terms of Precision, Recall and F1. To conclude, results discussion and a summary of findings for Detection and Tracking is reported.

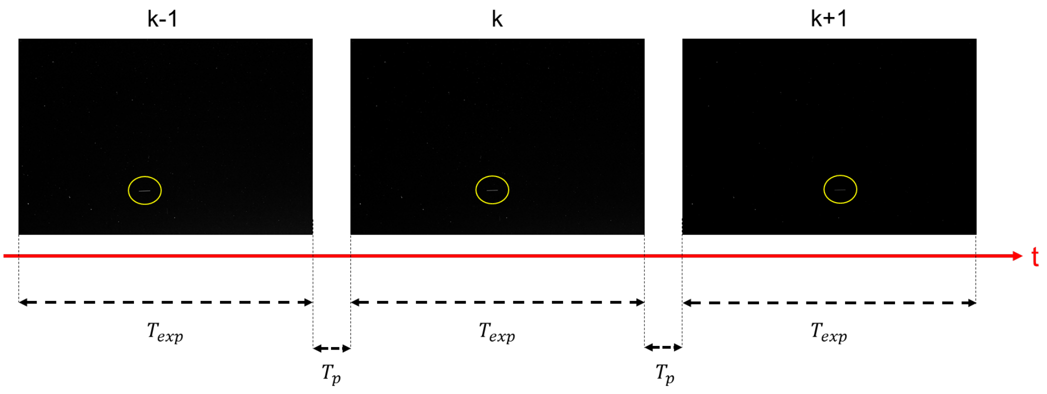


Figure 4.14. Images frame sequence and acquisition philosophy (Section 4.2.2).

4.2.1 Dataset Creation and Description

In the framework of designing and realizing an AI based segmentation, detection and tracking algorithm, a choice on the image size is made to work with a fixed sensor size. This is needed because the used U-Net works with fixed size inputs. Thus, the image size is chosen during the dataset creation phase. The new selected images size is 960 px $\times$ 640 px, to simulate an adequate ST sensor size. To train and to validate another 128 filters U-Net architecture, a set of 100 real images I took in a night sky acquisition campaign at Campo Imperatore, Gran Sasso, Italy has been used. They are not the same of the dataset in Section 4.1.1. These 100 images have been resized with the above mentioned size and then the U-Net training and validation process has been carried out. Then, each of the 100 images has been processed with the U-Net, and the corresponding output mask has been saved. This set of 100 output masks has been adopted as the baseline backgrounds for the creation of 5000 segmented images datasets, using a MATLAB streaks simulator which I contributed to develop. This 5000 images dataset has been used for the training and validation of the YOLOv3 NN in the *Detection Module*. This 5000 images dataset will be referred to as the *original dataset*. Each image consists of stars and streaks on a night sky segmented background and it is associated to a text file containing the information about the streaks' boxes in the YOLOv3 format. To guarantee a proper training of the YOLOv3, the dataset has to be as uniform and complete as possible in terms of streak length, orientation, position and number of streaks per image. The data-augmentation process involved the creation of 100 background images removing the objects from the original set. Then, one of these backgrounds is randomly chosen and a random number of streaks is printed on the image, with length, orientation and position randomly chosen from uniform distributions (between 0.1% and 10% of the image diagonal length and between -90° and 90° for length and orientation, respectively). The printing process starts by selecting a top-left coordinate and a length for the object from which a rectangular box is created. The top-left and bottom-right points of the box are then united with a one pixel-thick or two pixels-thick white line according to the following probability split:

- 60% two pixels / 40% one pixel if the streak length is inferior to the 1% of the image diagonal length;
- 20% two pixels / 80% one pixel if the streak length is between 1% and 3% of the image diagonal length;
- 5% two pixels / 95% one pixel if the streak length is greater than the 3% of the image diagonal length;

for a better modeling of the real case. Finally, the text file containing the information on the totality of objects in the image is compiled in YOLOv3 format. The box from which the information is collected is 0.9 times the size of the box used to draw the streak. The effects of the data-augmentation on the orientation, length and number per image of the streaks are shown in Figures 4.15, 4.16 and 4.17.

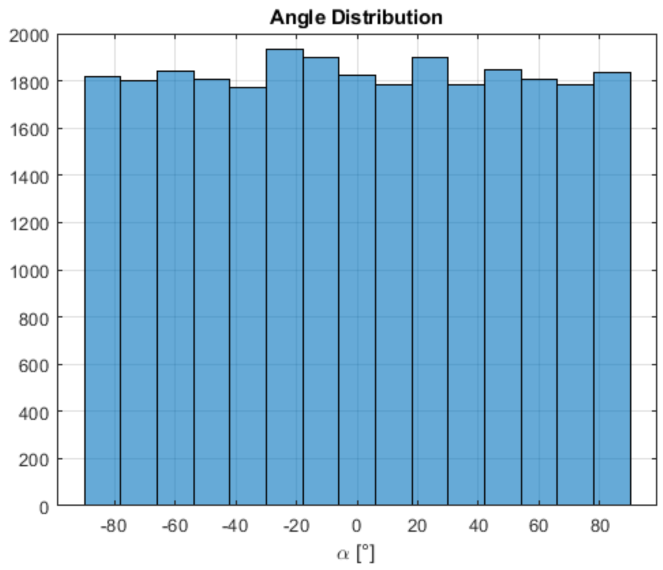


Figure 4.15. Streaks' angle distribution of the Dataset.

4.2.2 Algorithm Design and Configuration

The present RSOs Detection & Tracking is a combination of ML and classical programming techniques:

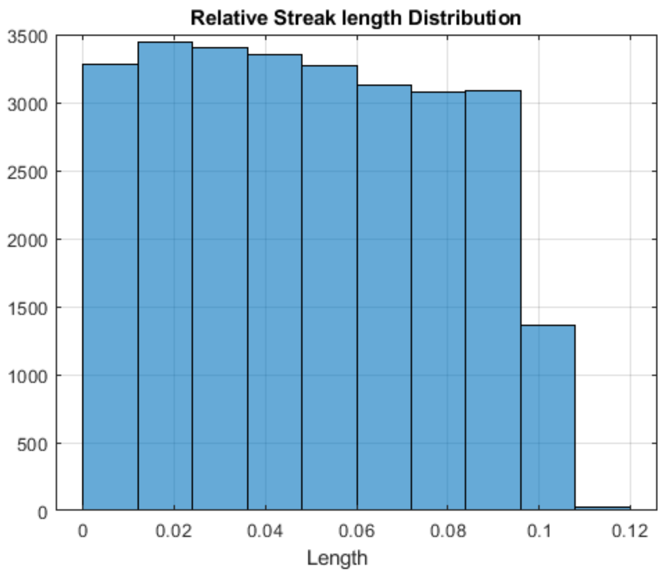


Figure 4.16. Streaks' relative length distribution of the Dataset.

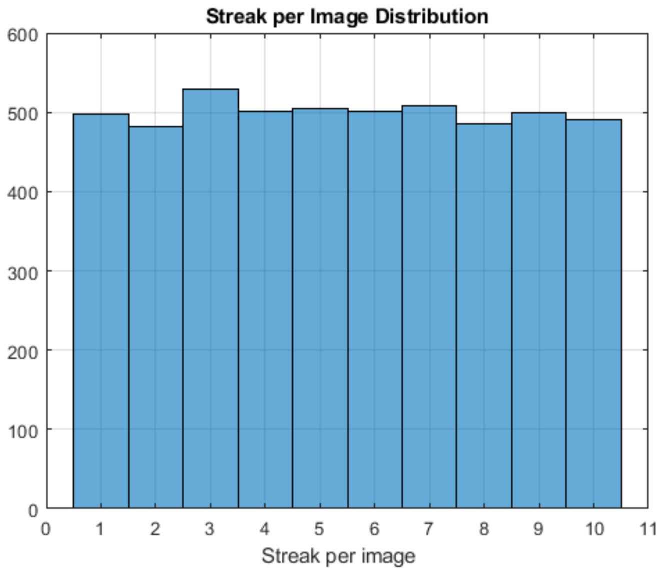


Figure 4.17. Streaks per image distribution of the Dataset.

- The Detection function is achieved by stacking U-Net[104] and YOLOv3[105, 51]. In this way the detection operation is helped by the image segmentation process performed by U-Net[106].
- The Tracking function is faced through a traditional hardcoded algorithm based on the minimization of a cost function.

The reason why object detection is performed on a mask rather than on the original raw image is due, now, to the possibility of training the U-Net in segmenting accurately the streaks (the new training strategy described in Section 4.2.1). Actually, the segmented streaks result bright enough inside the Mask to be easily detected by the YOLOv3. For the tracking task, a classical coding approach has been used in order to reduce the computational burden of the whole algorithm while achieving the desired RSOs tracking inside the FOV.

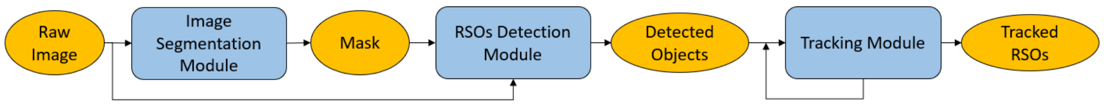


Figure 4.18. RSOs Detection and Tracking module algorithm structure.

The algorithm structure is shown in Figure 4.18 with the main blocks. The incoming raw image from the ST arrives to the *Image Segmentation Module* where it is binarized. The output of this module (*mask*) is a map of foreground and background pixels. The mask is used by the *RSOs Detection Module* to recognize the streaks in the night sky image and localize them into rectangular boxes. The convolution between the box-delimited regions in the *mask* and the same sky portions in the *raw image* provides rectangular areas in which every foreground pixels shows its original energy value. In this way, the computation of the streaks *centroids* and *extremal points* (see next subsections) is possible and, knowing the *exposure time*, the horizontal and vertical components of the streak, velocity can be derived. At this point, the computed information relative to the detected objects is used by the *Tracking Module* to distinguish the objects inside the FOV and provide their time evolution in terms of image coordinates as the output of the algorithm.

RSOs Detection Module

In the framework of this algorithm, RSOs are considered as streaks over a background made of point-like stars under the implicit assumption of low angular velocity of the orbiting platform. This assumption is reasonable since most of the missions involving STs require a sidereal pointing for other payloads constraints. In Figure 4.21 the RSOs Detection module is shown. The *mask* is processed by a YOLOv3 model. This classifies the streaks and localize them through rectangular boxes. For every detected streak, the box top left pixel coordinates ($\mathbf{P}_{box}$), width and height and class probability are known. This group of info is then used to extract the relative rectangular area both on the *mask* and on the *raw image*. Through the convolution of these two arrays, the *Clustering Module* computes the streak centroid with a weighted energy based average (Equation 4.9).

$$\mathbf{P}_{cluster,i} = \mathbf{P}_{box} + \sum_{j} \left(\frac{E_{px,j} \cdot \mathbf{p}_{px,j}}{E_{cluster,i}} \right) \tag{4.9}$$

with

$$E_{cluster,i} = \sum_{j} E_{px,j} \tag{4.10}$$

Where $\mathbf{P}_{cluster,i}$ are the *absolute* coordinates of the i^{th} cluster, $\mathbf{p}_{px,j}$ are the *local* coordinates of the j^{th} pixel in the box and $E_{cluster,i}$ is the sum of the energies $E_{px,j}$ of the i^{th} cluster's pixels.

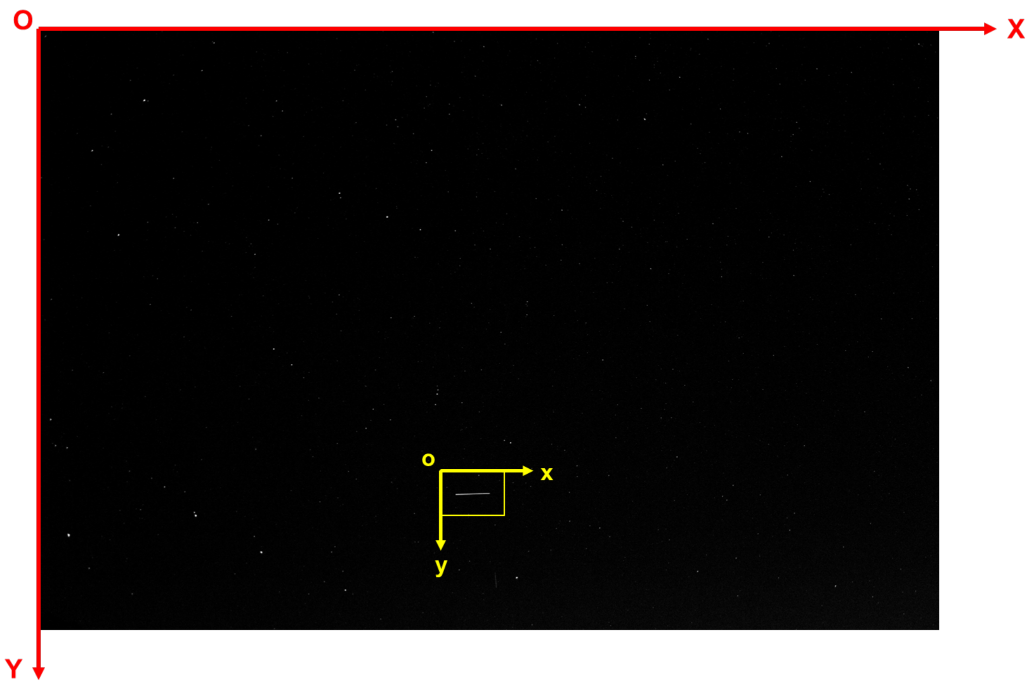


Figure 4.19. Absolute (red) and local (yellow) reference frame.

Absolute coordinates are relative to the *image reference frame* and *local* coordinates to the *box reference frame* (Figure 4.19). The *Clustering Module* computes also the *Extremal Points* of the streaks, namely the coordinates of the upper and lower tips of the objects, as the points in which the horizontal coordinate takes its minimum and maximum value. If the streak goes from the lower-left corner of the box to the upper-right corner the minimum value corresponds to the lower tip and the maximum to the upper tip, vice versa if the streak goes from the upper-left corner of the box to the lower-right one.

Since more than one foreground pixel may have the same value of horizontal coordinate (x), the vertical coordinate (y) of the extremal points is computed as the mean of the y values of all the points sharing the same horizontal coordinate (Figure 4.20). Considering the absolute value of the extremal points coordinate differences, it is possible to divide them by the image *Exposure Time* and to obtain an estimate

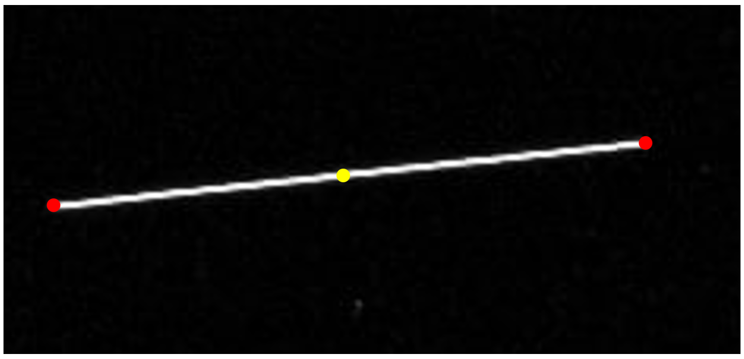


Figure 4.20. Extremal points (red) and cluster centroid (yellow).

of the RSO velocity in the image plane (both in terms of vertical and horizontal components). Then this velocity is used to predict the position of the centroid in the next image taking into account both the *exposure time* (T_{exp}) and the time between two subsequent camera acquisitions, hereafter referred to as *pause time* (T_p), see Figure 4.14.

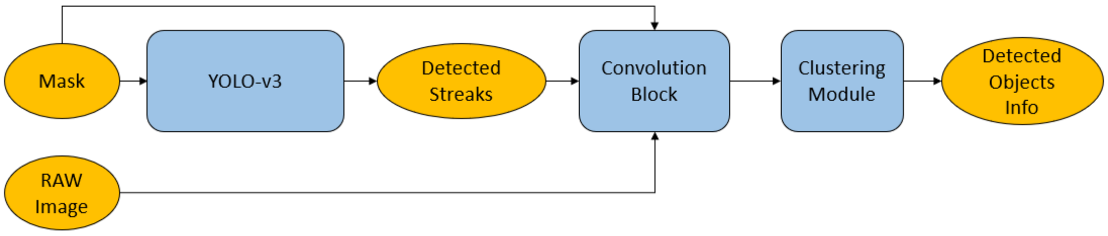


Figure 4.21. RSOs Detection Module algorithm structure.

RSOs Tracking Module

The *Tracking Module* algorithm is in charge of the association between detected objects in a sequence of images. The task is to interpret correctly the incoming information from the detection module by grouping the detected centroids under different indices which identify each RSOs crossing the FOV. With this operation it is possible to understand which RSOs every centroid belongs to. Before starting the module description a note on the nomenclature used is needed: the number of objects stored in the algorithm memory will be referred to as n_{obj} while the number of detected objects (boxes) in the current image will be indicated with n_{box}.

Generally, the algorithm will have n_{obj} stored in its memory and its task is to understand if the n_{box} information in each new image are associated to the stored objects or belong to a new one. The tracking module is based on the minimization of a *Cost Function* J which is computed for each box-object pair. From the RSOs detection module the generic box's centroid information has the following format:

$$box_i = [x_i \; y_i \; m_i \; \Delta x_i \; \Delta y_i \; n_{frame}] \tag{4.11}$$

where:

- x_i and y_i are the horizontal and vertical box's centroid coordinates in the image plane;
- m_i is the detected box's diagonal orientation (rough approximation of the streak's angle) w.r.t. the horizontal direction;
- Δx_i and Δy_i are the horizontal and vertical lengths of the box's streak which are computed through the streak's extremal points positions. Their ratio would be numerically equal to m_i if the streak's extremal points would occupy the opposite bounding box's corners (that is not generally true);
- n_{frame} is the number of frame where the box has been detected.

At each image the *Cost Function* takes into account the position and orientation of the box and compares them to the corresponding expected values determined from the previous image objects. The *Cost Function* for the i-th box with respect to the j-th stored object is given by:

$$\begin{aligned} J_{ij} = w_1 &\left| |(x_i{}^{new} - x_j{}^{latest})| - v_{xj}{}^{latest} \cdot \Delta T \cdot (n_{skip}+1) \right| + \\ + w_2 &\left| |(y_i{}^{new} - y_j{}^{latest})| - v_{yj}{}^{latest} \cdot \Delta T \cdot (n_{skip}+1) \right| + \\ + w_3 &\left| (m_i{}^{new} - m_j{}^{latest}) \right| \end{aligned} \tag{4.12}$$

where:

- v_{xj} and v_{yj} are the horizontal and vertical components of the velocity of the j-th object, respectively. They are obtained by dividing the Δx_j and Δy_j with the exposure time T_{exp} for the stored j-th object;
- ΔT is the time interval between two consecutive images, namely the sum of T_{exp} and the pause time T_p (Figure 4.14);
- n_{skip} is the number of frames where the object is not actually detected by the algorithm (miss or too weak object). It takes into account the fact that an object can be missed by the detection module for one or more images Figure 4.22;
- w_1, w_2 and w_3 are the weights of the three different contributions to the total *Cost Function.*

The superscript *new* refers to the information contained in the new incoming image, while the superscript *latest* contains the latest available information related to the j-th object.

Cases

The algorithm is divided according to four possible cases or scenarios.

1. $n_{box} = 0$ No box is detected, the algorithm will not update anything (even if at least one object is stored in memory);

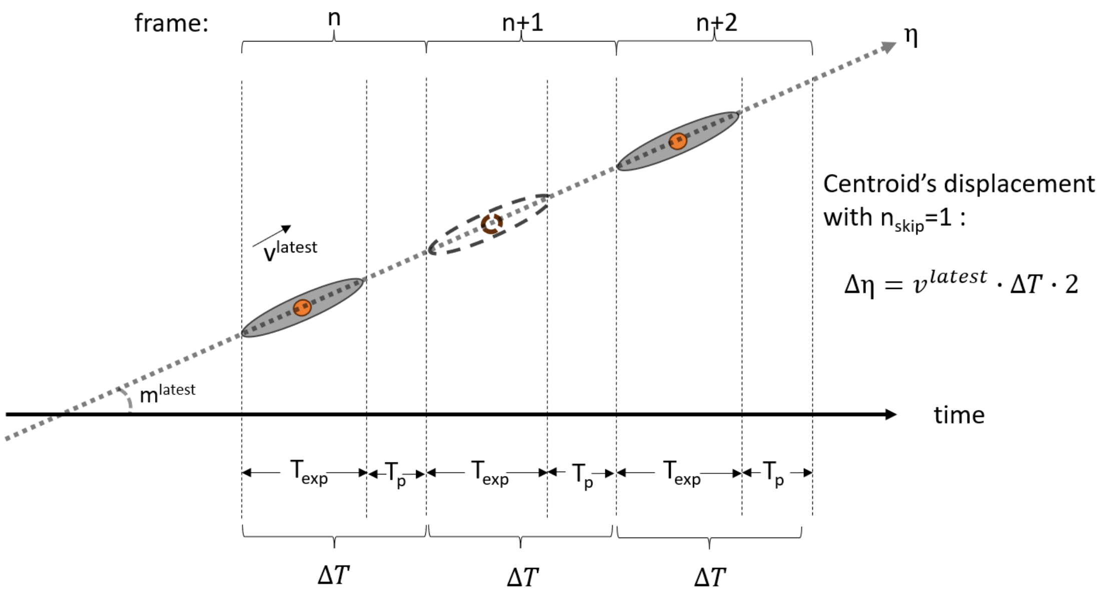

frame:
n
n+1
n+2
η
v^{latest}
m^{latest}
Centroid's displacement with n_{skip}=1 :
$\Delta\eta = v^{latest} \cdot \Delta T \cdot 2$
time
T_{exp}
T_p
ΔT

Figure 4.22. Missing object case with $n_{skip} = 1$. Centroids are the orange circles while the grey objects are the streaks.

2. $n_{box} > 0$ and $n_{box} < n_{obj}$ This case can be divided into two subcases:
   (a) The detected information n_{box} are all related to stored objects. The new information n_{box} is a own subset of n_{obj}. This case is faced through a simple minimization process;
   (b) Among the detected n_{box} there is at least a n^* which is not related to stored objects. This case is faced through the minimization of the *Cost Function J* but a *Threshold* J_{THR} and a condition are introduced.

To better understand the solution processes of simple minimization and minimization plus the condition involving J_{THR}, two examples are provided. In case 2.a all the detected boxes were previously stored in memory (Figure 4.23).

The association Table 4.9 in this case will be like this (hypothetical values have been used for sake of illustration):

Table 4.9. 2.a Case association Table

J	Obj_1	Obj_2	Obj_3	Obj_4
Box_1	4.5	2	1	3
Box_2	1.5	1	2	4
Box_3	4	3	2	1

Table 4.9 shows that Box_1 will be associated to Obj_3, Box_2 to Obj_2 and Box_3 to Obj_4. In this process the Obj_1 will not be associated with the detected boxes (correct behaviour of the algorithm).

In case 2.b among the boxes there is at least one that is not related with the algorithm knowledge (Figure 4.24).

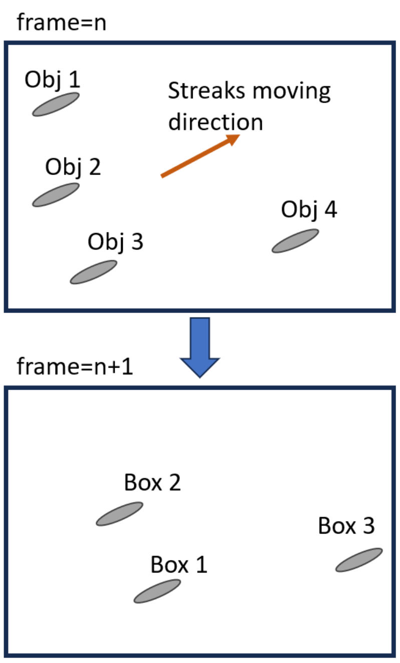


Figure 4.23. Case 2.a: detected boxes are a own subset of stored objects.

The association Table 4.10 in 2.b will be like this (hypothetical values have been used for sake of illustration):

Table 4.10. 2.b Case association Table

J	Obj_1	Obj_2	Obj_3
Box_1	3	1	3
Box_2	4.5	4	3

According to Table 4.10 the Box_1 would be correctly associated to Obj_2 while the Box_2 would be associated to Obj_3. Anyway, the association between Box_2 and Obj_3 is not correct because Box_2 has a different inclination and position if compared with the time evolution of Obj_3. This brought me to introduce a further step where each minimum $Box - Object$ value of J is then compared with a *Threshold* J_{THR}. This threshold can be defined as follows:

$$J_{THR} = w_1 \Delta\tilde{X} + w_2 \Delta\tilde{Y} + w_3 \Delta\tilde{M} \tag{4.13}$$

This is a static value of J_{THR} because through an a priori definition of $\Delta\tilde{X}$, $\Delta\tilde{Y}$ and $\Delta\tilde{M}$ values this threshold does not change. $\Delta\tilde{X}$, $\Delta\tilde{Y}$ and $\Delta\tilde{M}$ are user defined threshold that can be empirically defined.

Another different and dynamic approach to define J_{THR} is the following:

$$J_{THR} = k \cdot min(J_{ij}) \tag{4.14}$$

This approach is dynamic because the minimum value of J varies in every situation. The k safety-factor can be put equal to 2 to lower the risk of wrong associations.

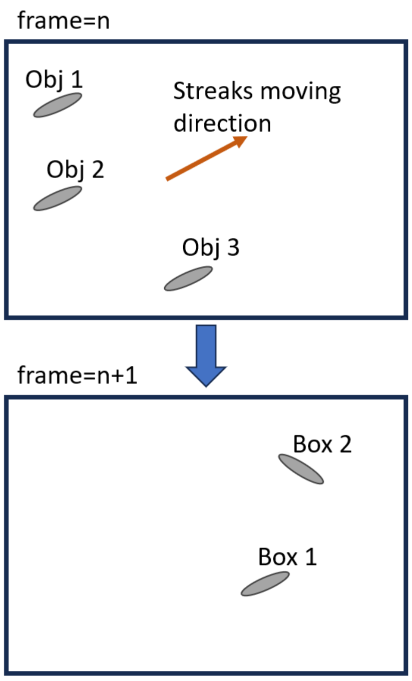


Figure 4.24. Case 2.b: detected boxes are not an own subset of stored objects.

Coming back to the 2.b case and Table 4.10 the $min(J_{ij}) = 1$. This causes a $J_{THR} = 2$ and the following comparison is done:

$$J_{Box2-Obj3} = 3 \geq J_{THR} = 2 \tag{4.15}$$

Because of 4.15, the Box_2 will not be associated to Obj_3 but an Obj_4 will be initialized in the algorithm memory with information coming from Box_2. This comparison check is done both in 2.a and 2.b case despite the threshold has been introduced in 2.b case description.

In the following shown tests a static approach for J_{THR} definition is adopted and the Cost Function-Threshold comparison does not differ from what has been described now.

3. $n_{box} > 0$ and $n_{box} = n_{obj}$

   Case 3 is divided into two subcases:

   (a) The detected boxes are all the objects stored in memory;

   (b) The detected boxes contain a never seen before object.

   Case 3.a is solved in the same manner of 2.a and 3.b is solved like the 2.b

4. $n_{box} > 0$ and $n_{box} > n_{obj}$ There is at least a detected box which belongs to a new object. Also this case presents the same solution approach of 2 and 3.

Figure 4.25 shows an example of a real passage of two objects in the night sky in which each image represents a different classification scenario.

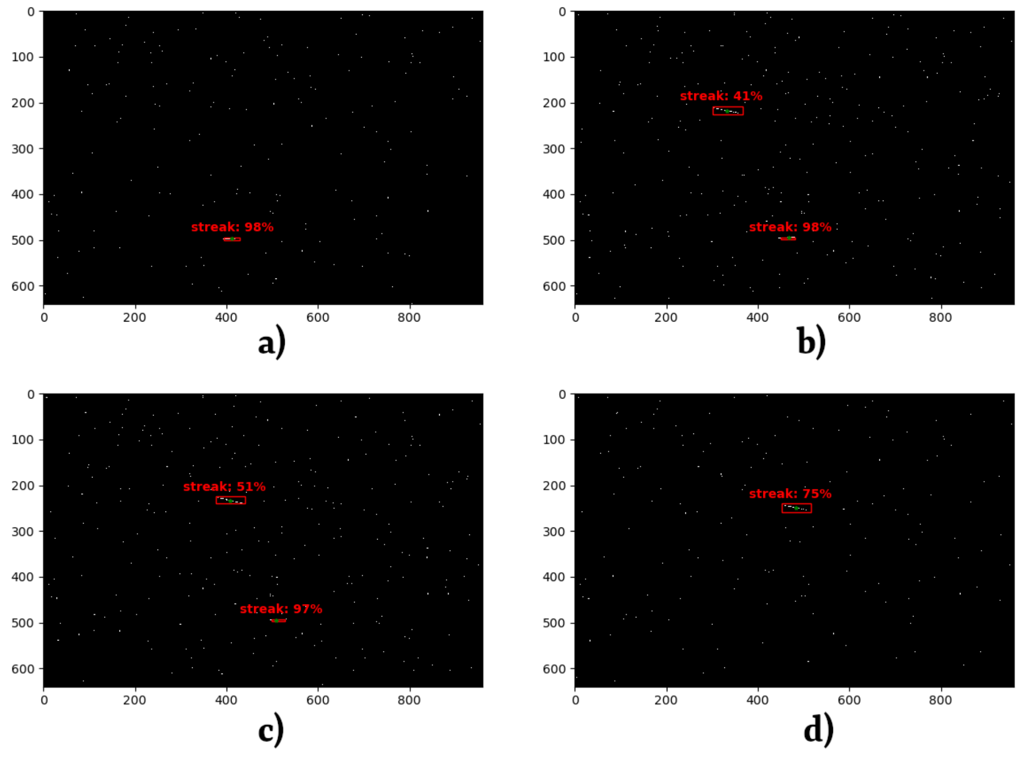


Figure 4.25. A real passage of two objects in the night sky. Cases evolution is (assuming initial absence of objects in memory): a) case 4; b) case 4; c) case 3; d) case 2.

Filters

After a sequence of three images without a detection of new boxes, the sequence is considered completed and several checks on the stored objects are performed. Objects (of the completed sequence) which have been detected in few frames (if compared to the number of images in the sequence) are removed from stored objects and considered again as potential candidates (possible noise, not correct association performed through the association table etc...). Objects with an *anomaly degree* higher than a certain threshold are removed from the stored objects and considered again as potential candidates (unreliable objects). Potential candidates will be further used by the *aggregation filter* to try a last possible association based on a least square method process. To understand the filters actions let's consider the scenario in Figure 4.26. Here, the blue rectangles represent the detection history while the orange ones represents anomalies presence. Anomalies do not affect all the objects in general and the tracked objects can be detected just in few frames.

These actions/filters are:

- Occurrences filter: It checks the number of objects in memory. For every object a check on the Occurrence Percentage (OP) is performed:

$$OP_{Obj_k} = \frac{n_{Obj_k}}{n_{total}} \cdot 100 \tag{4.16}$$

where n_{Obj_k} is the number of frames in which the object has been detected while the n_{total} is the total number of frames of the completed sequence. If

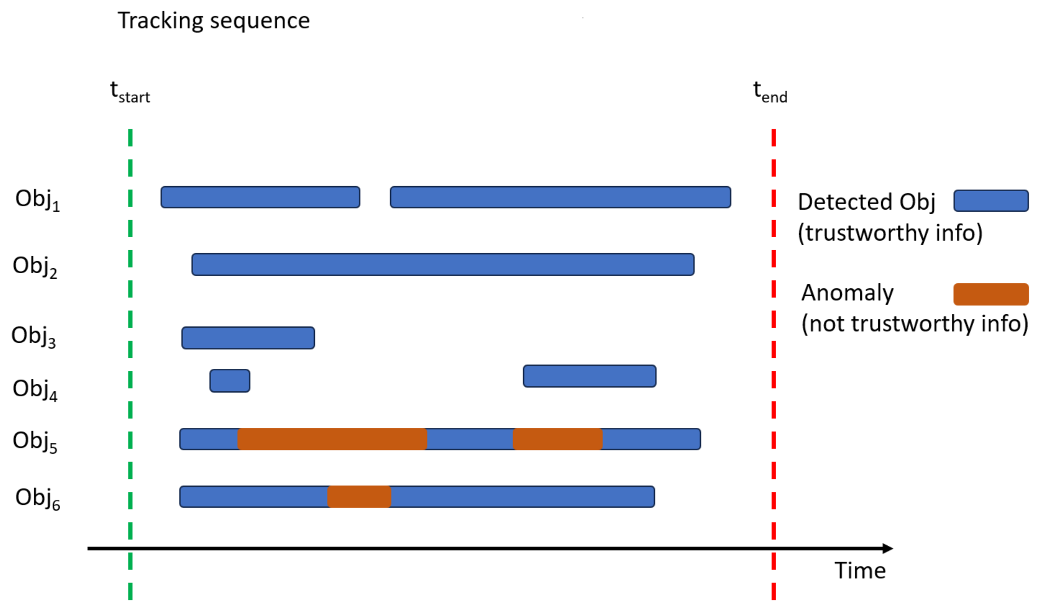

Tracking sequence
t_start
t_end
Obj1
Obj2
Obj3
Obj4
Obj5
Obj6
Detected Obj (trustworthy info)
Anomaly (not trustworthy info)
Time

Figure 4.26. Sequence with six tracked objects.

the OP_{Obj_k} is lower than a user defined threshold OP_{THR} then the object is removed from the tracked objects list and collected in the potential candidates list. Otherwise the object remains in the tracked objects list. Actually, in Figure 4.27, just objects 3 and 4 are removed from tracked objects list because a $OP_{THR} = 50\,\%$ is used.

- Anomaly filter: It checks the amount of anomalies in every object. Anomalies are caused by a combination of position and angle orientation and the J_{THR} value. This combination of values leads sometimes to a wrong association of the incoming boxes with the stored objects. Actually, it can happen that two boxes, related to the same frame, are associated to the same tracked objects by the algorithm. If this happens, an *anomaly* occurs (this definition of anomaly is adopted since now within this work). For every object an anomaly check is performed and whenever an anomaly detection occurs, the anomaly counter AnC_{Obj_k} is increased for the k-th object. Then a *anomaly degree* AnD_{Obj_k} is computed this way:

$$AnD_{Obj_k} = \frac{AnC_{Obj_k} \cdot 2}{n_{Obj_k}} \cdot 100 \tag{4.17}$$

The 2 factor means that the counter AnC_{Obj_k} needs to be considered twice because for every detected anomaly there are two information strings involved (two boxes associated to the same object). In the end if $AnD_{Obj_k} > An_{THR}$ (defined threshold value) then the object is removed by tracked objects list and put inside the potential candidates list. In Figure 4.28 this operation is shown using a $An_{THR} = 30\,\%$: just object 5 is removed from tracked objects list. Then the anomalies inside object 6 are removed (while the object 6 is kept in the tracked list).

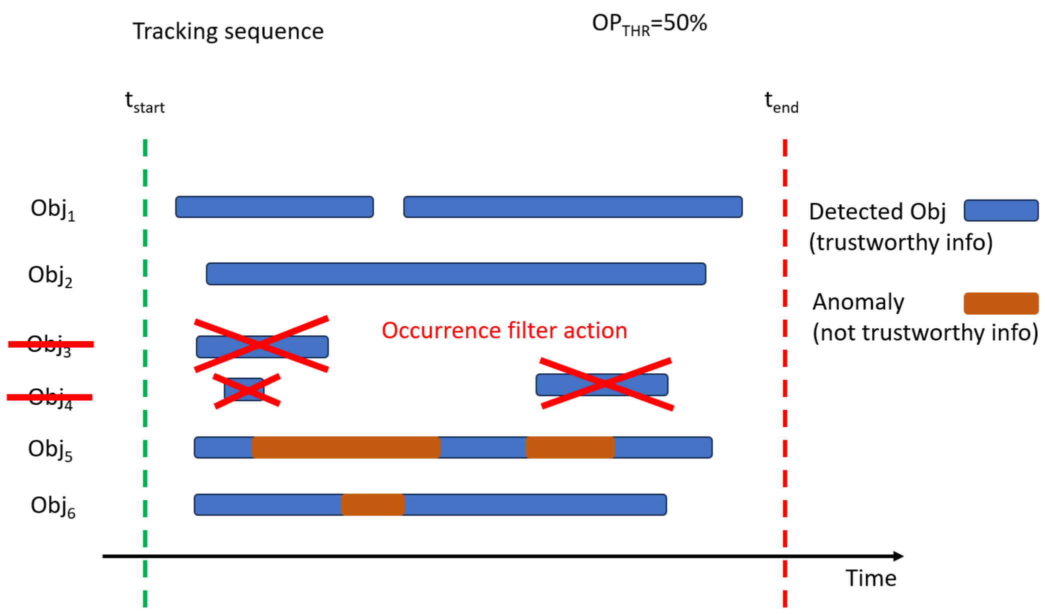


Figure 4.27. Effects of the occurrences filter in the tracked objects list.

In this way, corrupted information is removed but anomalies are still present in the tracked objects list. As a matter of fact, last operation performed by the anomaly filter is removing just the few anomalies contained inside the remaining objects. The removed information is stored in the potential candidates list (Figure 4.29);

- Aggregation filter: This filter acts in a scenario like the one depicted in Figure 4.29. The tracked objects position in the FOV draw straight lines with good approximation. Each line is the trajectory of a different potential RSO. The objects that remain in the tracked objects list after occurrences and anomalies filters are called *nuclei* and represent trustworthy information for the RSOs tracking problem. In the potential candidates list, missing pieces of these tracked objects could still be present. The aggregation filter uses a least square approach to associate potential candidates list information to the nuclei. This further association process differs from the threshold based association (J_{THR}) because now, a priori information (nuclei) is available. In this way, the output will be thrustworthy tracked objects while the remaining not associated candidates will be deleted from the algorithm memory. Nuclei's ideal straight trajectories will be computed using a least square process. These lines will have the number of frames as independent variables and the object centroids coordinates as dependent variables:

$$\begin{cases} x_{obj} = a \cdot n_{frame} + q \\ y_{obj} = b \cdot n_{frame} + k \end{cases} \tag{4.18}$$

The coefficients a, b, q and k will be computed through the least square method applied on the system in Equation 4.19).

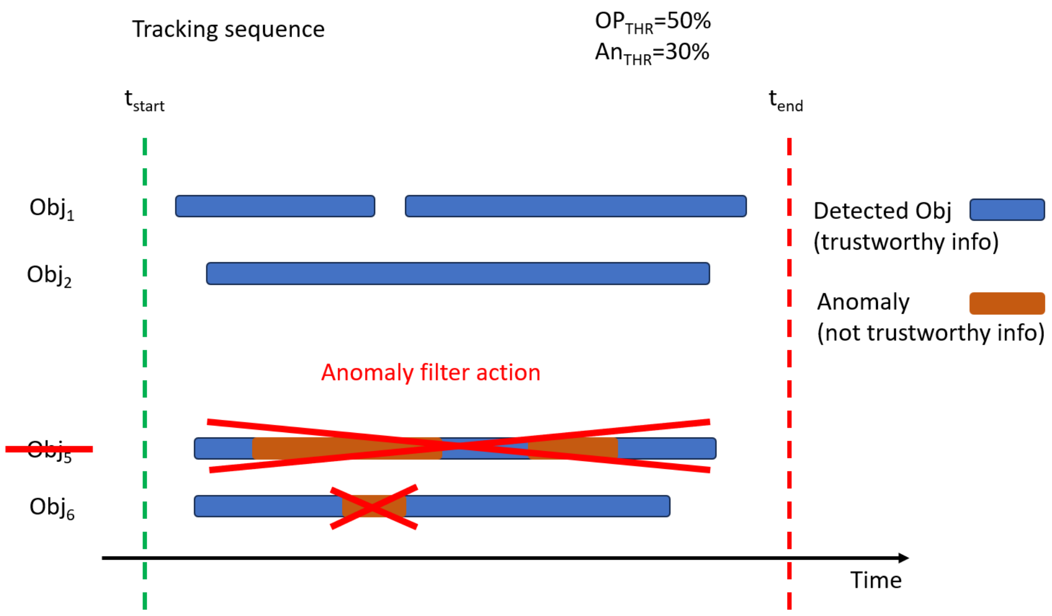


Figure 4.28. Effects of the anomaly filter in the tracked objects list.

$$\begin{cases} \begin{bmatrix} n_{frame_1} & 1 \\ n_{frame_2} & 1 \\ n_{frame_3} & 1 \\ \vdots & 1 \\ n_{frame_m} & 1 \end{bmatrix} \begin{bmatrix} a \\ q \end{bmatrix} = \begin{bmatrix} x_{obj_1} \\ x_{obj_2} \\ x_{obj_3} \\ \vdots \\ x_{obj_m} \end{bmatrix} \\ \begin{bmatrix} n_{frame_1} & 1 \\ n_{frame_2} & 1 \\ n_{frame_3} & 1 \\ \vdots & 1 \\ n_{frame_m} & 1 \end{bmatrix} \begin{bmatrix} b \\ k \end{bmatrix} = \begin{bmatrix} y_{obj_1} \\ y_{obj_2} \\ y_{obj_3} \\ \vdots \\ y_{obj_m} \end{bmatrix} \end{cases} \tag{4.19}$$

By minimizing the differences between the right hand side and left hand side of 4.19 with the least squares approach, it is then possible to know the best fitting straight line for each nucleus. Then, for each detected centroids from the potential candidates list (rem), the distance between it and each nucleus (j) straight lines is computed through:

$$\begin{cases} diff_x = (a_j \cdot n_{frame_{rem}} + q_j) - x_{rem} \\ diff_y = (b_j \cdot n_{frame_{rem}} + k_j) - y_{rem} \\ d = \sqrt{{diff_x}^2 + {diff_y}^2} \end{cases} \tag{4.20}$$

If this distance d is lower than a threshold agg_{THR} then the association is performed. If the distance check is satisfied for more than one nucleus straight

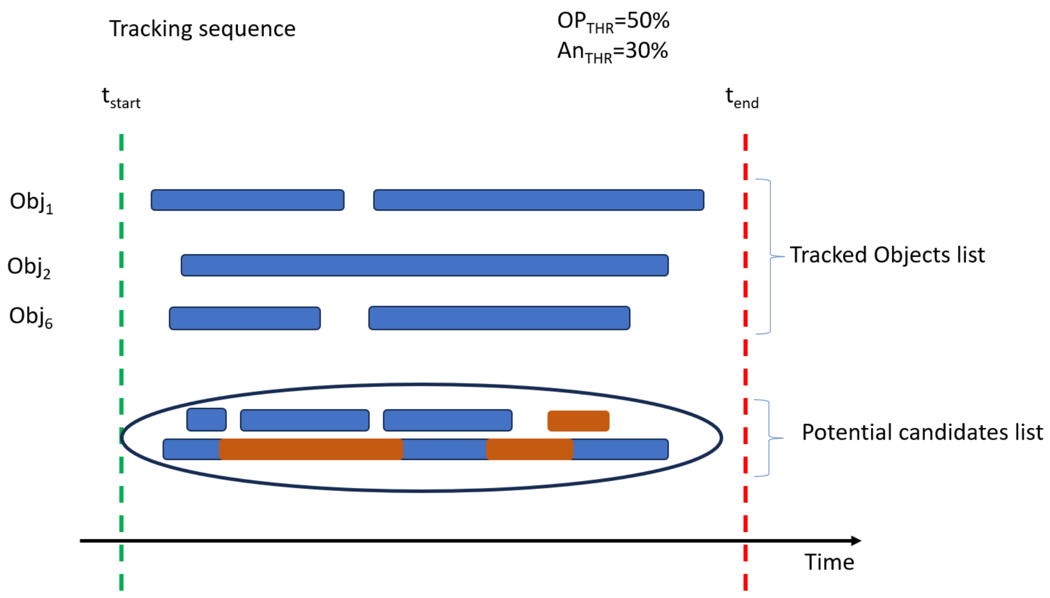


Figure 4.29. Effects of the occurrences and anomaly filter in the tracked objects list.

line, the potential candidate centroid isn't associated to any nucleus in order to remove any possible ambiguity (Figure 4.30).

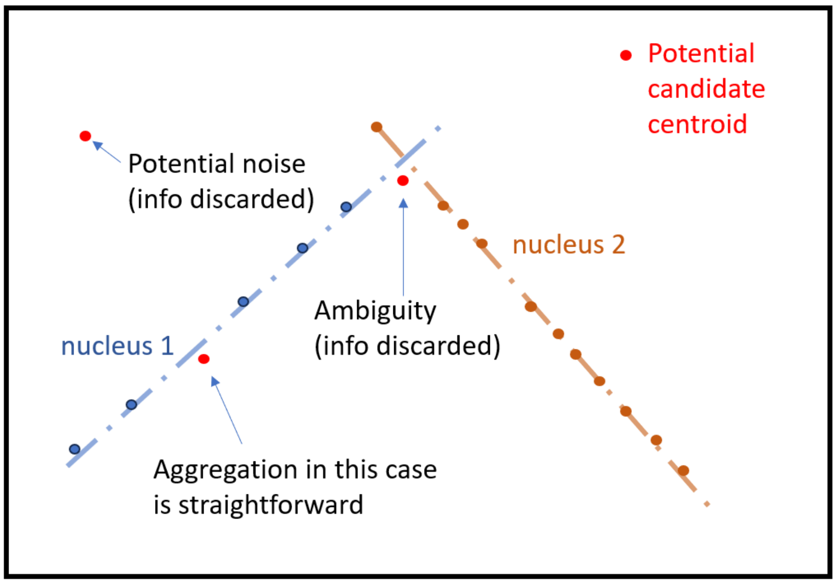

● Potential candidate centroid
Potential noise (info discarded)
nucleus 2
Ambiguity (info discarded)
nucleus 1
Aggregation in this case is straightforward

Figure 4.30. Image plane with RSOs centroids evolution and nuclei's straight lines.

4.2.3 Tests and Results

U-Net and YOLOv3 Training

This section describes the U-Net and YOLOv3 training steps. In particular, the U-Net is configured with the values shown in Table 4.11.

Table 4.11. U-Net Configuration Parameters

Parameter	Value
Image size	960×640
Initial filters	128
Learning Rate	10^{-4}
Regularization Factor	10^{-5}
Dropout Rate	0.25
Kernel Size	3
Kernel initializer	$'he\_normal'$

U-Net training and validation were performed in Tensorflow Keras[28] considering 3 *epochs* and a dimension of the training batch equal to 3. This has been done for 128 filters. The trained network shows reached accuracy higher than 99% and a loss lower than 0.01. Moreover, no overfitting phenomenon occurred.

YOLOv3 training and validation were performed in darknet framework considering the previous dataset. The trained network shows a mean Average Precision (mAP) equal to 78.41% considering a IOU of 50%.

The training, validation and test were performed on a workstation with the following specifications:

- CPU: AMD Ryzen Threadripper PRO 3975WX 32 Cores 3.50 GHz
- RAM: 128 GB

- GPU: Nvidia Quadro RTX 5000.

Further and several approaches for YOLO training have been used to increase the robustness to FNs and FPs. The strategy foresees a training on the first dataset (*original dataset*), then a Transfer Learning (TL) process over the second dataset and finally a second TL over the third dataset. TL process is the training of a previously trained Network: a Network has been trained on a specific dataset, the last saved weights are then loaded and the same network is trained again on a different dataset. This technique is used to increase performances of the network against FPs and FNs. The first TL is performed with another training on a further 5000 images dataset. The second TL with a 30000 images dataset. A second training from scratch has been performed on the third dataset to verify the effectiveness of the double TL strategy. mAP is computed with an IOU equal to 50% and TPs, FPs, FNs values are reported in Table 4.12 for the tested configurations. mAP is calculated by computing AP for each objects class and then average over a number of classes (N):

$$mAP = \frac{1}{N} \cdot \sum_{k=1}^{N} AP_k \tag{4.21}$$

The steps to compute the AP_k values (k is the class index):

1. Generate the prediction scores using the model on the Dataset;
2. Convert the prediction scores to class labels;
3. Calculate the confusion matrix—TP, FP, TN, FN;
4. Calculate the precision and recall metrics for each class;
5. Compute the AP_k as the Area Under the precision-recall Curve for each class;
6. Compute the mAP.

The mAP performance evaluation is carried out on the test subset of the *original dataset.*

Table 4.12. YOLOv3 Performance report

	YOLOv3			
Strategy	**mAP** [%]	**TP** [%]	**FP** [%]	**FN** [%]
Original DS	78.4	72.4	8.5	19.1
First TL	79.8	74.4	8.5	17.1
Second TL	**82.7**	**77.3**	**7.7**	**15.0**
Scratch 30k	78.0	71.9	8.3	19.8

As can be seen, the most promising training strategy for YOLOv3 model is a double TL on different datasets. The percentage of TP increases together with a decrease of FN.

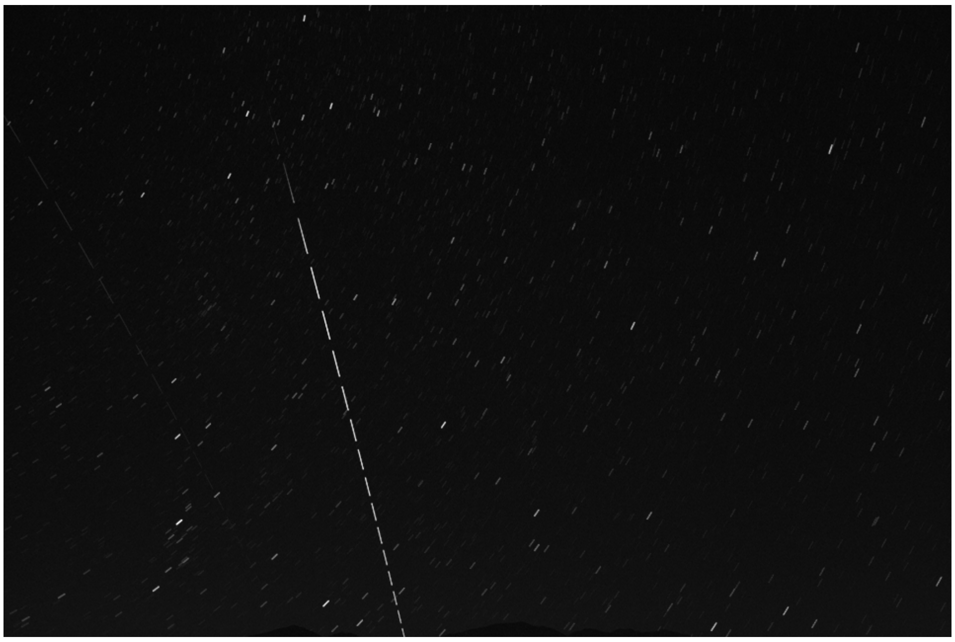


Figure 4.31. ISS and Starlink Passage used for the test (superposed images).

Test on a Real Passage

A sequence of real images is considered to test the Detection and Tracking modules. The whole passage is shown in Figure 4.31, where two objects are moving inside the FOV from top left region towards the bottom-center of the image: the ISS (brightest) and a Starlink (weakest). The sequence was acquired with a reflex camera with the same philosophy described in Figure 4.14 with $T_{exp} = 5sec$ and a $T_p = 2sec$ and then processed. The Objects Detection and Tracking module was able to detect both the objects but not in every image as can be seen in Table 4.13. Here the detected objects' centroids and streaks orientation are shown for every frame. For this test the weights of the *Cost Function* where set to 1 and a static approach for J_{THR} was used considering $\Delta\tilde{X} = 10\,px$, $\Delta\tilde{Y} = 10\,px$ and $\Delta\tilde{M} = 15\,°$.

Just the ISS was clearly detected in almost every frame with the exception of the first frame due to the sun-off/ sun-on transition and the last two frames when the streak is too small and close to the image edge. The proximity to image border causes the YOLOv3 not to detect the objects. This problem is related to the YOLOv3 training where the streaks were more concentrated far away from borders (it is not useful to collect information of streaks close to the borders due to the possibility of out-FOV portion of the streak). The Starlink, instead, appears too weak to be detected in almost every frame and this is due to the brightest ISS which *obscures* it.

Table 4.13 is a table representation of the tracked objects list with $OP_{THR} = 0\%$ and $An_{THR} = 20\%$.The ISS orientation varies due to the rough streaks' orientation estimation approach that confuses the streak's orientation with the YOLO provided bounding box's diagonal orientation (angle differences are due to statistical fluctuations). The sequence was acquired in the Campo Imperatore acquisition campaign, Gran Sasso, L'Aquila, Italy (5-7/07/21). Camera parameters during the acquisition

Table 4.13. Tracking Table for the ISS and Starlink.

Frame	x_{Obj1} $[px]$	y_{Obj1} $[px]$	α_{Obj1} $[deg]$	x_{Obj2} $[px]$	y_{Obj2} $[px]$	α_{Obj2} $[deg]$
1	–	–	–	–	–	–
2	291.26	187.60	67.38	–	–	–
3	–	–	–	–	–	–
4	316.04	281.70	68.20	–	–	–
5	327.25	324.71	66.57	85.60	256.46	57.99
6	337.46	363.92	69.23	–	–	–
7	346.57	399.18	66.50	–	–	–
8	354.44	429.78	68.63	–	–	–
9	362.37	461.05	66.80	–	–	–
10	369.05	487.48	67.17	–	–	–
11	375.27	512.51	67.62	–	–	–
12	381.61	537.91	70.02	–	–	–
13	386.78	559.23	67.62	–	–	–
14	392.18	581.07	66.37	–	–	–
15	397.00	601.01	66.80	–	–	–
16	401.53	620.09	72.47	–	–	–
17	–	–	–	–	–	–
18	–	–	–	–	–	–

are reported in Table 4.14.

Table 4.14. Camera parameters in Gran Sasso campaign.

Features	**Values**
Location	Campo Imperatore, Gran Sasso, Italy
GPS Latitude	42°26’ 11.088" N
GPS Longitude	13°36’ 30.018" E
GPS Elevation	1710 m
Camera	Nikon D3100
Optical System	Nikkor 18 -105 mm
Platform	Tripod
focal length	24 mm
Size	960 x 640 px (Resized)
Exposure Time	5 sec
Pause	2 sec
Vertical FOV	35.57°
Horizontal FOV	51.40°

Performance Indices Evaluation over Real Passages

A total of 31 real passages were collected in the Campo Imperatore acquisition campaign. The algorithm was tested on each image of every passage and an evaluation of Precision, Recall and F1 index is provided over the whole real scenarios set. Following numbers were obtained:

Table 4.15 shows good performance with real images. It suffers more of FNs rather than FPs (this confirms the results in Table 4.12). This means that it is harder for the algorithm to detect a streak when there is no real streak in the image. From the other side the weak eye-detectable objects still represent an issue for the

Table 4.15. Performance Indices

TP	FP	FN	Precision [%]	Recall [%]	F1 [%]
201	13	37	93.93	84.45	88.94

algorithm especially because they are segmented as fragmented streaks. Processing times were measured during every passage and an average time lower than 3 seconds was shown for every image processing. This processing time may be accelerated with a dedicated System on Chip (SoC) implementation that is under investigation. As dynamic memory point of view, less than 60 MB of RAM are used in every image processing.

YOLOv4 Upgrade

After YOLOv3 tests and the problem of FNs, I tried to replace a YOLOv3 with a YOLOv4 and compare their performances. The v4 model was trained at the same way of YOLOv3 and with the same environment and available datasets. Performances of v4 trained models are reported in Table 4.16.

Table 4.16. YOLOv4 Performance report

	YOLOv4			
Strategy	**mAP [%]**	**TP [%]**	**FP [%]**	**FN [%]**
Original DS	82.82	74.7	9.5	15.8
First TL	84.50	75.8	10.1	14.1
Second TL	**85.95**	**77.1**	**10.3**	**12.6**
Scratch 30k	82.12	74.2	10.0	15.8

Interestingly, with the YOLOv4 network double TL, the percentage of FPs is increased if compared with YOLOv3 (see Table 4.12) while the percentage of FNs is decreased (achieving the double TL goal). The FPs percentage increment is due to a multiple and undesired detection of the same streak as shown in the following Figure 4.32. However, this does not constitute a big deal because through a double detection removal filter (applied after the YOLOv4), it is possible to merge the redundant information associated to double detection. The highlighted values in Table 4.16 is referred to the best YOLOv4 trained model which is used in the reported tests (the same applies for YOLOv3 and Table 4.12).

Comparison of YOLOv3 and YOLOv4 Based RSOs Detection Modules

Real RSOs Passages Test

In this section, the computation of TPs, FPs and FNs for the best YOLOv4 model has been performed and shown. It is made on a test set of 31 real RSOs passages which were acquired during a real images acquisition campaign in Campo Imperatore. Moreover a comparison with the best YOLOv3 model performances from my previous investigation is presented [100] in Tables 4.17 and 4.18.

Table 4.17 represents a pseudo confusion matrix due to the impossibility of True Negatives (TNs) computation. If no streak are present in the image, the network is

Table 4.17. YOLOv3 and v4 overall Real RSOs passages performance

YOLOv3			**YOLOv4**		
FP	**FN**	**TP**	**FP**	**FN**	**TP**
13	**37**	**201**	**18**	**26**	**208**

not able to detect the absence of streaks and thus the info of TNs is impossible to be provided.

Table 4.18. YOLOv3 and v4 Real RSOs passages comparison

YOLOv3			**YOLOv4**		
Precision [%]	Recall [%]	F1 [%]	Precision [%]	Recall [%]	F1 [%]
93.93	84.45	88.94	92.04	88.89	90.44

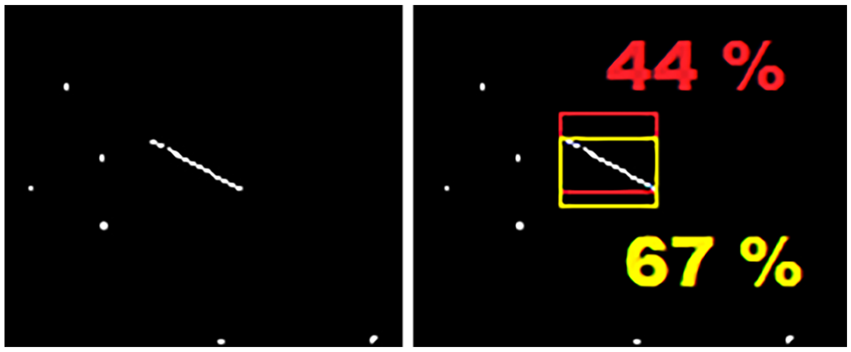


Figure 4.32. An example of double detection of the same RSO. The segmented streak (left). The double detection (right).

Looking at the comparison tables, it is possible to assess that in YOLOv4 predictions there is a perceptible decrement of FN and an increment of FP with respect to YOLOv3 model. In particular, the increased robustness to FNs brings YOLOv4 model to detect correctly heavily fragmented streaks, resulting in an increasing probability of dim target detection. Sometimes the model detects twice clear streaks in the night sky. This was noticed while testing the v4 model on the 31 passages. A case where this occurs is shown in Figure 4.32. The same streak is contained in both the boxes and this does not represent a problem for the *centroid*, *extremal points* and velocity components computation. A double detections removal filter has been implemented and performed before RSOs tracking module (Section 4.2.2).

Test on a Real RSO Passage

In this subsection the ISS and Starlink sequence of 18 images is condensed into a single image (Figure 4.31). This passage was already tested with YOLOv3 model and here the test is repeated with YOLOv4 for comparison purposes. Data of the detected RSOs are collected in Table 4.19. The main differences with respect to the YOLOv3 results (Table 4.13) are:

- Correctly detection of ISS in frame 3 and Starlink in frame 4 thanks to YOLOv4 upgrade and thus to the improved capability in fragmented streaks detection.
- Slight differences in streak orientation values due to an improved way in *extremal point* computation.
- Differences between computed objects coordinates are generally smaller than 1 px. This is not due to a different U-Net segmentation action but to a better way of detecting the area around the box (less stars' foreground pixels in the box which affect the *centroids* and *extremal points* computation).

Table 4.19. ISS and Starlink Tracking Table with YOLO v4

Frame	x_{Obj1} $[px]$	y_{Obj1} $[px]$	α_{Obj1} $[deg]$	x_{Obj2} $[px]$	y_{Obj2} $[px]$	α_{Obj2} $[deg]$
1	–	–	–	–	–	–
2	291.27	187.61	67.75	–	–	–
3	303.95	235.67	66.97	–	–	–
4	316.04	281.70	65.43	61.23	213.97	53.97
5	327.34	325.06	66.37	83.71	253.10	55.89
6	337.46	363.92	65.10	–	–	–
7	346.44	398.65	63.43	–	–	–
8	354.76	431.05	65.38	–	–	–
9	362.32	460.86	64.54	–	–	–
10	369.05	487.48	67.17	–	–	–
11	375.55	512.53	64.65	–	–	–
12	381.51	537.50	64.80	–	–	–
13	387.08	560.15	64.80	–	–	–
14	392.10	580.76	68.20	–	–	–
15	397.09	601.41	68.20	–	–	–
16	401.54	620.10	66.37	–	–	–
17	–	–	–	–	–	–
18	–	–	–	–	–	–

Test with High Fidelity ST Simulator Images

In this test, 20 HFSTS images have been tested with the present algorithm [107] The aim was comparing the developed and latest version of the RSOs detection algorithm against simulated ST images. This because the availability of real on-board images sequence from STs is something very hard to achieve due to the limited downlink band of the existing on-board platforms sensors and telecommunication equipments and the huge weight of each ST image file to be stored on-board (there should be dedicated data collection missions to have more ST's real images data). Thus, the only way to test the algorithm for on-board applications was using a ST simulator. In the test scenario five objects were generated into the FOV while the camera performances are listed in Table 4.20. The simulator is part of the Automation Robotics & Control for Aerospace Laboratory (ARCALab) at School of Aerospace Engineering, Sapienza University of Rome. The simulator is a heritage of ARCALab cooperation in several projects on STs and on-board debris surveillance mission cooperation. For this set the exposure time has been set to 0.2 seconds with a negligible time interval between the end of one acquisition and the starting of the successive one, in order to simulate the behaviour of a possible ST. The algorithm application with simulated ST images provides 5 detected objects where for each frame the computed *centroids* coordinates are shown (Table 4.21). When coordinates are not reported a blank box or a "-" can be found. The blank box means that the detected object is not in the camera FOV anymore while the sign "-" means the presence of the object within the image without a successful detection (FN).

Table 4.20. Simulated ST parameters

Features	**Values**
Camera	HFSTS Cam
Platform	On-board
focal length	51 mm
Size	960 x 640 px
Pixel size	18.0 $\mu m/px$
Exposure Time	0.2 sec
Pause	0 sec
Vertical FOV	13.6°
Horizontal FOV	19.2°

By inspection of Table 4.21 object 1 disappears after frame 8 when it is not correctly detected. Object 2 is always detected while object 3 disappears after frame 17 whith a missed detection at the second frame. Object 4 is correctly detected untill frame 8 when it disappears. Object 5 is detected just in frame 4 because it is heavily fragmented. The last row of Table 4.21 shows the Percentage of Correct Detection (PoCD) which is the ratio between the correct detections and the number of frames in which the object is actually within the FOV. In this test both OP_{THR} and An_{THR} were put equal to 0%. This is the reason why low occurrences objects like the first and fourth are present in the tracking table as the apparent noise too (object 5).

The whole passage composed of 20 images with detected objects is shown in Figure 4.33.

The YOLOv4 network together with the double detection removal filter is able to detect correctly all the 4 out of 5 objects avoiding the undesired FP which occurs in this test.

Effects of the Filters Action on a Real and Noisy Sequence

To prove the correct working and utility of *Occurrences*, *Anomalies* and *Aggregation* filter strategy, their effects in a noisy situation will be shown. It must be remembered that filters are needed to refine the tracking algorithm outputs to provide reliable information for ground OD modules. A sequence of 40 real images acquired in Frascati in winter 2022 is shown. The sequence lasts 280 seconds and sensor specifications are listed in Table 4.22.

Noise conditions were characterized by a strong stray-light due to the town light pollution causing a more challenging situation both for image segmentation and detection module. Several objects appeared crossing the sky but only the most suitable three ones were in the algorithm outputs' list in the end. Used thresholds

Table 4.21. HFSTS tracking results with YOLOv4.

	Obj_1		Obj_2		Obj_3		Obj_4		Obj_5	
Frame	x $[px]$	y $[px]$	x $[px]$	y $[px]$	x $[px]$	y $[px]$	x $[px]$	y $[px]$	x $[px]$	y $[px]$
1	295.5	433.2	366.8	497.6	452.5	200.0	619.6	487.5	–	–
2	273.2	379.9	388.2	484.8	–	–	660.4	462.3	–	–
3	247.0	319.2	407.9	472.1	426.9	182.3	709.8	433.7	–	–
4	221.0	258.9	426.8	460.4	412.3	172.1	755.4	406.1	108.1	506.6
5	193.4	195.2	446.9	448.0	398.2	162.4	801.5	378.8	–	–
6	170.0	140.7	468.5	434.7	383.3	152.1	845.9	352.3	–	–
7	143.6	81.1	487.9	422.9	367.4	141.1	892.4	324.5		
8	–	–	504.1	412.6	351.7	130.0	934.4	299.8		
9			524.3	400.2	334.7	118.5				
10			544.6	387.3	318.6	107.0				
11			564.1	375.5	300.9	94.8				
12			585.2	362.2	281.7	81.5				
13			606.6	349.2	262.2	68.1				
14			624.2	338.3	241.6	53.8				
15			642.5	327.0	222.0	40.2				
16			663.9	313.8	200.8	25.5				
17			684.5	301.0	177.2	9.0				
18			703.4	289.3						
19			723.7	276.8						
20			739.6	266.8						
PoCD [%]	87.50		100		94.11		100		16.67	

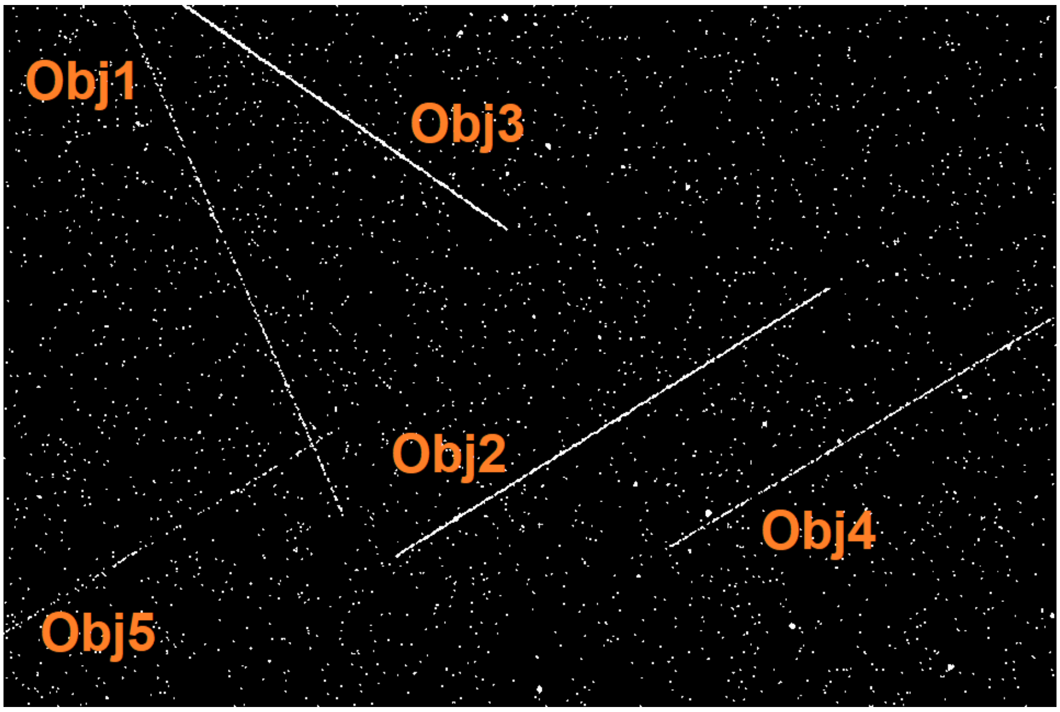


Figure 4.33. High Fidelity ST simulator images passage. The 5 objects are correctly detected.

Table 4.22. Camera parameters in Frascati Night Sky campaign.

Features	**Values**
Location	Frascati, Italy
GPS Latitude	41°48' 32.6" N
GPS Longitude	12°40' 16.9" E
GPS Elevation	269 m
Camera	Nikon D3100
Optical System	Nikkor 18-105 mm
Platform	Tripod
focal length	24 mm
Size	960 x 640 px (Resized)
Exposure Time	5 sec
Pause	2 sec
Vertical FOV	35.57°
Horizontal FOV	51.40°

for this test are: $OP_{THR} = 5\,\%$, $An_{THR} = 20\,\%$ and $agg_{THR} = 50\,px$. Results are visually reported in Figures 4.34 and 4.35.

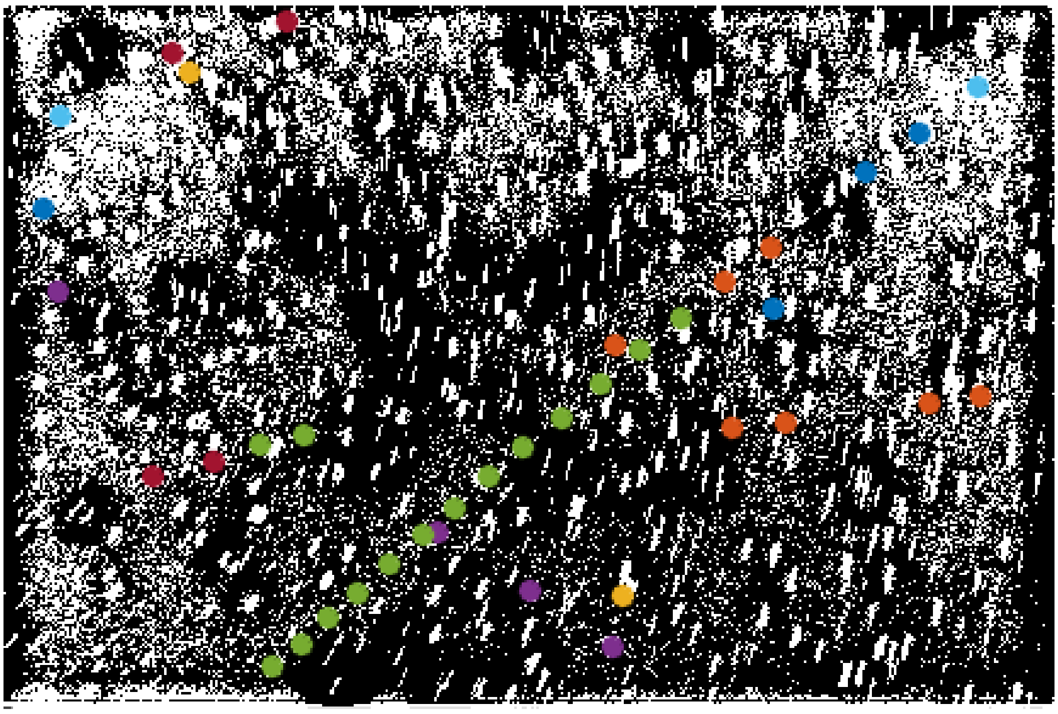


Figure 4.34. Frascati RSOs merged sequence. Tracking outputs' centroids are shown on the merged masks sequence. Each color is a different tracked object. No filter action.

Filters action removed a lot of information which for sparse nature or low numbers could not have any concrete utility to be stored and processed for OD purposes. Another reason to filter the tracking outputs is to reduce the payload used storage memory (thinking about an on-board application). In this way, the limited storage memory will be used for significant and valuable information.

4.3 Summary of Findings for the AI-Based RSOs Detection and Tracking Algorithm

To conclude, this section provides a description, design, development and test of a Convolutional Neural Network based algorithm for Image Processing, Object Detection and Tracking oriented to space optical sensors applications. Details and characteristics of every sub module are presented together with the optimization problem for the Tracking module. An *ad hoc* dataset for *objects detection* sub-module was created and presented with characterising histograms through a streaks simulator in night sky images. Details of U-Net and YOLOv3 training have been provided in terms of accuracy. Module tests over multiple real passages have been carried out showing encouraging preliminary performances of the algorithm in terms of Precision, Recall and F1 indices. The YOLOv3 has shown the correct road for objects detection because of its high accuracy. Then the double TL strategy together with

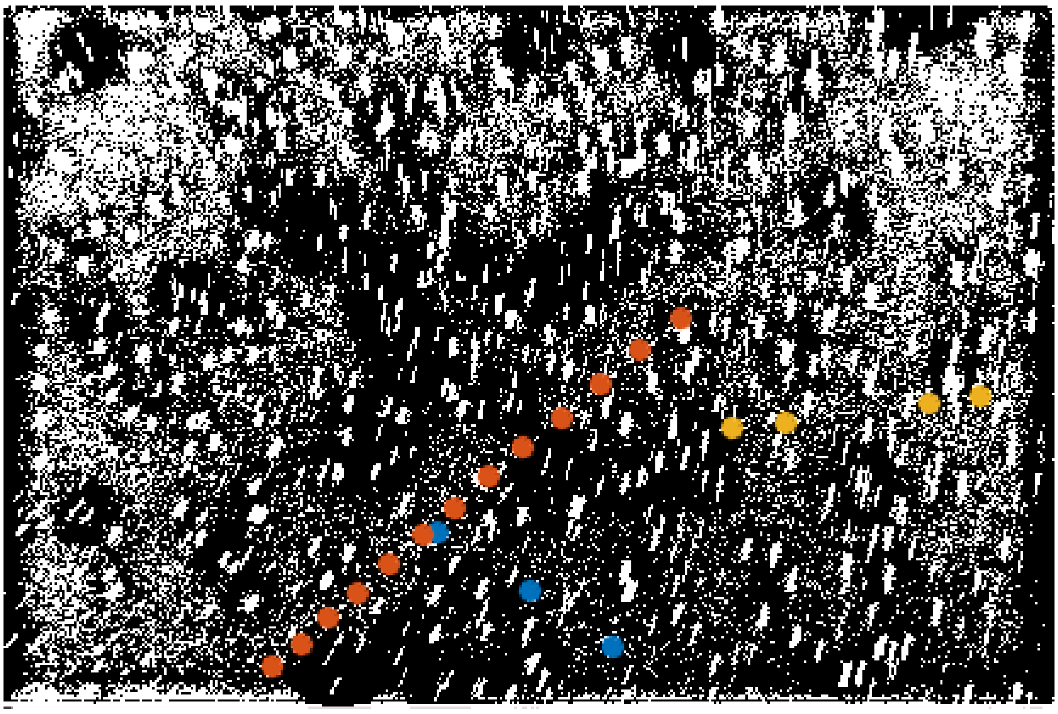


Figure 4.35. Frascati RSOs merged sequence. Tracking outputs' centroids are shown on the merged masks sequence. Each color is a different tracked object. Occurrences, Anomalies and Aggregation filters actions are evident.

the YOLOv4 model increased the network robustness against the most interesting RSOs: dim targets. I chose to adopt the RSOs detection CNN applications to these models and these versions because they are capable of real time working (mandatory for on-board application especially in SSA & SST purposes).

An improvement before *RSOs tracking module* to couple with multiple detections in the first frame together with the merging of redundant information have been implemented: FPs are not a problem anymore for the global algorithm performances.

The implemented and added filters after the *RSOs tracking module* show a refinement of the Tracking outputs list which now contains more reliable information for OD modules. Moreover, an AI-based module will be studied and developed for the *extremal points* computation of the detected streaks. The following areas need further research:

- Improvement of the dataset to be used for the detection task with a data augmentation technique to reduce the percentage of FN;
- Heuristic optimization of *Cost Function* weights through extensive tests (Monte Carlo simulations);
- Analysis of a possible integration of the developed architecture on SoC for a potential on-board application.

Chapter 5

Dual-Purpose of the Segmentation Detection and Tracking Chain

This chapter is devoted to the extension of the U-Net/YOLOv4 based night sky streaks detection algorithm which has been described in the previous sections. The discussed algorithm was designed to detect RSOs under sidereal or near sidereal pointing conditions of the optical payload (or platform). What if there were a not negligible stars motion? In this condition the optical payload has an attitude evolution and thus an angular rate. The streaks are not RSOs anymore but represent stars. Generally, the angular rate (vector quantity) can be derived after a proper star identification and attitude quaternion estimation process. This is possible by the knowledge of the quaternion evolution after a classical finite differences based approach. It guarantees good estimation of the angular rate for low values ($\leq 1°/\text{s}$). Moreover, it is limited to the attitude quaternion knowledge and this is used during normal operations. Sometimes it happens that for some reasons the satellite goes into a SAFE mode after an unexpected event. In this mode the on-board computer put the satellite in a protected condition and it can be recovered by ground telecommands. Often the first step to recover the spacecraft is understanding its motion (orbital and attitude). If the attitude evolution is considerable, then it is really difficult to detect the stars positions and to perform a LISA with the on-board STs. Actually, the number of detectable stars is lower (ST cut-off magnitude decreases with the angular motion increase) and the stars are streaks-like objects therefore classical image processing methods fail (they are designed to better deal with point like objects). By this consideration it seems really hard for modern STs to work under such conditions. This brought me to the idea of using the developed AI-based streaks detection algorithm in such HAR scenarios to know part of the spacecraft attitude motion:

- the streaks can have a considerable length in the FOV and be easily detected by the algorithm;
- streaks can be few in the FOV in HAR conditions and be correctly localized;

- a priori attitude knowledge is not needed if the stars centroids evolution are enough to directly compute the spacecraft angular rate.

The need to retrieve the HAR of the platform without the knowledge of a priori attitude is needed to control and slow down the spinning motion of the platform bringing the classical attitude routines to work in a more favourite condition. The knowledge of the attitude means controlling it and thus, stabilizing it (normal operations retrieval).

This application represents a dual-purpose of the developed AI-based streaks detection algorithm if an angular rate estimation routine with stars centroids as input is developed. During my research I found an algorithm based on least squares which shows considerable performances in angular rate estimation. This independent research brought me to sadly discover that it was already developed and tested for low angular rates scenarios about ten years ago [108]. However, what I did was coupling this method with the previous streaks detection algorithm and pushing the capabilities of this angular rate method in very HAR scenarios. Results are promising for several angular rate vector orientations and noise presence in a wide range of angular speed intensities.

5.1 Space Navigation Application

5.1.1 HAR Estimation Module Description

This algorithm routine is in charge of estimating the spacecraft angular rate through the knowledge of the tracked stars' positions in the FOV. This application is targeted for STs but can be extended to every stars position detection capable sensor. The main assumption in this section is that just stars will be present in the FOV and no real RSOs. Moreover, the spacecraft pointing will not be sidereal anymore but a non negligible and constant angular rate will produce a spacecraft attitude evolution. Stars move in the FOV under HAR conditions leaving tracks which are pieces of ellipse [109]. The AI-based streaks detection algorithm at each frame updates its tracked objects list which will be used to estimate the HAR. The HAR estimation module is based on least square approach starting from a simple kinematics formula:

$$\frac{d}{dt}\hat{b}(t) = \vec{\omega} \times \hat{b}(t) \tag{5.1}$$

Where $\vec{\omega}$ is the spacecraft angular rate while $\hat{b}(t)$ is the generic stars unit vector. The algorithm at each frame knows the stars centroids and thus the stars unit vectors ($\hat{b}(t)$). By a finite differences approach it is possible to know even the time derivatives of such unit vectors ($\frac{d}{dt}\hat{b}(t)$). In this way, at each frame there is a growing number of linked information that can be processed in a least square fashion to know the constant parameter of the problem ($\vec{\omega}$). This is the rationale behind the HAR estimation module and the steps performed by the algorithm follows:

1. Stars centroids to camera Unit Vectors (UVs) conversion;

2. Camera UVs to Platform UVs conversion;

3. Finite differences of Platform UVs and Mean UVs Value computation;

4. Differential Model to Algebraic Model transition;
5. Least Square solution (LS): the Platform angular rate estimation.

Considering a sequence of N frames under angular motion of the ST, generally stars will be in the FOV. The centroid of the i-th star at the j-th frame is indicated as $c_i(t_j) = [x_{i,j}, y_{i,j}]$. Through the camera pin-hole model, and the knowledge of image size, pixel size and camera focal length (f_0) it is possible to compute the corresponding i-th star unit vector at (t_j):

$$\hat{b}_i(t_j) = \begin{bmatrix} b1_{i,j} \\ b2_{i,j} \\ b3_{i,j} \end{bmatrix} \tag{5.2}$$

This vector is expressed into the CRF and needs to be rotated into the Platform Reference Frame (PRF) through a Mounting Matrix (MM) which can be assumed constant and is well known since the mission design. MM indicates how the ST is mounted into the satellite (platform). After the rotation, the same stars UV is expressed in platform coordinates:

$$\hat{X}_i(t_j) = \begin{bmatrix} X1_{i,j} \\ X2_{i,j} \\ X3_{i,j} \end{bmatrix} \tag{5.3}$$

Everything described untill here is related to a single frame. When the next (j+1)-th frame arrives, for the same i-th star two UVs information is available:

$$\hat{X}_i(t_j) = \begin{bmatrix} X1_{i,j} \\ X2_{i,j} \\ X3_{i,j} \end{bmatrix} \;\; and \;\; \hat{X}_i(t_{(j+1)}) = \begin{bmatrix} X1_{i,(j+1)} \\ X2_{i,(j+1)} \\ X3_{i,(j+1)} \end{bmatrix} \tag{5.4}$$

With this information a mean vector can be computed:

$$\bar{X}_i = \left[\frac{\hat{X}_i(t_j) + \hat{X}_i(t_{(j+1)})}{2} \right] \tag{5.5}$$

Which after normalization respresents the PRF i-th star UV ($\tilde{X}_i$) at the time instant ($\frac{t_j + t_{(j+1)}}{2}$). With the vectors of 5.4 it is possible to estimate the i-th star UV time derivative at the same mean instant:

$$\dot{\tilde{X}}_i = \left[\frac{\hat{X}_i(t_{(j+1)}) - \hat{X}_i(t_j)}{(T_{exp} + T_p)} \right] \tag{5.6}$$

In this way with a sequence of N frames there are N-1 ($\dot{\tilde{X}}_i, \tilde{X}_i$) couples for each star. Assuming that there are N_s stars which are being tracked, the total amount of ($\dot{\tilde{X}}_i, \tilde{X}_i$) "points" to be processed after N frames is equal to $(N-1) \cdot N_s$. By this a pair of considerations can be done:

- the number of inputs increases linearly with time if the amount of tracked objects remains constant;

- as the tracking evolves, more information is available and a better angular rate estimate is produced by the algorithm. Actually, if some tracking anomalies occur, their effect on the final HAR estimate is "covered" by the huge and increasing number of information from the correct tracked stars.

The Equation 5.1 is a system of three scalar equations:

$$\begin{cases} \dot{x} = z \cdot \omega_y - y \cdot \omega_z \\ \dot{y} = x \cdot \omega_z - z \cdot \omega_x \\ \dot{z} = y \cdot \omega_x - x \cdot \omega_y \end{cases} \tag{5.7}$$

which can be written in a matrix form:

$$\begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{5.8}$$

Last equation highlights the unknown quantity to be computed. The quantities x, y, z represent the components of $\tilde{X}_i$ while $\dot{x}, \dot{y}, \dot{z}$ represent the components of $\dot{\tilde{X}}_i$. After N frames and N_s tracked stars it is possible to write $(N-1) \cdot N_s$ systems like 5.8 which can be put together obtaining a final over-determined system with just three unknown scalars (the angular rate components $\omega_x, \omega_y, \omega_z$). This system is then solved with a LS approach giving the platform angular rate components in output. Last step is to invert the solution changing the sign to each component. This because the stars centroids motion w.r.t. camera is in the opposite direction of the spinning camera. This algorithm is runned at each frame after the second frame. At each frame the information is processed in a batch fashion, using all the available centroid information from frame 1 to the last.

5.2 Tests of the AI-Based HAR Estimation Algorithm

Tests on the AI-based HAR estimation algorithm have been carried out for five different angular velocity moduli while maintaining the same direction. The camera used in these tests is the same shown in Table 5.1. As initial test the considered PRF direction is:

$$\hat{\omega} = \frac{1}{\sqrt{3}} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{5.9}$$

This direction gives the same angular rate component for each reference axis. The five selected values for the moduli are: 5°/s, 7°/s, 10°/s, 13°/s and 15°/s. The camera and the scenarios were simulated with the HFSTS. At each angular rate module, the sequence of 100 images are generated and prepared to be fed to the algorithm. The exposure time is set to 0.1 sec to simulate the 10 Hz working frequency of most of the commercial STs and the pause time is set to 0 s (because this time interval is generally negligible at these working frequencies). The other parameters (Table 5.1) are similar to commercial STs.

Tested angular rate moduli are really high if compared to the normal angular rates during spacecraft maneuvers ($\leq 1.5°/s$). Each scenarios is simulated for a total

Table 5.1. Simulated ST parameters for HAR.

Features	**Values**
Camera	HFSTS Cam
Platform	On-board
focal length	16 mm
Size	960 x 640 px
Pixel size	6.6 $\mu m/px$
Exposure Time	0.1 sec
Pause	0 sec
Vertical FOV	15.0°
Horizontal FOV	22.4°

of 10 seconds per each simulation. This is enough to test the algorithm capabilities. Results show the trend of the angular rate components errors over this period, the trend of the deviation angle and angular rate norm error. This first test is done just for a HAR direction with no added sensor and camera noise sources. Further test will cover the added noise sources and more tested HAR directions. The AI-based configuration parameters used in these tests are listed in Table 5.2. For each of the reported tests the target machine used is:

- CPU: AMD Ryzen Threadripper PRO 3975WX 32 Cores 3.50 GHz
- RAM: 128 GB
- GPU: Nvidia Quadro RTX 5000.

The above mentioned machine shows an average processing time with this preliminary unoptimized code of $6.4\,sec$ with a $134.22\,MB$ of RAM used for each processed frame.

Table 5.2. AI-based streaks detection and tracking configuration parameters.

Parameter	**Values**
p_{min}	$15\,\%$
IOU_{THR}	$35\,\%$
w_1	1.0
w_2	1.0
w_3	1.0
$\Delta\tilde{X}$	$10\,px$
$\Delta\tilde{Y}$	$10\,px$
$\Delta\tilde{M}$	$15\,°$
OP_{THR}	$5\,\%$
An_{THR}	$10\,\%$

5.2.1 Test at $|\vec{\omega}_{ref}| = 5, 7, 10, 13, 15$ °/s with No Noise Sources

This is the unique angular rate direction to be tested in no noise conditions just to show preliminary algorithm capabilities. The sequence is launched and at each frame the collected data from the moving stars are processed in a batch fashion. At each frame the angular rate estimate is provided and several quantities can be computed:

- Angular rate components errors:

$$\Delta\omega_i = \omega_i - \omega_{refi} \tag{5.10}$$

  This is the error on the i-th component;

- Deviation angle: Angle between the estimated angular rate direction and the reference one;

- Angular rate norm error:

$$\Delta_{norm\%} = \frac{||\vec{\omega}_{est}| - |\vec{\omega}_{ref}||}{|\vec{\omega}_{ref}|} \cdot 100 \tag{5.11}$$

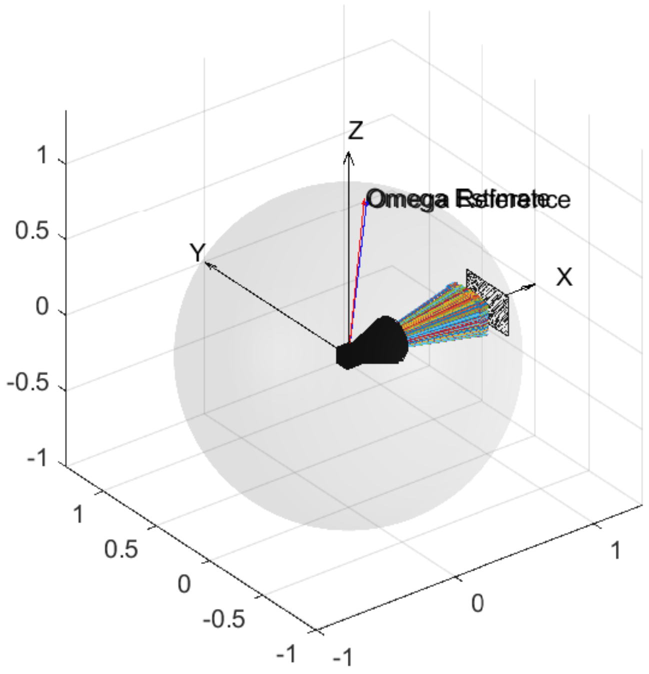


Figure 5.1. Graphic results for the 5 °/s case. The sensor and stars UVs are shown together with the reference and estimated direction for the angular rate. The estimate (red) is close to the reference (blue).

Trends of the errors and deviation follow: Figures 5.2, 5.3 and Tables 5.3, 5.4. Mean values are computed in the last 3 seconds of simulation when the HAR predictions stabilized.

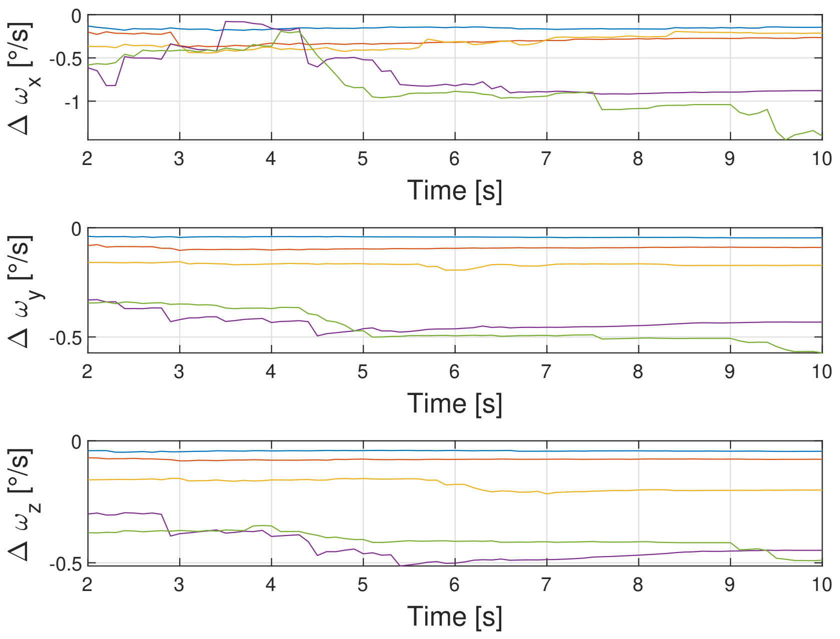


Figure 5.2. HAR components errors trends for the 5 °/s (blue), 7 °/s (orange), 10 °/s (yellow), 13 °/s (purple) and 15 °/s (green) cases and no added noise sources.

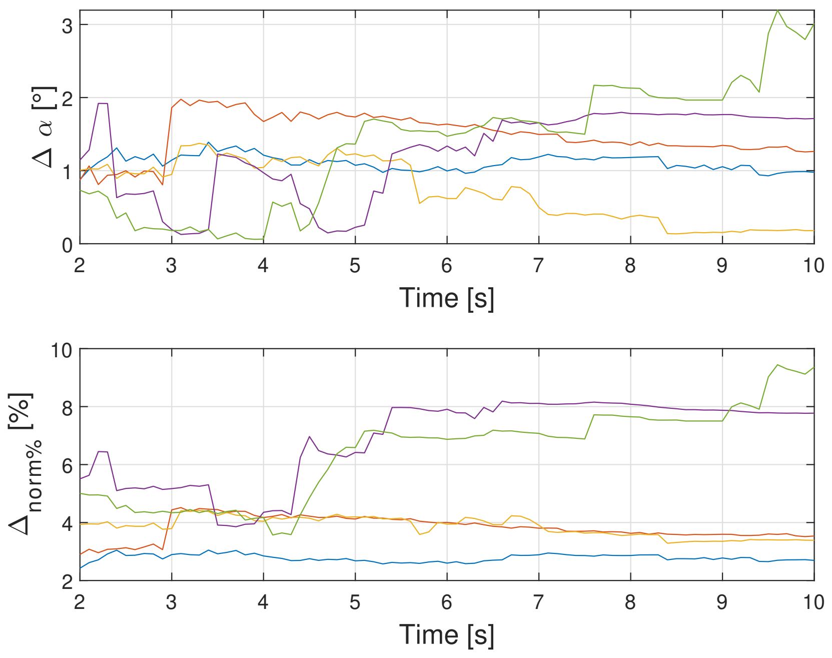


Figure 5.3. HAR components angle and norm deviation trends for the 5 °/s (blue), 7 °/s (orange), 10 °/s (yellow), 13 °/s (purple) and 15 °/s (green) cases and no added noise sources.

Table 5.3. HAR components errors mean values and standard deviations for the 5, 7, 10, 13 and 15 °/s cases with mixed Out of Plane, In Plane motions and no added noise source.

$\vert\vec{\omega}_{ref}\vert$	$\overline{\Delta\omega_x}$	$\overline{\Delta\omega_y}$	$\overline{\Delta\omega_z}$	$\sigma_{\Delta\omega_x}$	$\sigma_{\Delta\omega_y}$	$\sigma_{\Delta\omega_z}$
5 °/s	−0.16°/s	−0.045°/s	−0.042°/s	0.0088°/s	0.00093°/s	0.00077°/s
7 °/s	−0.28°/s	−0.091°/s	−0.075°/s	0.0092°/s	0.0011°/s	0.00058°/s
10 °/s	−0.23°/s	−0.17°/s	−0.20°/s	0.027°/s	0.0031°/s	0.0036°/s
13 °/s	−0.90°/s	−0.44°/s	−0.46°/s	0.013°/s	0.0091°/s	0.013°/s
15 °/s	−1.10°/s	−0.52°/s	−0.43°/s	0.16°/s	0.025°/s	0.029°/s

Table 5.4. HAR deviation estimates at 10 seconds for the 5, 7, 10, 13 and 15 °/s cases with mixed Out of Plane, In Plane motions and no added noise source.

$\vert\vec{\omega}_{ref}\vert$ [°/s]	$\Delta\alpha$ [°]	$\Delta_{norm\%}$ [%]
5	0.98	2.69
7	1.26	3.54
10	0.18	3.39
13	1.71	7.77
15	3.01	9.37

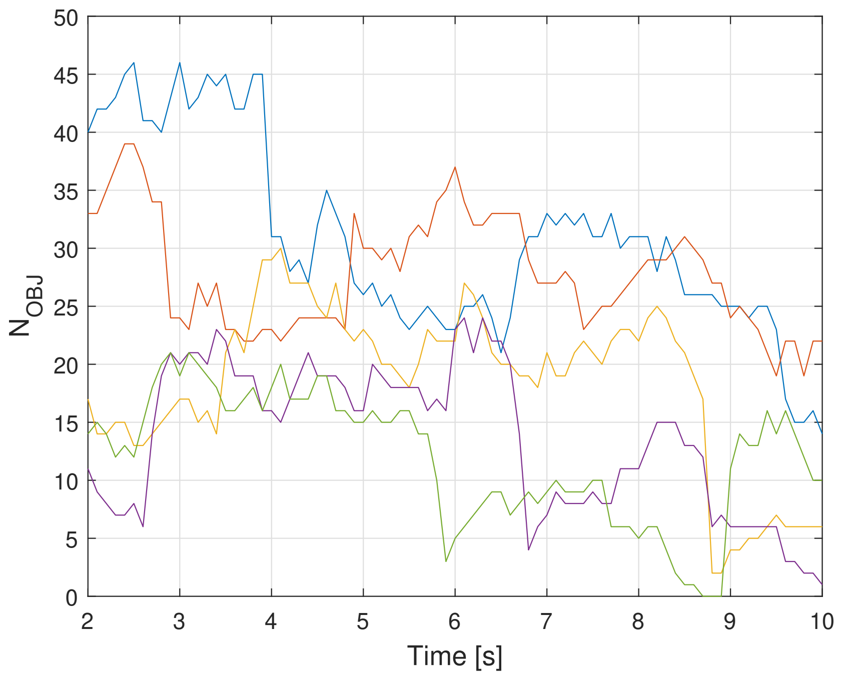


Figure 5.4. Number of stars history for the 5 °/s (blue), 7 °/s (orange), 10 °/s (yellow), 13 °/s (purple) and 15 °/s (green) cases and no added noise sources.

Results show good values for error components, angle deviations and norm deviations for relatively low angular rates, where the measurements are more accurate and improve with observation time. For higher rates, the measurements are less accurate and quickly degrade, mainly because of the decrease in the number of tracked stars. Table 5.3 confirms this trend because it shows the increase of the mean error components and their standard deviations with the angular speed increase (mean values and standard deviation measured on HAR estimates in the last 3 seconds of observation interval). This behaviour is the same for norm deviation and is due to the decrease in number of stars too. Actually, stars' apparent magnitude decreases with the angular rate increase (less stars photons reach the sensor's detector). Thus, the weaker star signal intensity causes both less detected stars and a higher degree of fragmentation for the star streaks in the image masks with a worse centroids estimation process (decrease of the SNR, worse BBs positioning by YOLO).

The error components trends, generally show small values if compared to the reference angular rate (both at low and high rates). They start with sensible fluctuation in values, during first seconds of simulation, and then they stabilize around steady values with reduced fluctuation entities. This fluctuation decrease is due to the growth of available information to be processed with the increasing of time which stabilize the estimate value. The steady state values are affected by all the uncertain information the algorithm collects since the beginning. This uncertain information can come from: close stars (a couple detected as one star), missing stars for several consecutive frames and background noise (the above mentioned apparent magnitude decrease effects). Angle deviation instead, is affected not only by centroids estimates errors but even by the stars distribution in the FOV and thus it depends by the initial attitude and its particular evolution (stars history). This leads to a non linear behaviour of the deviation angle with the angular speed value trend. It can be seen by Figure 5.4 that worst HAR predictions are associated to the lower number of stars. This is a behaviour that is confirmed by further tests. However, the angular rate estimation goodness is confirmed and can be more appreciated by having a look to the angle deviation and norm deviation trends. These always show an angle deviation below 4-5 °, while the norm deviation is always smaller than 10 %. These values are acceptable enough for the recovery of a spacecraft which experienced an anomaly and entered in SAFE mode.

5.2.2 Test for HAR Estimation at $|\vec{\omega}_{ref}| = 5, 7, 10, 13, 15$ °/s with Noise Sources

The only difference in this case is the presence of noise which confuses more the algorithm in detecting and tracking the streaks. Stray-light obscures the weak signal of moving stars while hot pixels have a high value of "virtual" energy and could obscure the streaks for the U-Net based segmentation algorithm (worse mask quality for streaks detection). This is a more challenging scenario whose noise sources and entities are:

- Shot Noise: Poisson probability model and proportional to square root of the detected signal;
- RON: Normal distribution with mean 0 $e-$ and standard deviation of 82 $e-$;

- DC: Constant value of 550 $\frac{e-}{px \times sec}$;
- DSNU: Normal distribution with 1 mean value and a standard deviation of 0.065;
- PRNU: Normal distribution with 1 mean value and a standard deviation of 0.01;
- Stray-light: 30000 $\frac{e-}{sec}$.

These levels are comparable to the commercial STs even if the stray-light level used is too aggressive. The aim of these tests is to verify the robustness of the AI-based algorithm for different HAR directions in a wide angular speed range. In the following pages I first analyze the case with mixed In Plane (IP) and Out of Plane (OoP) rotation components, then a full IP rotation and in the end purely OoP rotations cases.

Test with Mixed Out of Plane and In Plane Motions

In this test the HAR's PRF direction is still:

$$\hat{\omega} = \frac{1}{\sqrt{3}} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{5.12}$$

Thus, the star rotations have both IP and OoP components. They appear as diagonal streaks which cross the FOV 5.5. Now the noise confuses more the image segmentation algorithm by obscuring the streaks (less detected and tracked streaks). Results are summarized in Figures 5.6, 5.7 and Tables 5.5, 5.6.

Table 5.5. HAR components errors mean values and standard deviations for the 5, 7, 10, 13 and 15 °/s cases with mixed Out of Plane and In Plane motions and added sensor noise sources.

$\lvert\vec{\omega}_{ref}\rvert$	$\overline{\Delta\omega_x}$	$\overline{\Delta\omega_y}$	$\overline{\Delta\omega_z}$	$\sigma_{\Delta\omega_x}$	$\sigma_{\Delta\omega_y}$	$\sigma_{\Delta\omega_z}$
5 °/s	−0.12°/s	−0.061°/s	−0.085°/s	0.013°/s	0.0011°/s	0.0020°/s
7 °/s	−0.21°/s	−0.13°/s	−0.13°/s	0.068°/s	0.0083°/s	0.0068°/s
10 °/s	−0.24°/s	−0.30°/s	−0.30°/s	0.051°/s	0.028°/s	0.026°/s
13 °/s	0.53°/s	−0.43°/s	−0.46°/s	0.30°/s	0.017°/s	0.027°/s
15 °/s	−0.35°/s	−0.76°/s	−0.74°/s	0.19°/s	0.032°/s	0.030°/s

Looking at Figures 5.6, 5.7 the off-bore-sight errors have a less fluctuating behaviour with respect to the first one (bore-sight), which is generally the most critical component to evaluate by using STs. The error components mean value increases with the angular speed increase which is due to the more fragmented nature of streaks (bad centroids accuracy). In Table 5.5 it is possible to verify the increase of error components with the reference angular speed increases and the bigger oscillation in the bore-sight component for the last two cases (higher values of $\sigma_{\Delta\omega_x}$ w.r.t. $\sigma_{\Delta\omega_y}$ and $\sigma_{\Delta\omega_z}$). In Table 5.6 the angle deviations are really contained as for the norm deviations. The only ambiguous value interests the 13 °/s case: a high

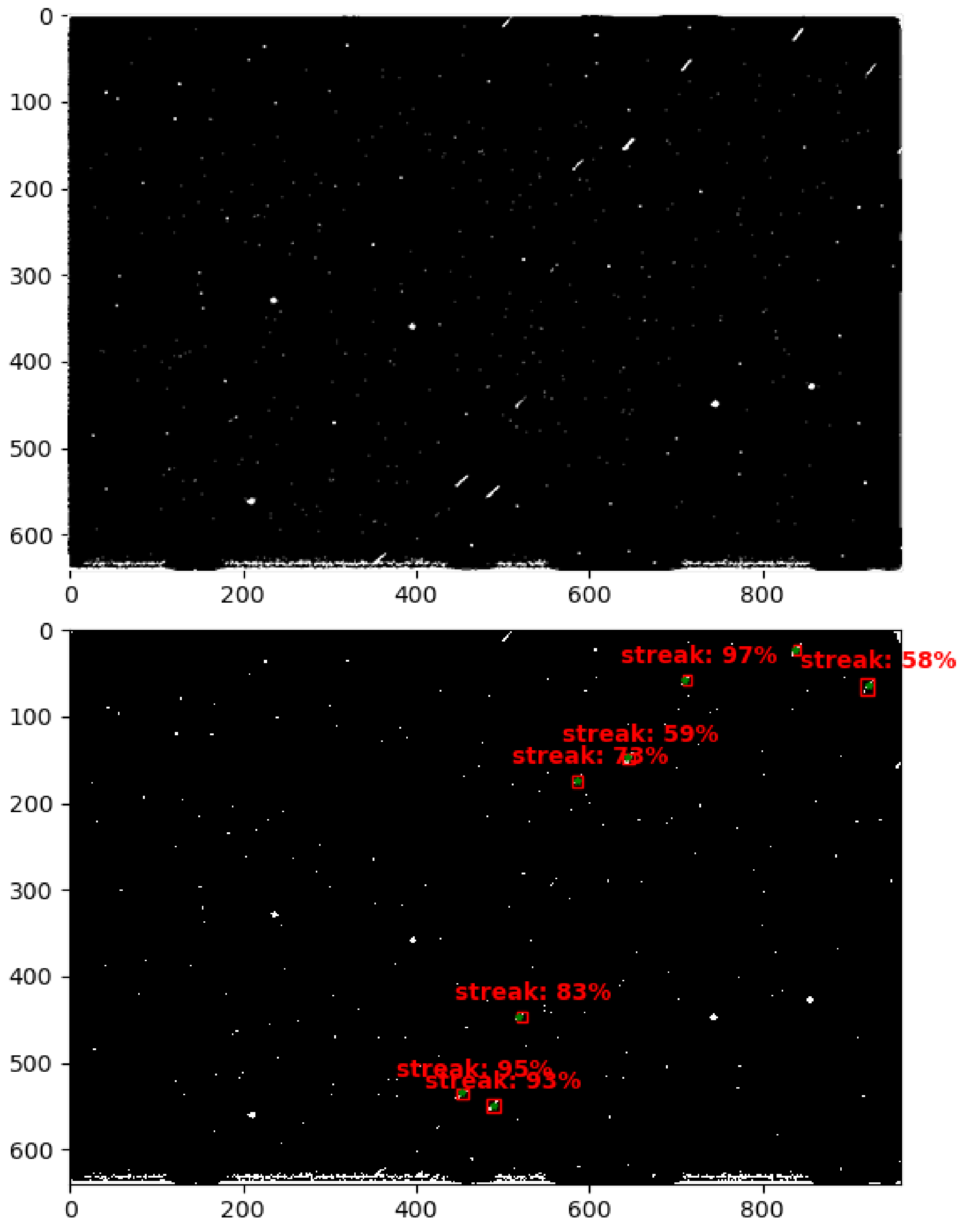


Figure 5.5. IP and OoP stars motion for the HAR case of 5 °/s: Segmented Mask (Top). Detection algorithm outputs (Bottom).

Table 5.6. HAR deviation estimates at 10 seconds for the 5, 7, 10, 13 and 15 °/s cases with mixed Out of Plane and In Plane motions and added sensor noise sources.

$\lvert\vec{\omega}_{ref}\rvert$ [°/s]	$\Delta\alpha$ [°]	$\Delta_{norm\%}$ [%]
5	0.58	3.14
7	0.24	3.03
10	0.70	4.84
13	2.00	3.41
15	0.93	7.13

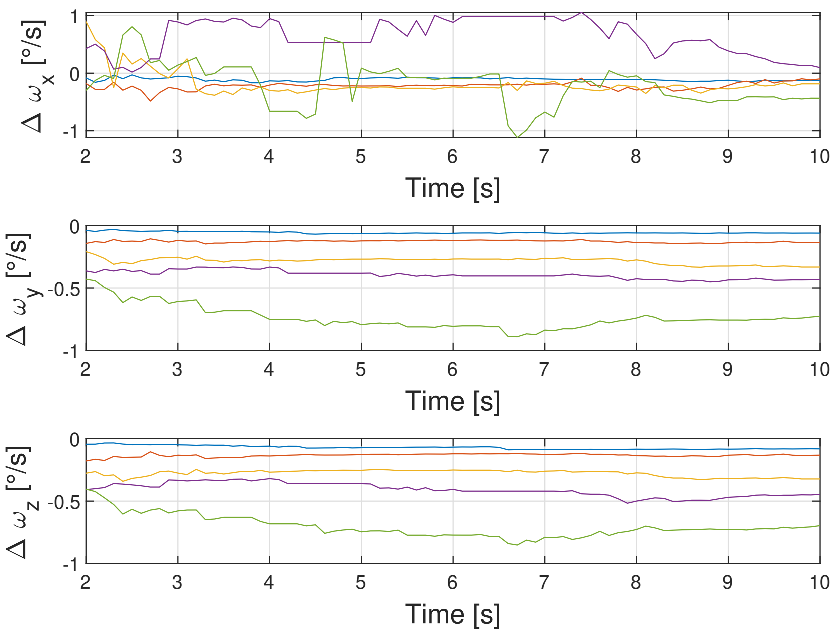


Figure 5.6. HAR components errors trend for the 5 °/s (blue) , 7°/s (orange), 10 °/s (yellow), 13 °/s (purple), 15 °/s (green) cases and angular rate direction parallel to [1,1,1] with added sensor noise sources.

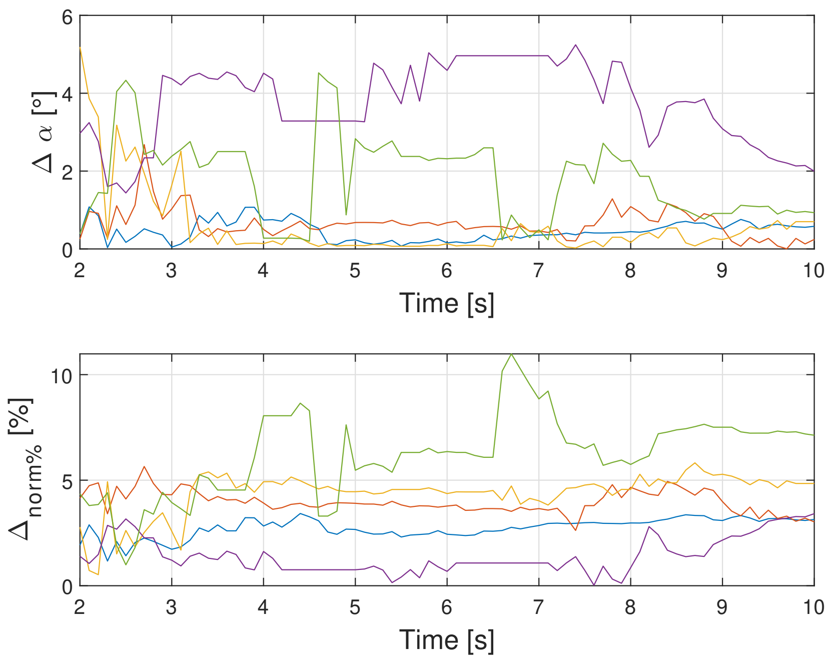


Figure 5.7. HAR angle and norm deviations trend for the 5 °/s (blue) , 7°/s (orange), 10 °/s (yellow), 13 °/s (purple), 15 °/s (green) cases and angular rate direction parallel to [1,1,1] added sensor noise sources.

angle deviation if compared to the other angular speed levels. This is associated with the not negligible fluctuating behaviour shown by the error components discussed before.

Investigations for the 13 °/s case occurred and trends of the number of tracked stars are closely correlated with the error components evolution. Actually, frames where the number of stars is higher and almost constant show lower errors and better final predictions. While in 13 °/s case the higher value of the speed and the random initial attitude brings to a bad number of star history which is characterized by an initial number of few stars ($n_{OBJ} \leq 3/4$). Then a 1 second time interval with no tracked stars, 1 second with a single tracked star followed by another 1 second with no stars. This initial bad star history makes the error estimate to be high and to grow during the time. The prediction improves in the last seconds ($n_{OBJ} \simeq 8-9$): however, the final results is affected by the whole "bad history" in terms of tracked stars because information are processed in a batch fashion. Trend of first error component for 5 °/s and 13 °/s cases are compared with the respective number tracked objects histories to prove what has been just discussed (Figure 5.9).

The random initial attitude is something that, combined with the angular speed and direction, influences the number of star history (and thus the final HAR prediction). This is something on which there is no control. However, this algorithm can be run several times in real missions to mitigate the initial attitude effect and overcoming this issue (every time the ST sees a different portion of sky, a different number of tracked objects histories occur).

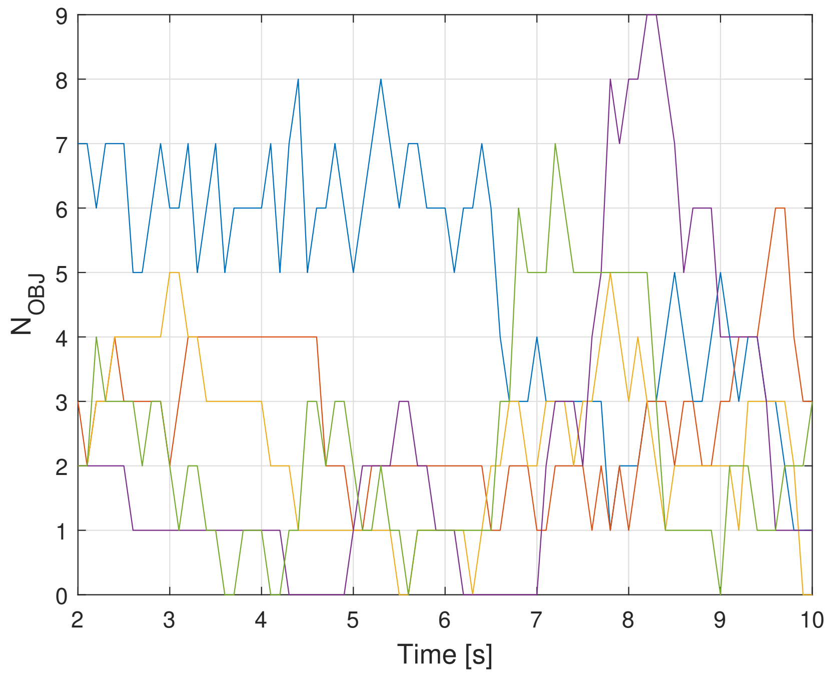


Figure 5.8. Number of stars history for the 5 °/s (blue), 7 °/s (orange), 10 °/s (yellow), 13 °/s (purple) and 15 °/s (green) cases. Direction parallel to [1,1,1] and added noise sources.

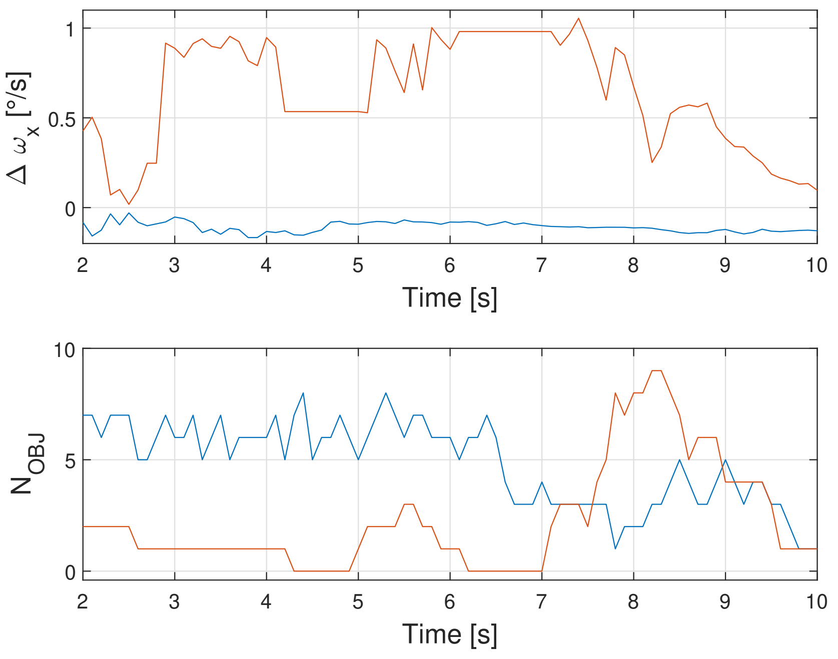


Figure 5.9. HAR bore-sight component errors trend for the 5 °/s (blue) and 13 °/s (orange) cases (top). Number of tracked stars trend (bottom). Results are related to angular rate direction parallel to [1,1,1] with added sensor noise sources.

Another aspect to consider with this particular angular rate direction is the effect of the noise. Results in Figures 5.6, 5.7, 5.8 and Tables 5.5, 5.6 can be compared with the noise-less test (Figures 5.2, 5.3, 5.4 and Tables 5.3, 5.4). This comparison shows:

1. The number of tracked stars has decreased a lot if compared with the noise-less case;

2. Greater error component values and variances for the noise sources test w.r.t. the noise-less case;

3. A norm error which is comparable at lower angular rates.

The second statement is not true for the bore-sight error component of the angular rate estimate. This is due to the really high number of tracked stars in the noise-less scenarios which introduces too much centroids errors at high speeds causing a reduction of the angular rate bore-sight component prediction. Actually, in the cases with a huge number of star density, the YOLO can detect close stars within the same box. This leads to a centroid estimation error. These wrong cases are frequently in noise-less scenarios and causes a reduction in angular rate estimate accuracy if compared with a noisy images sequence (further improvements in YOLO predictions will occur but here the algorithm design is at a preliminary level). The third is not true at the 13 °/s and 15 °/s because of the bore-sight component behaviour.

In the end, noise and high speeds cause less tracked stars and worse centroids estimation errors which affect the angular rate prediction in different ways:

- When the tracked stars decrease or are very low (≤ 5), the angular rate estimate gets worse;

- When the number of stars is really high, the introduced centroids errors can cause a worse estimate of the angular rate direction and intensity.

High angular speed effect cannot be eliminated, initial attitude is something on which no control is possible, number of star history depends from the previous effects. The only way to push the HAR prediction beyond these errors is improving the centroids estimates and using a more sensible sensor to detect objects in very HAR scenarios.

The angular rate prediction is affected by the these effects. Some mitigation actions can be actuated:

- Sensor resolution can be increased while maintaining the same optics;

- Fragmented streak shape reconstruction module may be applied to retrieve a close-original shape (not investigated yet);

- Improving the U-Net capabilities in streaks segmentation could avoid the need of a streak shape reconstruction module after segmentation (not investigated yet).

These actions would increase the angular rate estimates accuracy even in presence of a high and low number of stars (better input data quality).

Test with Full In Plane Stars Motions

In this test the HAR's PRF direction is:

$$\hat{\omega} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{5.13}$$

Thus, the star rotations will have just IP components. All of the stars will rotate around the image plane center describing concentric circles with uniform speed. Stars close to the image center will have a tangential velocity of low intensity while the outer stars will have a higher velocity. Because of fixed exposure time working, the inner stars will result as point like objects while the outer will show up as streaks. This will be more evident as the angular speed increases.

Table 5.7. HAR components errors mean values and standard deviations for the 5, 7, 10, 13 and 15 °/s cases with full IP stars motion with added noise sources.

$\|\vec{\omega}_{ref}\|$	$\overline{\Delta\omega_x}$	$\overline{\Delta\omega_y}$	$\overline{\Delta\omega_z}$	$\sigma_{\Delta\omega_x}$	$\sigma_{\Delta\omega_y}$	$\sigma_{\Delta\omega_z}$
5 °/s	−°/s	−°/s	−°/s	−°/s	−°/s	−°/s
7 °/s	−°/s	−°/s	−°/s	−°/s	−°/s	−°/s
10 °/s	−0.71°/s	0.064°/s	0.085°/s	0.098°/s	0.027°/s	0.0076°/s
13 °/s	−0.22°/s	0.025°/s	0.016°/s	0.031°/s	0.0042°/s	0.0037°/s
15 °/s	−0.15°/s	0.0025°/s	0.011°/s	0.013°/s	0.0035°/s	0.0017°/s

Table 5.8. HAR deviation estimates at 10 seconds for the 5, 7, 10, 13 and 15 °/s cases with full IP motions and added sensor noise sources.

$\|\vec{\omega}_{ref}\|$ [°/s]	$\Delta\alpha$ [°]	$\Delta_{norm\%}$ [%]
5	−	−
7	−	−
10	0.67	7.66
13	0.15	1.56
15	0.049	1.14

By Figures 5.10, 5.11, 5.12 and Tables 5.7, 5.8 it can be noticed that no results for the first two angular speeds are obtained. This is due to the detection module characteristic of detecting just streaks and not point-like objects. The combination of exposure times and angular speed under 10 °/s does not let the network to detect streaks. Stars are point-like and the YOLOv4 model is not trained to detect points. This issue can be easily overcome by coupling this AI-based streaks detection module with a classical point-like detection algorithm(not investigated yet). For higher angular speed values the algorithm behaves well, giving good predictions both in direction and norm. Even here, the low number of detected stars and long intervals with no tracked stars brings error components to start with high values which decrease in the end due to a greater amount of collected centroids. In particular in Figure 5.10 the highest error components occur for the 10 °/s case where the number of tracker stars is really few due to the lower angular speed (few stars detected in

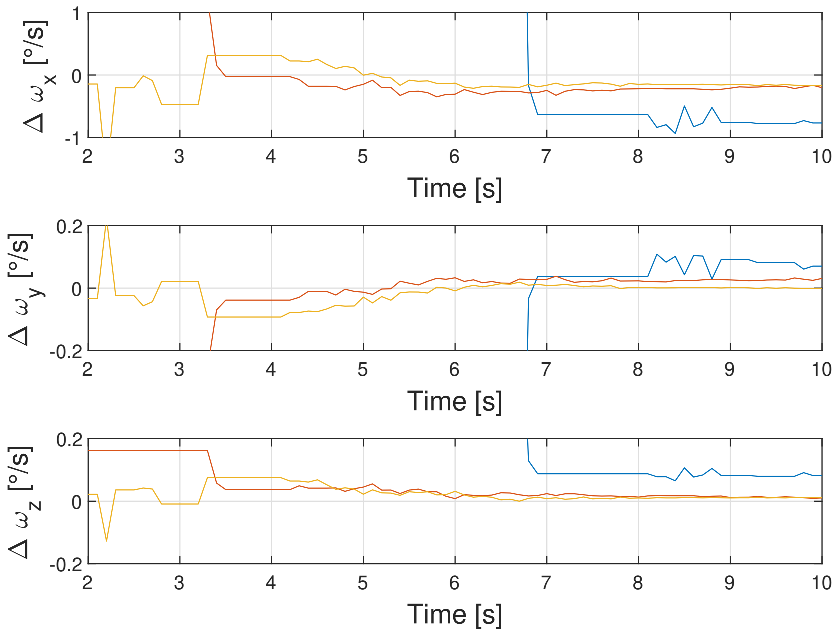


Figure 5.10. HAR components errors trend for the 5 °/s (absent) , 7°/s (absent), 10 °/s (blue), 13 °/s (orange), 15 °/s (yellow) cases and angular rate direction parallel to [1,0,0] with noise sources.

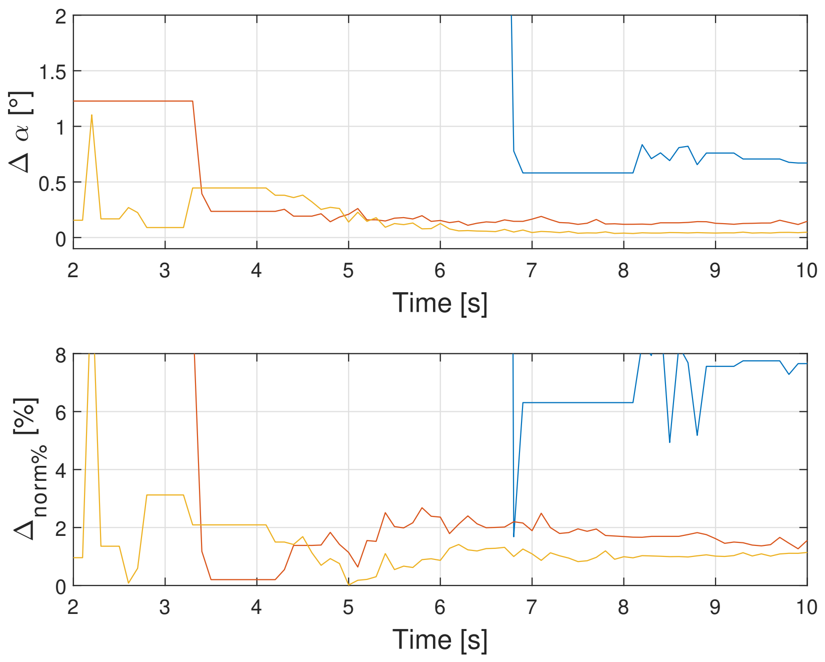


Figure 5.11. HAR angle and norm deviations trend for the 5 °/s (absent) , 7°/s (absent), 10 °/s (blue), 13 °/s (orange), 15 °/s (yellow) cases and angular rate direction parallel to [1,0,0] with noise sources.

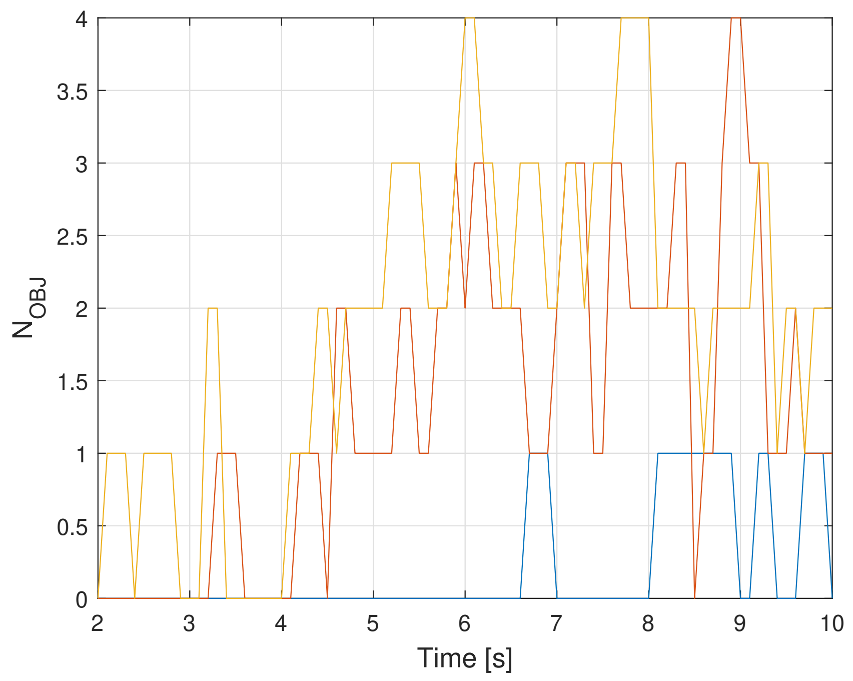


Figure 5.12. Number of stars history for the 5 °/s (absent) , 7°/s (absent), 10 °/s (blue), 13 °/s (orange), 15 °/s (yellow) cases with noise and angular rate direction parallel to [1,0,0].

the outer FOV region). As the speed increase, the streak-like objects increase, the tracked stars increase and the prediction error rapidly improves.

Test with Full Out of Plane Motions (Vertical Streaks)

In this test the HAR's PRF direction is:

$$\hat{\omega} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{5.14}$$

Thus, the star rotations will have just OoP component. All of the stars will appear as vertical streaks while moving in the FOV.

Table 5.9. HAR components errors mean values and standard deviations for the 5, 7, 10, 13 and 15 °/s cases with full OoP stars motion and stars as vertical streaks.

$\lvert\vec{\omega}_{ref}\rvert$	$\overline{\Delta\omega_x}$	$\overline{\Delta\omega_y}$	$\overline{\Delta\omega_z}$	$\sigma_{\Delta\omega_x}$	$\sigma_{\Delta\omega_y}$	$\sigma_{\Delta\omega_z}$
5 °/s	0.057°/s	−0.11°/s	−0.012°/s	0.041°/s	0.0092°/s	0.0040°/s
7 °/s	0.095°/s	−0.39°/s	0.0015°/s	0.22°/s	0.035°/s	0.0040°/s
10 °/s	−0.41°/s	−0.44°/s	−0.031°/s	0.57°/s	0.11°/s	0.042°/s
13 °/s	−1.31°/s	−0.92°/s	−0.054°/s	0.89°/s	0.082°/s	0.052°/s
15 °/s	0.38°/s	−4.42°/s	−0.018°/s	0.67°/s	5.81°/s	0.055°/s

Table 5.10. HAR deviation estimates at 10 seconds for the 5, 7, 10, 13 and 15 °/s cases with full OoP stars motion and stars as vertical streaks.

$\|\vec{\omega}_{ref}\|$ [°/s]	$\Delta\alpha$ [°]	$\Delta_{norm\%}$ [%]
5	0.30	2.48
7	1.12	6.08
10	0.41	3.77
13	5.71	6.79
15	0.69	8.40

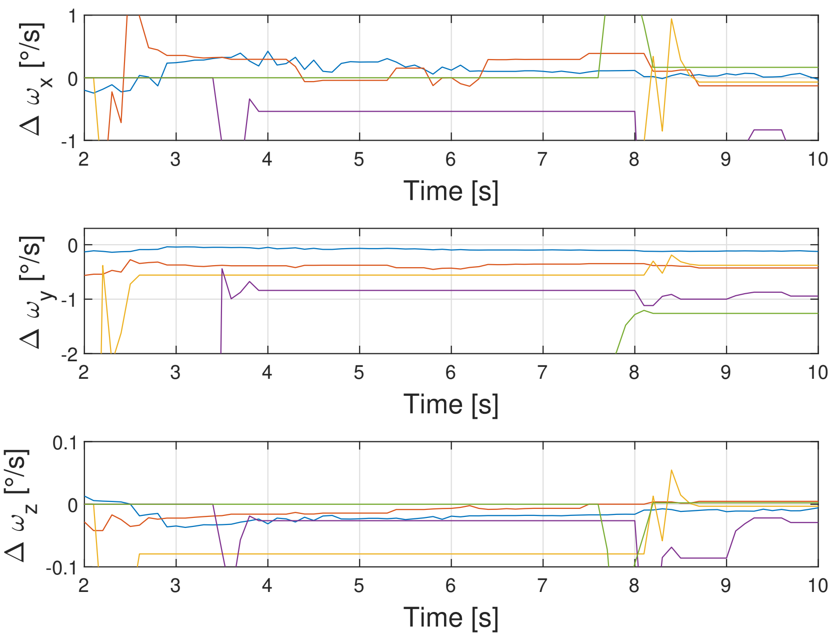


Figure 5.13. HAR components errors trend for the 5 °/s (blue) , 7°/s (orange), 10 °/s (yellow), 13 °/s (purple), 15 °/s (green) cases and angular rate direction parallel to [0,1,0] with added noise sources.

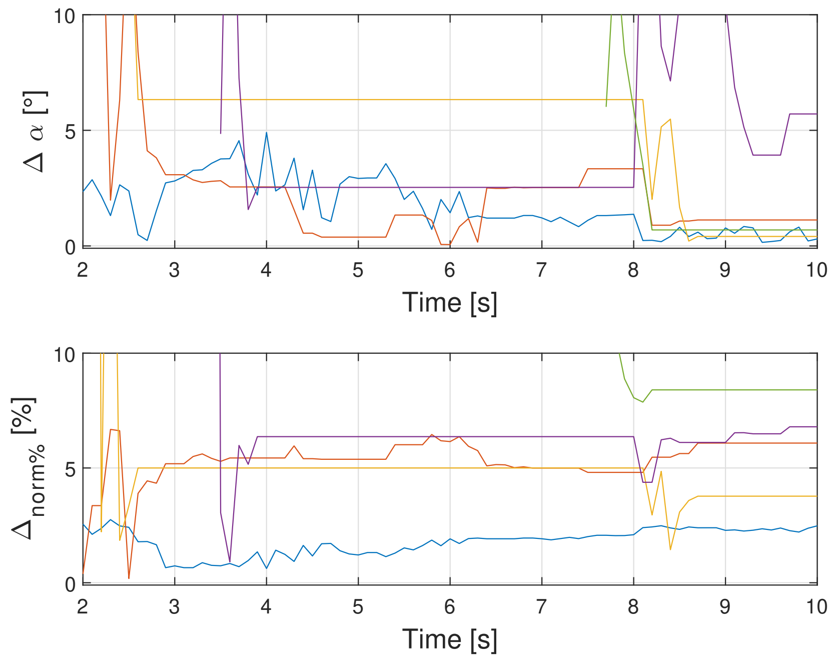


Figure 5.14. HAR angle and norm deviations trend for the 5 °/s (blue) , 7°/s (orange), 10 °/s (yellow), 13 °/s (purple), 15 °/s (green) cases and angular rate direction parallel to [0,1,0] with added noise sources.

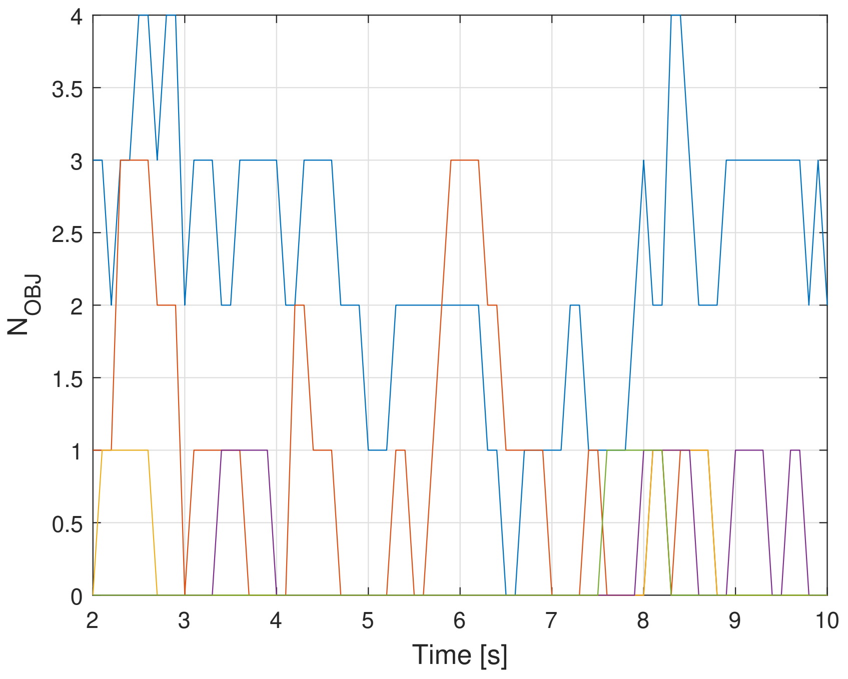


Figure 5.15. Number of stars history for the 5 °/s (blue), 7 °/s (orange), 10 °/s (yellow), 13 °/s (purple) and 15 °/s (green) cases and added noise sources ([0,1,0] angular rate direction).

From Figures 5.13, 5.14, 5.15 and Tables 5.9, 5.10 it is possible to see the HAR prediction degradation with the increase of the angular speed. In particular in the latest case the deeply bad star history cause a huge error in the HAR norm prediction.

Test with Full Out of Plane Motions (Horizontal Streaks)

In this test the HAR's PRF direction is:

$$\hat{\omega} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.15}$$

Thus, the star rotations will have just OoP component. All of the stars will appear as horizontal streaks while moving in the FOV.

Table 5.11. HAR components errors mean values and standard deviations for the 5, 7, 10, 13 and 15 °/s cases with full OoP stars motion and stars as horizontal streaks.

$\lvert\vec{\omega}_{ref}\rvert$	$\overline{\Delta\omega_x}$	$\overline{\Delta\omega_y}$	$\overline{\Delta\omega_z}$	$\sigma_{\Delta\omega_x}$	$\sigma_{\Delta\omega_y}$	$\sigma_{\Delta\omega_z}$
5 °/s	0.37°/s	−0.012°/s	−0.051°/s	0.031°/s	0.0023°/s	0.0074°/s
7 °/s	−0.079°/s	0.0082°/s	−0.22°/s	0.098°/s	0.0045°/s	0.011°/s
10 °/s	−0.19°/s	−0.017°/s	−0.25°/s	0.19°/s	0.013°/s	0.045°/s
13 °/s	−0.60°/s	0.0055°/s	−0.32°/s	0.076°/s	0.0041°/s	0.027°/s
15 °/s	−0.28°/s	0.019°/s	−0.53°/s	0.14°/s	0.0032°/s	0.023°/s

Table 5.12. HAR deviation estimates at 10 seconds for the 5, 7, 10, 13 and 15 °/s cases with full full OoP stars motion and stars as horizontal streaks.

$\lvert\vec{\omega}_{ref}\rvert$ [°/s]	$\Delta\alpha$ [°]	$\Delta_{norm\%}$ [%]
5	3.81	0.96
7	0.85	3.14
10	0.84	2.61
13	2.57	2.51
15	1.20	3.34

From Figures 5.16, 5.17, 5.18 and Tables 5.11, 5.12 it is possible to see the HAR prediction degradation with the increase of the angular speed. Now, predictions are better than the vertical streaks tests. Having a look to the number of stars history (Figure 5.18) it is possible to understand that this is due to a higher number of tracked stars. The improvement in the prediction is possible thanks to the particular sensor (rectangular with a wider HFOF than VFOV) which leads to the higher residence time of the streaks in the FOV.

5.3 Test with RSOs in HAR Conditions

What if now there is a rotating sensor which experience a RSO's passage? How can it resolve the RSOs from the coherent rotational star motion? As a professor of

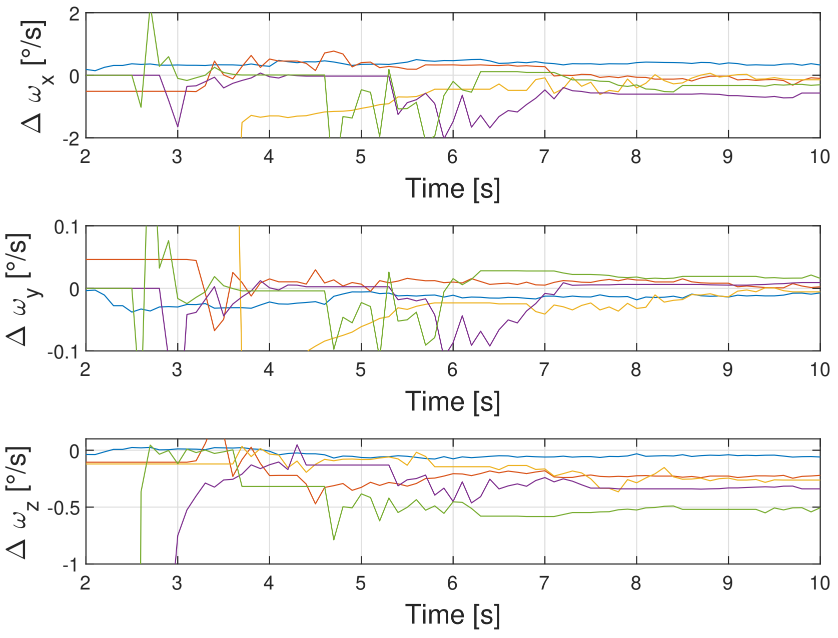


Figure 5.16. HAR components errors trend for the 5 °/s (blue) , 7°/s (orange), 10 °/s (yellow), 13 °/s (purple), 15 °/s (green) cases and angular rate direction parallel to [0,0,1] with added noise sources.

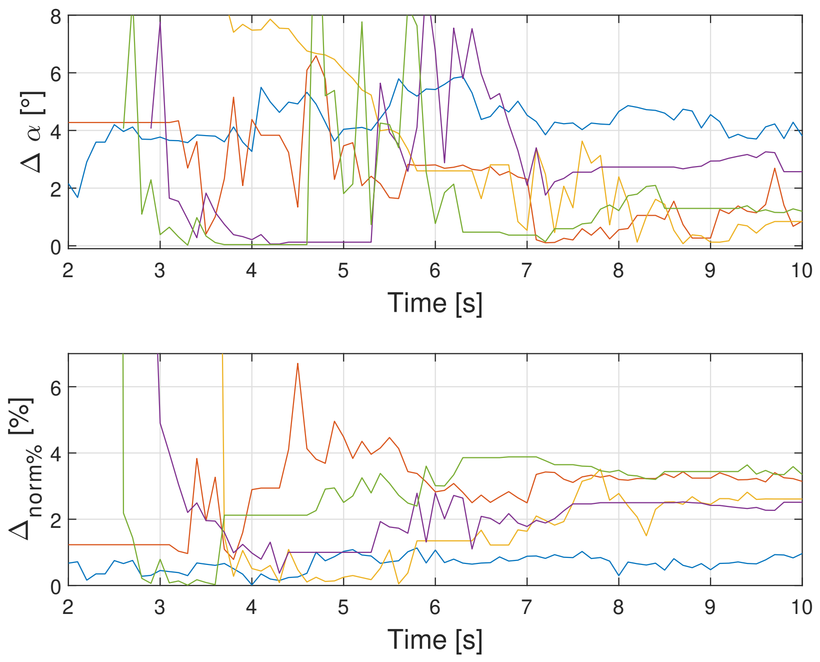


Figure 5.17. HAR angle and norm deviations trend for the 5 °/s (blue) , 7°/s (orange), 10 °/s (yellow), 13 °/s (purple), 15 °/s (green) cases and angular rate direction parallel to [0,0,1] with added noise sources.

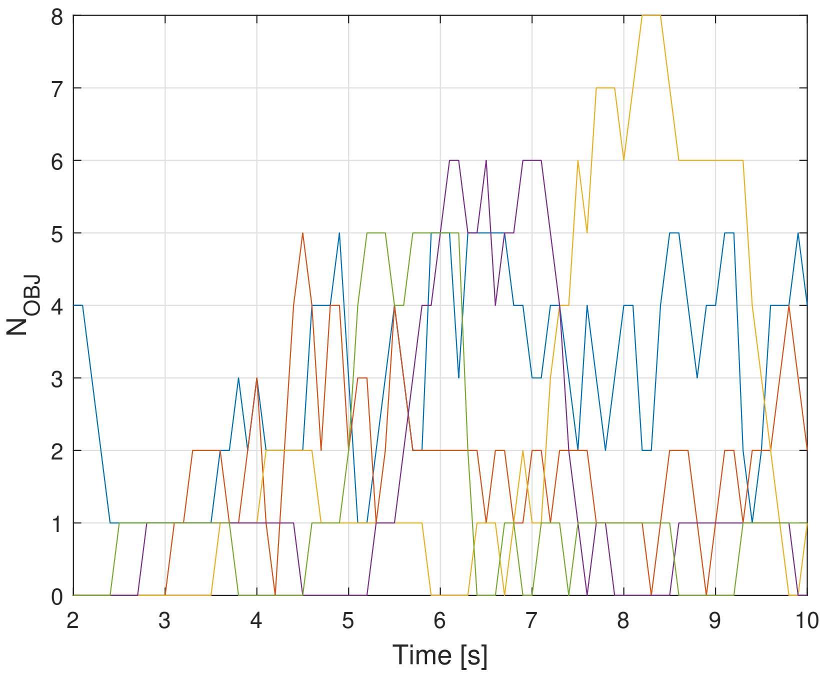


Figure 5.18. Number of stars history for the 5 °/s (blue), 7 °/s (orange), 10 °/s (yellow), 13 °/s (purple) and 15 °/s (green) cases and added noise sources ([0,0,1] angular rate direction).

mine said, everything is contained in the phenomenon equations and through them I achieved the goal. The simulated scenario sees the sensor of Table 5.1 and the algorithm configuration of Table 5.2 working in a 10 second sequence with 100 ms of exposure time. The angular rate module is 5 °/s with a direction parallel to [1,1,1]. The pause time between an image acquisition and the following one is kept equal to 0 ms without any distancing from reality. For this tests no noise was applied to focus on the RSOs resolving criterion feasibility analysis and validation.

Figure 5.19 shows a merging of first 6 images of the sequence. The simulated RSO is clearly visible and is in the FOV since the first frame untill frame 6.

The algorithm is capable of both tracking the star motions and the RSO. Once the implemented resolving strategy is applied, the HAR estimates and RSOs centroids evolution can be retrieved. They are reported in Tables 5.13 and 5.14. The estimate final angle and norm deviations are: $\Delta\alpha = 0.72\,°$ and $\Delta_{norm\%} = 4.22\,\%$.

Table 5.13. HAR components errors mean values and standard deviations for RSO-Stars scenario.

$\|\vec{\omega}_{ref}\|$	$\overline{\Delta\omega_x}$	$\overline{\Delta\omega_y}$	$\overline{\Delta\omega_z}$	$\sigma_{\Delta\omega_x}$	$\sigma_{\Delta\omega_y}$	$\sigma_{\Delta\omega_z}$
5 °/s	0.075 °/s	0.14 °/s	0.16 °/s	0.0041 °/s	0.00048 °/s	0.00075 °/s

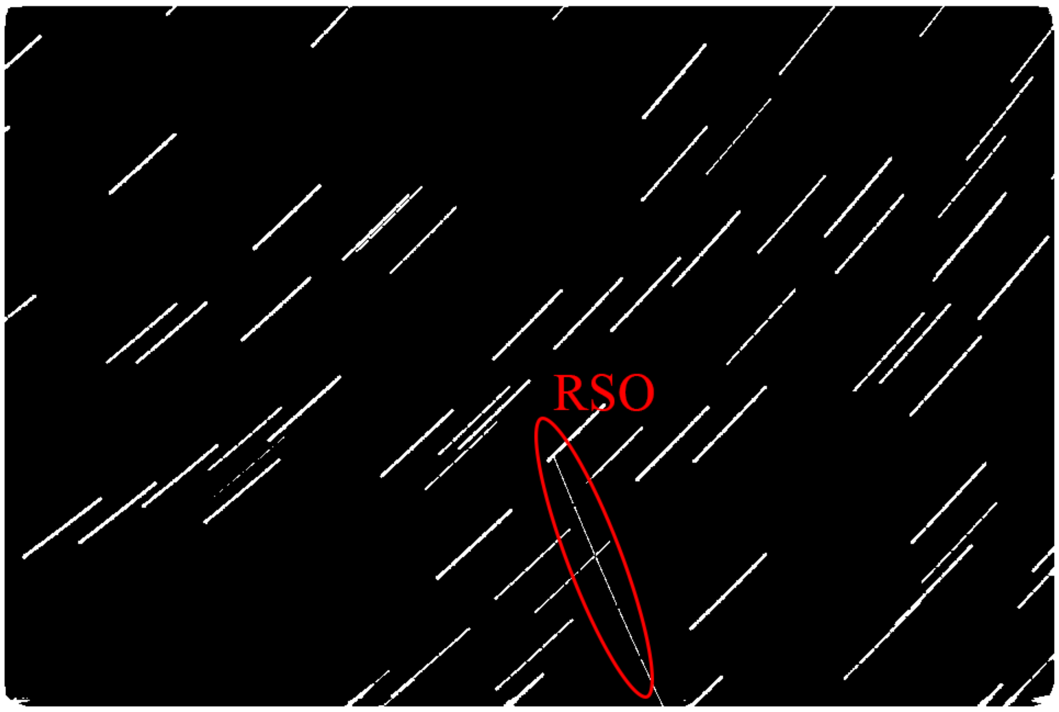


Figure 5.19. RSO's passage with the rotating sensor. Partial merging of the 10 seconds sequence.

Table 5.14. RSO's centroids evolution.

Frame	1	2	3	4	5	6
X_{RSO} [px]	491.72	510.42	530.15	548.94	568.71	591.05
Y_{RSO} [px]	385.88	432.02	479.07	523.12	568.33	616.74

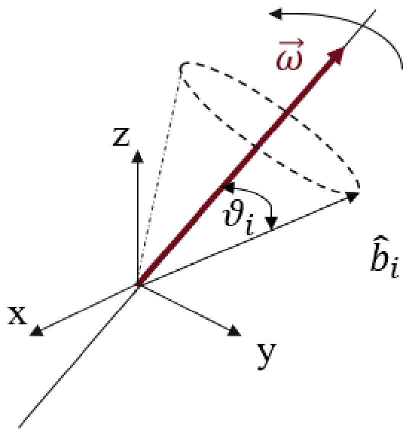


Figure 5.20. Star unit vector time evolution in HAR presence.

5.3.1 RSOs-Star Resolving Method

As said before, the solution is contained in the phenomenon equations. For a star in the rotating sensor's FOV, the star unit vector will obey to the kinematics constraint Equation 5.1. Thus, under the knowledge of the HAR estimate and unit vector evolution is then possible to evaluate the module of the Left Hand Side (LHS) and Right Hand Side (RHS). The sinus computation of the costant angle between the star unit vector and the angular velocity is then straightforward (Figure 5.20):

$$\left|\frac{d}{dt}\hat{b_i}\right| = \omega \cdot |sin(\theta_i)| \tag{5.16}$$

From the star unit vector and angular velocity scalar product is then possible to retrieve the cosinus of the θ_i angle:

$$|\vec{\omega} \cdot \hat{b_i}| = \omega \cdot |cos(\theta_i)| \tag{5.17}$$

In the end, the following equation needs to be satisfied for each star object:

$$|sin(\theta_i)|^2 + |cos(\theta_i)|^2 = 1 \tag{5.18}$$

If now a RSO is considered, its centroids evolution will not satisfy the Equation 5.16 because the RSO motion is not compatible with a rigid rotational motion having that sensor $\vec{\omega}$. Thus, by evaluating sinus and cosinus with equations 5.16 and 5.17 for a RSO, the goniometry identity (Equation 5.18) will not be satisfied anymore.

These considerations have lead me to define the following "RSO" metric:

$$f_i = ||sin(\theta_i)|^2 + |cos(\theta_i)|^2 - 1| \tag{5.19}$$

Where:

$$\begin{cases} f_i = 0 \; for \; moving \; stars \\ f_i \neq 0 \; for \; RSOs \end{cases} \tag{5.20}$$

This way it is possible to resolve RSOs from stars. At each time, with the HAR estimate, the f-index for each object can be evaluated. Due to centroids estimate errors the f-index will be really close to 0 for star-like tracked objects while it will be high enough for RSOs. By evaluating the mean f-index value for each object over the 10 seconds period, a distribution of f-indices is obtained. This distribution will have a mean value M_f and a standard deviation σ_f:

$$\begin{cases} M_f = \frac{\sum_{i=1}^{N} f_i}{N} \\ \sigma_f = \sqrt{\frac{\sum_{i=1}^{N}(f_i - M_f)^2}{N-1}} \end{cases} \tag{5.21}$$

If $f_i \geq M_f + 3 \cdot \sigma_f$, then the i-th object will be classified as RSO. Otherwise it will be considered as a star.

In Figure 5.21, the f-indices trend is shown together with the threshold and the resolved RSO for the described test. Two star-like objects have a slightly higher f

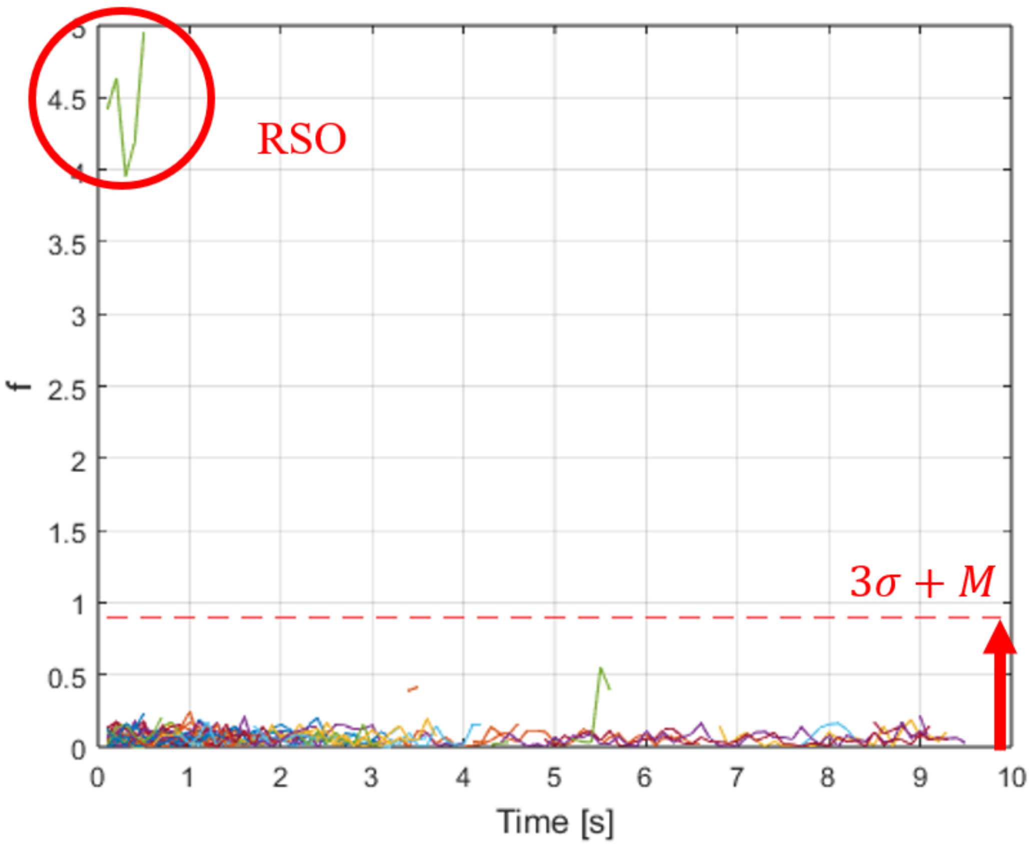


Figure 5.21. f-indices trends over the 10 sec time interval with the threshold. The unique RSO is far away from 0 and easily resolved.

value because of tracking errors which are related to the used cost function thresholds. Anyway, the 95 % of the star-like objects is confined below the dashed red line. Thus the probability for a star-like object to overcome the threshold is low and acceptable for this preliminary study.

5.3.2 RSOs-Star Resolving Work-Flow

Test and results have been shown for this mixed RSO-HAR scenario and the method to resolve RSOs from stars has been described. Here, the work-flow strategy for this dual-purpose streaks detection and tracking algorithm is provided for HAR rotating sensors and RSOs presence in the FOV:

1. Image is received in input;
2. Segmentation, Detection and Tracking are performed;
3. Tracking list is refined with filters;
4. HAR module is applied both using stars' and RSOs' information;
5. RSOs are resolved with the f-index strategy;
6. HAR estimate is produced again with only stars information.

Then the HAR info is sent to the on-board Navigation computer while the RSO information needs for further corrections before being sent to storage memory for being downlinked. Actually, the RSOs centroids estimates need to be corrected for the platform HAR motion. This is something that has not been investigated in this work yet, but it will be done in the future.

5.4 Summary of Findings for the Dual-Purpose

The dual-purpose of the AI-based streaks segmentation, detection and tracking algorithm has been investigated. Contribution of this chapter are:

1. Extension of the AI-based chain for HAR estimate purposes;
2. Extension of an angular rate estimator based on least square approach to HAR scenarios;
3. Extensive tests based on HFSTS images which prove satisfying HAR estimates both in noisy conditions and with RSOs;
4. Analysis of the initial attitude and angular rate effects on the number of tracked stars;
5. Analysis of performances degradation and correlation with the preliminary AI-based algorithm limits;
6. Provision of a Kinematics-based strategy for resolving RSOs and stars;
7. Realization of a starting point algorithm for AI-based Star Sensors.

About performances degradation, the random initial attitude and actual angular speed effects can't be controlled. From an algorithm limits point of view there are some roads to explore in order to overcome these limits:

- Streaks' centroids estimates errors: A streak shape reconstruction module could be investigated to improve the accuracy in centroids estimate;
- Streaks Tracking errors: These are related to the used threshold for the tracking cost function, which can lead to the uncorrect tracking of clear and close streaks having similar orientation and motion. Different tracking strategies could be investigated;
- Undetected Point-like moving stars: A point detection and tracking module can be implemented to track stars when the camera integration time is not high enough to appreciate the stars' streaks. YOLO-v4 could be trained in a two class problem (points and streaks) or a classical point-detection algorithm can be used in parallel with the YOLO to get point-like stars information and fuse the outputs together. This way the ideal range of operations of the angular rate estimator could be extended in the [0°/s ; 15°/s] range due to the capability in point-like objects tracking and the following amount of information for angular rate estimations.

About RSOs' centroids estimates correction for HAR effects, this is something that it will be investigated in my future research activity.

In the end, tests have shown that this preliminary concept of the dual purpose algorithm, is capable of providing an estimate with a deviation angle and norm less than 10° and 10 % respectively. Despite they do not come from a Monte Carlo simulation (this work is being investigated at the moment), these errors are suitable for a first rough estimation of the HAR in case of spacecraft in SAFE mode. They can be used to consecutively slow down the spacecraft at acceptable rates through a succession of finite maneuvers. Then, at a lower angular rates, it is possible to retrieve more accurate estimates to perform ordinary maneuvers and the complete recovery of the spacecraft by the mission control center operators.

Chapter 6

Design and Development of a Dual-Purpose AI-based Autonomous Star Sensor for Attitude Navigation and Space Surveillance: TRIDAENT

The idea of a dual-purpose on-board payload for attitude and SST purposes is not something so far anymore. Trend in the next future of space missions is not just having platforms that achieve the mission goals but using them as space debris sentries too. This is needed both for the platform own safety and for the safety of all the other active missions in the near Earth space segment. STs have proven to be the suitable candidates for such tasks. Several studies and applications were performed and my PhD activity shows that this is possible even with the latest software technologies such the AI. What I achieved is a starting point to develop and design the next generation of dual-purpose STs. Actually, through an efficient extraction of complex information from the ST's image I proved that it is possible to track both RSOs and stars, it is possible to resolve them and use this information for SST and SN tasks. This brought me to the idea of designing a dual-purpose ST which implements the presented AI-based algorithm: TRacking IDentification and Angular rate Estimation NeTwork star tracker (TRIDAENT).

TRIDAENT couples the developed algorithms with classical routines for what concerns LISA and attitude estimation processes, combining the AI capabilities in image filtering with long heritage flight proven routines.

Results of the above mentioned design will be a ST which provides real time information of platform attitude, angular rate and RSOs with the orbit navigation information because it will be integrated with a Global Positioning System (GPS) sensor. This way, RSOs time evolution data will be coupled with platform navigation data to perform RSOs catalogue update through ground based orbit determination nowadays (on-board based in the next future who knows?).

6.1 TRIDAENT Design

The basic hypothesis to build this product is having a module capable of providing in real time:

- Stars' centroids evolution;
- RSOs' centroids evoultion.

With stars information, the LISA and attitude determination routines can be performed while the RSOs will be stored to be down-linked to the Earth. Thus, the goal is to filter and track stars and RSOs in an image while removing the noise. A top-down approach will be followed to identify requirements and low level architecture for dual-purpose product. This is required to achieve the TRIDAENTs' capability in identifying and tracking both points and streaks. In the end, a suitable method to classify RSOs and stars will be required. Thus, the work-flow is:

1. Developing a points and streaks detector and tracker module;
2. Developing a method to classify stars and RSOs from initial points/streaks data;
3. Including the LISA and attitude determination algorithms in the process chain.

After having achieved these steps, the Software (SW) core of TRIDAENT is completed. Its high level HardWare (HW)/SW architecture is presented in Figure 6.1:

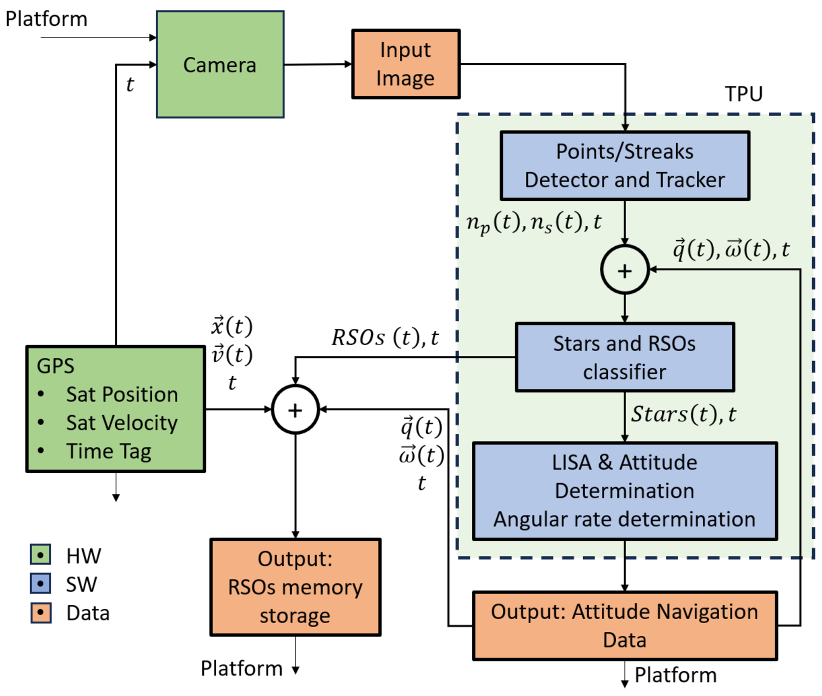

Platform
Camera
Input Image
TPU
t
Points/Streaks Detector and Tracker
$n_p(t), n_s(t), t$
$\vec{q}(t), \vec{\omega}(t), t$
+
$\vec{x}(t)$
$\vec{v}(t)$
t
$RSOs\ (t), t$
Stars and RSOs classifier
GPS
• Sat Position
• Sat Velocity
• Time Tag
+
$\vec{q}(t)$
$\vec{\omega}(t)$
t
$Stars(t), t$
LISA & Attitude Determination Angular rate determination
HW
SW
Data
Output: RSOs memory storage
Output: Attitude Navigation Data
Platform
Platform

Figure 6.1. TRIDAENT high level HW/SW architecture.

The Input image data is provided from the optical camera which is synchronized with GPS through time information provision. The camera works at fixed exposure time and cycle (typically at 1 Hz, 2 Hz, 5 Hz up to 10 Hz). The Input image is segmented and objects (point/streaks) are detected. Then each object is tracked (*Points/Streaks Detector and Tracker*) and their time evolution is used by the *Stars and RSOs classifier* to solve stars and RSOs. This module can work both if a priori attitude information (quaternion $\vec{q}(t)\,, \vec{\omega}(t)$) is available or not. The classifier sends RSOs information to the TRIDAENT memory storage while the stars time evolution is sent to the attitude determination module (or to LISA if no attitude information is a priori available). This high level architecture system can receive commands from the platform and it is capable of retrieving the desired attitude and RSOs information. Figure 6.1 shows the TRIDAENT Processing Unit (TPU) which hosts SW and is in charge to handle data flows and to check the respect of processes sequence.

6.1.1 Points/Streaks Detection & Tracking Module

This module is in charge of distinguishing and tracking points and streaks. A two classes detection and tracking action can be performed by using one of the two proposed solutions:

- a U-Net for the image segmentation process with a two classes trained YOLOv4 detector and the minimization-based tracking function;
- a two classes trained recent YOLO v7 coupled with BlendMask.

The former has been developed and analyzed for single class detection in this research activity while the latter is based on YOLO v7 [110] and it will be investigated in the next future due to the network's recent publication. YOLO v7 is capable of performing both objects detection and tracking. Moreover, if compared with YOLOv4, the v7 reduces the number of parameters by 75%, requires 36% less computation, and achieves 1.5% higher AP (average precision) [111]. Finally, it can be coupled with BlendMask [112] obtaining the YOLO v7-mask [111] which could segment the night sky images, detect and track points and streaks. The YOLO v7-mask seems to be the next evolution of my actual research activity, by creating ad-hoc datasets to training the BlendMask and the YOLO v7 and performing on the field Verification & Validation tests.

6.1.2 Stars and RSOs Classifier

For mapping points and streaks into stars and RSOs, we need to take into account not only their shape but their evolution too. Shape recognition and evolution tracking is possible through the proposed solutions (previous section). This approach combines the use of shapes and evolution bringing to an algorithm which retrieves stars and RSOs.

As can be seen in Figure 6.1, this algorithm receives:

- points' centroids evolutions ($n_p(t)$);

- streaks' centroids evolutions ($n_s(t)$);
- platform's quaternion ($\vec{q}(t)$);
- platform's angular rate ($\vec{\omega}(t)$).

The *Star and RSOs Classifier* needs the $n_p(t)$ and $n_s(t)$ as mandatory input data to produce some outputs. Besides the mandatory data, the auxiliary data $\vec{q}(t)$ and $\vec{\omega}(t)$ can simplify the RSOs and stars classification. I identified five different input data scenarios for this module (time information t is omitted because it is considered always available):

1. $n_p(t)$, $n_s(t)$, $\vec{q}(t)$ and $\vec{\omega}(t)$ availability;
2. $n_p(t)$, $n_s(t)$ and $\vec{q}(t)$ availability;
3. $n_p(t)$, $n_s(t)$ and $\vec{\omega}(t)$ availability;
4. No data availability;
5. $n_p(t)$ and $n_s(t)$ availability;

In the first scenario, both $\vec{q}(t)$ and $\vec{\omega}(t)$ will be used to resolve stars from RSOs. Actually, if the quaternion $\vec{q}(t)$ is available then the tracked stars are known and can be removed from the tracked objects. This *quaternion filter* is then followed by the *angular rate filter* which uses the $\vec{\omega}(t)$ information to filter again the objects whose evolution obeys to Equation 5.1. The angular rate filters takes the unit vector evolution $\frac{d\hat{b}(t)}{dt}$, $\hat{b}(t)$, computes the f-metric and applies the condition 5.20. So, with the sequential application of both quaternion and angular rate filters is then possible to resolve stars and RSOs.

In the second scenario the quaternion filter is performed only and every object that is not filtered out is considered a RSO. The same situation happens in the third scenario with the angular rate filter that replace the quaternion filter, but the same logic applies. The fourth scenario is straightforward and the algorithm will wait for the next input data. All these cases are summarized in Figure 6.2.

The most interesting scenario is the last: no attitude information is available and some assumptions need to be done in order to develop a classification method. It must be pointed out that: point does not inevitably mean star and streak does not inevitably mean RSO because stars shape varies according to the sensor attitude evolution:

- Low angular rates: points could be stars and noise while streaks could be RSOs and noise;
- Medium-high angular rates: points could be stars and noise while streaks could be stars, RSOs and noise;

Noise is something which can be detected by following its irregular evolution (single frame noise) and does not constitute a problem since it can be easily filtered out (section 4.2.2). Below, the assumptions I did to proceed in developing the *Objects only method* are described:

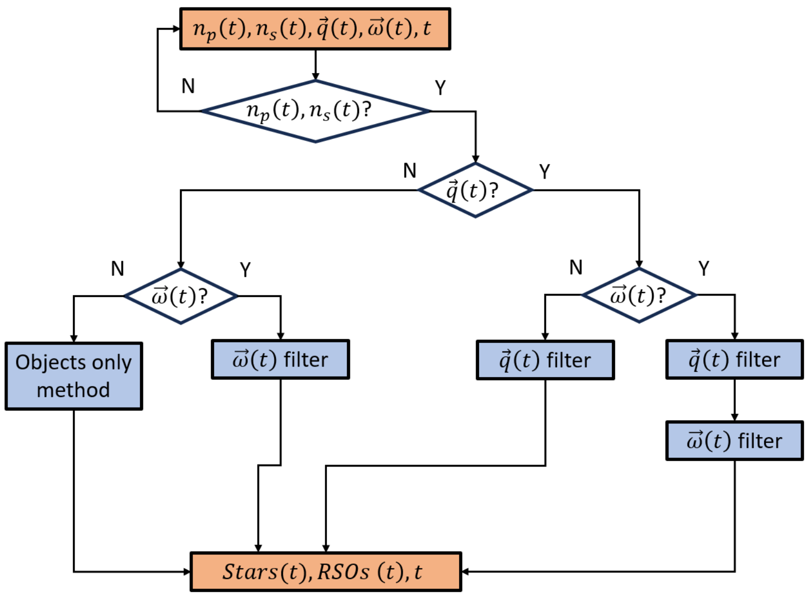


Figure 6.2. TRIDAENT Star and RSOs Classifier architecture.

- Low angular rates: points are stars and streaks are RSOs;
- Medium-High angular rates: "points" are stars and streaks could be both stars and RSOs.

In order to better understand why the existence of "points" in medium-high rates ($\omega \geq 1.5$ °/s), the ST exposure time must be taken into account. The exposure time could be not enough for stars resulting in long streaks and to be classified as streaks. Thus as "points" in this sections I mean both proper points and very short streaks (thickness comparable to the streak length).

With these premises, at every frame several points n_p and streaks n_s will be available for the RSOs and stars classification through the *Objects only method.*

This module work-flow is shown in Figure 6.3 and aims to solve the fifth scenario. Its input information is the time evolution of points ($n_p(t)$) and streaks($n_s(t)$).

If points' centroids evolution is available, then they are assumed to be stars and a LISA is performed to retrieve the platform attitude. In case of success, a quaternion filter is applied and stars can be separated by RSOs among the points and streaks. They will be provided in output together with the quaternion auxiliary information. If LISA fails, then a check on the amount of points w.r.t. the streaks is done. If the number of points is greater than the number of streaks (one order of magnitude) the algorithm checks for points and streaks information in the next frame. If the numbers of points and streaks are comparable, then streaks are assumed to be stars and with the streaks information the HAR (section 5.1.1) is computed. With the angular rate, the objects can be separated in stars and RSOs and the HAR computation is performed again with only stars information untill the number of stars and resolved RSOs does not change over the time. From here, a cycle exit is performed and the RSOs and stars are resolved with a further computation of

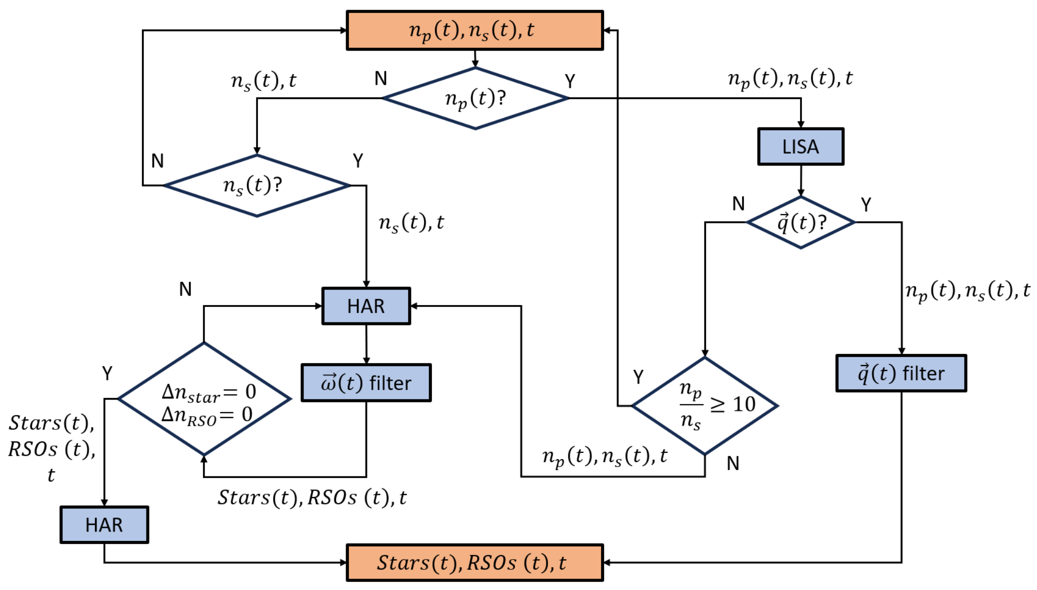


Figure 6.3. TRIDAENT's Objects only method work-flow.

HAR to produce the latest estimate of angular rate (auxiliary information). Starting again from the beginning, if no point evolution is available, then a check on streaks availability is performed. In case of no input data available the algorithm checks again the incoming input information. If streaks are the only tracked objects, then the process foresees the HAR estimation, RSOs and stars separation and HAR estimate refinement.

To summarize this process:

- If points are available, they are assumed to be stars;
- An attitude computation is performed to solve stars and RSOs;
- If this is not successful, then streaks are considered stars, HAR computation is performed to solve stars and RSOs;
- In case of only input streaks the direct angular rate computation is performed to solve stars and streaks.

With this process the *Object only method* can provide stars' and RSOs' time evolutions. Auxiliary information of quaternion and angular rate may appear among the outputs. All this information is then separated:

- RSOs are sent to the on-board storage memory;
- Stars, a priori attitude and angular rate are sent to the LISA and Attitude Determination, Angular rate estimation routines to perform the ordinary ST operations (Figure 2.7).

6.1.3 LISA, Attitude and Angular Rate Determination

LISA and attitude determination routines purposes is described in Section 2.1.2 while the angular rate determination module is presented in 5.1.1. Classical LISA and attitude determination algorithms are many and flight proven. For TRIDAENT project it will be necessary to select one of the available methods. For LISA, famous algorithms are: TRIAD [22], Planar Triangle algorithm [23], Multi-Poles algorithm [24] etc... For attitude determination the QUEST [113] algorithm works fine, but also the Singular Value Decomposition (SVD) [114] method may be applied. The design of this module is not a critical aspect for TRIDAENT development because the collected heritage is relevant and offers a wide panorama of tested routines.

6.2 TRIDAENT Development Planning: From Preliminary Design to IOV Mission Proposal

A preliminary idea and design for the TRIDAENT star sensor has been presented in the previous sections. High level architecture and a deeper look to the work-flow show the areas and actions to be performed to realize the final space product. From my experience as Co-PI and Documentation Manager of a small satellite payload project concerning SSA (Section 7), I can identify and write down a hypothetical development steps list to achieve a more concrete design for TRIDAENT. Summary of the main steps to carry on is:

- YOLOv4 Training and Test for points and streaks detection;
- YOLO v7 and Blend Mask solution development, Training and Test.
- Comparison of these solutions and Selection of the best architecture and performance;
- Verification and Validation (V&V) for the Points/Streaks Detection & Tracking module;
- SW implementation of the Objects only method work-flow;
- SW implementation of the full Stars and RSOs classifier;
- V&V of the Stars and RSOs Classifier
- V&V of the Points/Streaks Detection & Tracking integration with Stars and RSOs Classifier;
- Selection of the LISA, Attitude Determination routines;
- V&V of the entire TRIDAENT SW chain;
- Optical sensors selection;
- In-the-Loop communication protocol implementation and Interfaces development;

- V&V of the camera and implemented SW integration;
- Integration of the GPS sensor for the closed-loop final system;
- V&V of the entire HW/SW Proof of Concept: the Engineering Model (EM);
- Spatialization of the HW/SW systems;
- Thermo Vacuum, Vibration and Radiation Tests to meet European Cooperation for Space Standardization (ECSS) expectations;
- In the loop Documentation maintenance and completion and Flight Model (FM) realization;
- IOV mission proposal design and Call for Interest for space companies/agencies to have the TRIDAENT product integrated on their platforms.

These steps mark the road to transform this research activity into a real space product and space mission in the next few years. I precise that it is a hypothetical list, because it does not take into account of the real problems that in general occur in the development of a space project. Often these issues require some review processes, request for changes and deviations, bringing the original linear development path to several reiterations or evolution in something different.

IOV mission proposal

Previous results have proved that it is possible to detect, track, resolve RSOs and stars streaks in an efficient way for RSOs and AN purposes. Moreover, the steps to achieve a final real product have been identified.

In this section the preliminary hypothetical design of a SBSS mission orbit to validate the TRIDAENT concept is presented.

- The purpose: Detection of RSOs populations in the LEO region through a dual-purpose ST which offers a reliable AN support to the hosting platform.
- The orbit: RSOs spatial and mass distribution are strictly related to the former space activities and the collisions and fragmentation of orbiting objects. Several studies have described and modeled the debris evolution during the years, showing that LEO is the most densely populated region around the Earth, especially in 600-800 km altitudes with high inclinations [115]. For this reason, the onboard SST mission would be focused on LEO debris detection at a 600-800 km altitude range, and the orbit selection would be constrained by the sun illumination. To guarantee the correct detection functioning, the optical sensor (on-board camera or star sensor) cannot be oriented against the Sun. Figure 6.4 shows possible sun-illumination conditions. In particular, $\hat{d}_{s_i t_i}$ represents the target directions, with s_i denoting the observer satellite and t_i one of the debris elements. Satellite s_2 cannot detect target t_2 because it is backlit, while the sun illumination allows for a correct detection of target t_1 by satellite s_1. Another limiting factor to be considered is the presence of the Earth in the FOV, because of the consequent reduced FOV of the optical sensor. To guarantee the visibility of the target by the observer, the vector

$\hat{d}_{s_i t_i}$ cannot cross the Earth. Finally, to ensure the same sun-illumination condition during the mission, the best candidate orbits for object detection are the Sun-Synchronous Orbits (SSOs). In particular, the dawn-dusk orbits can ensure a continuous and constant utilization of the optical sensors, while the noon/midnight orbits are not recommended for this kind of application due to the possibility of having eclipses. Due to all these considerations, the preliminary selected orbit for the mission would be a circular dawn-dusk SSO with a height of 700 km.

- The Validation Process: The satellite would be equipped with an attitude control system (ACS) capable of orienting TRIDAENT ST's bore-sight direction in the along-track direction and anti-Sun direction to maximize the time the RSOs spend inside the FOV. Ground commands would be sent to the platform to schedule the pointing operation to switch between these two targeted directions. Several input images would be stored together with the corresponding masks and outputs for validation purposes during a test phase. Once on the ground, the raw images would be processed and compared with the onboard produced masks and outputs. The platform would be equipped with an independent attitude determination module to compare the payload outputs in terms of quaternions and angular rate estimates (from gyroscopes) to complete the TRIDAENT validation process. Moreover, slew and high rate maneuvers would be performed to test TRIDAENT's angular rate estimation capability.

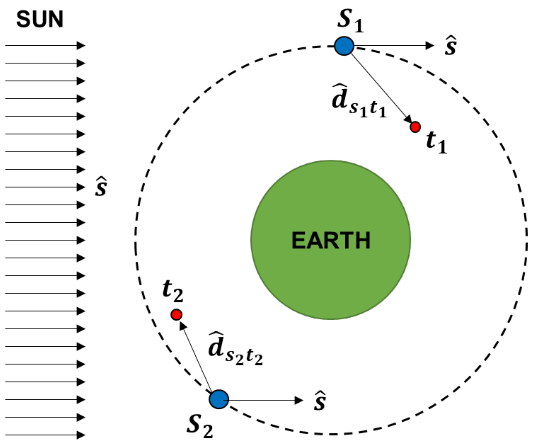


Figure 6.4. Visibility conditions.

The mission concept that has been described is hypothetical and comes from my matured experience in the above mentioned project. It may seem too much detailed for a preliminary IOV mission concept but not so far from real missions [6] in terms of sun-synchronous orbit selection.

Chapter 7

SPOT: Star sensor image Processing for orbiting Objects deTection

7.1 Mission Goals

This part of my final work is devoted to a parallel project I carried out during my PhD activity. It is named SPOT and is a School of Aerospace Engineering/Italian Space Agency project which is related to the SSA application involving STs. It is funded by Italian Space Agency and I am the Co-PI of the on-board part of the project under the supervision of the PI Prof. Fabio Curti at School of Aerospace Engineering untill the end of my PhD activity. I worked in this project both as System Engineer and Documentation Manager, leading the project through the System Requirements Review, Preliminary Design Review and Critical Design Review. The matured experience in this context gave me the idea to push this application towards the AI world creating an improved version of it. Results and experiences I gained with it were essential to search the right way to use the AI for streaks detection and tracking in my research activity. Moreover, the achieved experience was crucial for the preliminary idea and design of my TRIDAENT project.

The SPOT project involves me and other two young researchers of my department, in addition to my supervisor and Italian Space Agency personnel. This project provides an innovative space-based autonomous and versatile solution for RSOs' optical detection via star sensors and for different Earth orbits scenarios. This system is being designed and it is part of the IOV activity in 2024. The SPOT algorithms involve many complex arithmetic operations, which are relatively expensive in terms of computational latency, limiting their applicability to real-time image processing applications. Actually, I am the Co-PI of this project: I started both as Documentation Manager and Algorithm Designer for the on-board payload. I gained enough knowledge with ESA's standards to develop the project and face project reviews as the System Requirements Review (SRR), Preliminary Design Review (PDR) and Critical Design Review (CDR) with our customer Italian Space Agency. My Algorithm Designer activity was about the designing choices of the algorithm work-flow, payload budgets (storage memory, power, mass, volume), and mission constrains (exposure time, orbit, duty-cycle etc..). I was in charge of monitoring the SPOT SW/HW implementation process to materialize the developed MATLAB routines into an End-to-End system. I was in charge also of leading the night sky acquisition campaigns to acquire enough materials for SPOT payload Verification & Validation with real images data.

The SPOT system is essentially divided into two main units. The On-Board SPOT and Ground SPOT.

7.1.1 On-Board SPOT Unit

Figure 7.1 shows the On-Board SPOT unit architecture. The required inputs are the image (expressed as an $M \times N$ matrix) and the epoch (expressed, for instance, as an integer for the J2000 date and a double for the fractional part). On-board SPOT input are STs images, and it can work with single images or with a continuous flow. For every single image, the *Pre-processing* module oversees the image segmentation task. If the image is already segmented by ST processor this step will be avoided. The *Pre-processing* module returns as output the foreground segments in the image. These segments are then processed by *Clustering* module. This module takes the foreground pixel segments belonging to the mask as input and returns the clusters. Each cluster is characterized with its own centroid information and estimated energy data together with extremal points information. Once that at least two consecutive images are processed, clusters belonging to the first and second image must be associated to track the movement of every detected signal at the image couple level. This task is delegated to *Cluster Fusion* module. Its output are the fused data, i.e., the centroids, extremal points, and the estimated magnitudes in both images of the tracked clusters. Then the information from the *Cluster Fusion* are filtered by the *Antitracking* module which is in charge of removing the stars if the platform quaternion information is known. In the end the remaining non star objects will be tracked by the *Cluster Growth* module and organized for the downlink to the ground stations.

A deeper description of each module is presented below.

Pre-Processing

The raw image from the ST is processed by the *Pre-processing* module. The high-energy pixels belonging to stars' or RSOs' signals must be distinguished from the low-energy pixels relative to noise. Hence, the *Pre-processing* module selects the pixels whose signal-to-noise ratio is greater than a user-defined threshold and discards the useless information. In particular, this pre-processing is called *Segmentation* and it is based on a dynamic approach (in particular LTS) for the background noise estimation consisting in the comparison of the signal of each pixel with the average signal of its local neighborhood. Pixels are considered for further processing when their values are greater than the background noise (BKG_0) plus a threshold (τ_{pre}). Therefore, only segment information is considered for further analysis relative to objects' detection. In particular, the *Pre-processing* outputs are the pixels' coordinates of each detected foreground segment $\mathbf{p} = [p_y, p_z]^T$ and their energy $E(\mathbf{p})$, obtained subtracting the background noise to the signal intensity of the pixel.

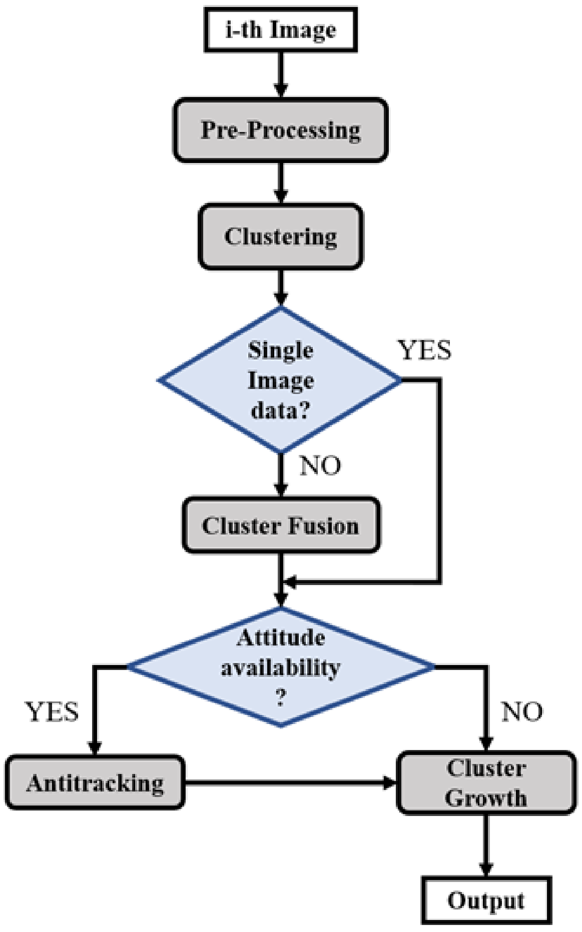


Figure 7.1. On-Board SPOT architecture.

Clustering

The *Clustering* module has to recognize all the neighboring pixels belonging to individual objects. Usually, star sensors carry out a simple clustering algorithm (called here as *Primitive Clustering*) but it is not enough when dealing with fast objects. A *"primitive cluster"* is defined as groups of pixels which share at least one corner, as shown in Figure 7.2, while single pixels are mostly related to noise or too faint objects and they are automatically discarded and not considered for the following analyses.

The green pixels in the figure share an edge or a corner and are associated to the same cluster (named "Cluster1"). The same can be said for the orange pixels ("Cluster2"), while the red ones are discarded representing noise or too faint objects.

Under dynamic conditions, objects in the FOV can generate broken streaks. It means that pixels associated to the same object can be spreaded into several small and distinct clusters. For this reason, a suitable technique has been developed in order to relate to the same object (star or RSO) the different primitive clusters of the

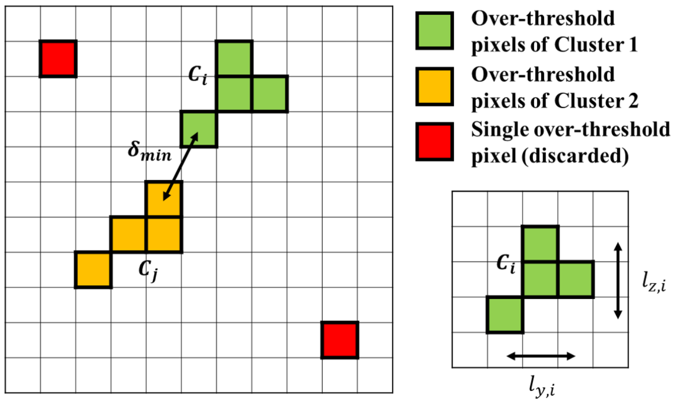

C_i
δ_{min}
C_j
Over-threshold pixels of Cluster 1
Over-threshold pixels of Cluster 2
Single over-threshold pixel (discarded)
C_i
$l_{z,i}$
$l_{y,i}$

Figure 7.2. Example of two primitive clusters and discarded single pixels.

broken streak. It is important during this operation to avoid wrong primitive clusters agglomeration coming from noise or wrong matching between different objects. This technique is called *Improved Clustering* and it is based on three filters ([116]):

- *Minimum distance filter*: The distance between clusters δ_{min} is evaluated as minimum pixel-by-pixel distance using the uniform norm and must be lower than a user-defined threshold ε_{dist} (Figure 7.2). The minimum distance condition is defined as:
$$\delta_{min} \leq \varepsilon_{dist} \tag{7.1}$$
- *Increasing length filter*: $\mathbf{l}_i = [l_{y,i}, l_{z,i}]$ is the length vector of the i^{th} Cluster (C_i), collecting the horizontal and vertical projections of the cluster. To merge the clusters C_i and C_j, the merged cluster C_ij must satisfy precise conditions related to $\mathbf{l}_i$, $\mathbf{l}_j$ and $\mathbf{l}_{ij}$;
- *Density filter*: Clusters C_i and C_j can be merged only if some necessary conditions related to their densities are satisfied. In particular, the density of the i^{th} cluster is defined as:
$$d_i = \frac{N_i}{\lambda_i} \tag{7.2}$$
where, N_i is the number of pixels of C_i and λ_i represents a virtual length of the cluster. Clusters with low density values are likely to be broken into different pieces.

At this point, some primitive clusters have been merged and the cluster centroids are computed using the pixel coordinates $\mathbf{p} = [p_y, p_z]^T \in C_i$ throughout an energy-weighted average:
$$\mathbf{c}_i = \frac{\sum_{j|\mathbf{p}_j \in C_i} \mathbf{E}(\mathbf{p}_j) \cdot \mathbf{p}_j}{\sum_{j|\mathbf{p}_j \in C_i} \mathbf{E}(\mathbf{p}_j)} \tag{7.3}$$
where $\mathbf{E}(\mathbf{p}_j)$ is the signal intensity of the pixel $\mathbf{p}_j$.

Cluster Fusion

To compare two successive images and merge the objects which are supposed to be the same in the two frames, the *Cluster Fusion* module was developed. This module is designed to work only if two consecutive images are available, otherwise it is skipped (Figure 7.1). It cannot be applied if a non-negligible time interval separates the first and second image. The required operations are very similar to the ones described for the Improved Clustering. The main difference is that the clusters C_i and C_j which are compared belong to two different images. The minimum distance filter and the density filter are applied. The output of the *Cluster Fusion* does not contain not-merged clusters, i.e. clusters that do not appear in both the successive images. In this way, it is possible to recognize the persistence of an object in the frames and to allow the software to understand the direction of the moving object in the FOV.

Antitracking

The *Antitracking* module is required in order to remove stars from the set of detected objects. This module requires the current attitude quaternion, a Hipparcos Catalogue (max magnitude 5.5) and the sensor's characteristics to be set as input. It is an operation very similar to the usual star tracking operation with the difference that all the objects recognized as stars are filtered out. *Antitracking* removes stars according to the following steps:

1. Evaluate image plane centroids in 3D camera frame (CRF UVs). This operation requires sensors' characteristics.
2. Evaluate the centroids PRF UVs in the inertial frame using the current quaternion and MM.
3. Compare the PRF UVs in the inertial frame with catalogue stars UVs.
4. Remove centroids which report angular distances from catalogued stars smaller than an user defined threshold. The adopted threshold depends on quaternion accuracy.

Cluster Growth

The *Cluster Growth* algorithm is used to identify and track objects in the star sensor's FOV. This module is applied to build an on-board database of tracked objects (stars or RSOs). Cluster Growth has to compare and analyse successive couples of images. The goal is to track RSOs from their appearance in the FOV until they leave it and save their positions and time instants of each couple of images. The output of this module represents the final output of On-Board SPOT unit. The on-board data structure is transmitted on ground for post-processing operations and orbit determination from too short arcs.

In this framework short RSOs observations are called "Too Short Arcs" (TSAs) observations, and they do not allow for a converge of classical Initial Orbit Determination (IOD) algorithms (Gauss, Laplace, etc.). The definition of TSAs, within this work, refers to those observations with a total acquisition time interval lower than: 30 seconds (Very Short Arcs), 3 minutes (Short Arcs).

At the initialization of the *Cluster Growth*, the merged clusters returned by the *Cluster Fusion* are saved as independent objects. When a new couple of images is analysed and the fusion operation is performed, the new merged clusters are compared with the previous objects and if a cluster is recognized as belonging to an already detected object, it is updated with the upcoming couples. When there is no correspondence, after a maximum number of couples defined by the user-defined parameter N_{jump}, no update occurs. On the contrary, merged clusters not recognized as belonging to previous objects lead to the creation of a new object in the on-board database.

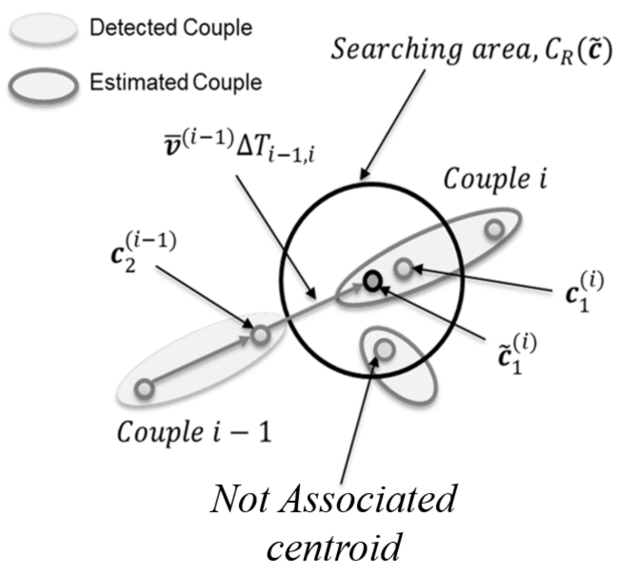


Figure 7.3. Cluster Growth algorithm.

The filtering operation is based on three separate steps:

- *Position filter*: The core of the algorithm is based on the estimation of the centroids' positions in the new couple of images for each tracked objects. The distance-based criterion of the *Cluster Fusion* must be extended to associate the merged clusters of the first and second couples. Indeed, the merged clusters are not as close as in the case of the *Cluster Fusion* and an estimate of the cluster position in the successive couple of images is required. Using velocity estimations computed as a first order finite difference during the fusion operation up to the $(i-1)^{th}$ couple of images, the average velocity $\bar{\mathbf{v}}^{(i-1)}$ is computed considering the number of available images N_a. With this information and the last saved position of the centroid, the estimation of the centroid position $\tilde{c}_1^{(i)}$ in the first image of the i^{th} couple is evaluated when $c_2^{(i-1)}$ and the displacement $\bar{\mathbf{v}}^{(i-1)}\,\Delta T_{i-1,i}$ are considered. The centroid search is performed if and only if $\tilde{c} \in D$, where D is a circular neighboring area of the centroid estimation, called *Searching Area* (Figure 7.3). If one or more detected centroids fall inside D, they are selected as candidates for the tracked object.

- *Velocity filter*: The position filter represents a first preliminary selection of feasible candidate centroids, but more than one centroid is likely to fall inside the searching area. The velocity filter refines the previous results by comparing the velocity of candidate centroids with the velocity of the tracked object. The filter considers both the direction and the magnitude of the velocities. By defining ϕ as the angle between the two directions of motion, the objects characterized by angles greater than a user-defined threshold ϕ^* are discarded.
- *Confirmation phase*: If more than one centroid has been detected by the previous filters, a criterion to choose the best candidate must be accounted for. To assess which is the most reliable centroid, the estimates of the cluster density and centroid displacements are included in a last filter based on a minimization criterion. Actually, a function F is introduced:

$$F = F_{dist} + F_{dens} \tag{7.4}$$

  where, F_{dist} is related to the difference between the estimated displacement of the centroid and the calculated displacement and F_{dens} is based on the object density and it is evaluated as a normalized difference. The candidate with the lowest value of F is chosen.

At this point, the centroids are associated to the considered object and the on-board database is updated [7].

The On-Board system can operate in two modes:

1. Active Mode: if the quaternion information associated to the generic image is available;
2. Passive Mode: if the quaternion information associated to the generic image is not available.

If SPOT is operating in Active Mode, the *Antitracking* module is activated to remove stars from the set of detected objects. This module requires the current image quaternion as input.

Then, regardless of the mode, *Cluster Growth* module is the last module of the chain. It takes the fused data from cluster fusion module (eventually filtered by *Antitracking*), the epochs and data from the RSOs and the previously stored RSO as inputs. In this manner it is possible to associate the detected RSOs to already existing ones inside the FOV and keep on tracking the moving objects.

7.1.2 Ground SPOT Unit

The architecture of Ground SPOT is reported in 7.4. Several modules are linked and cooperate:

- Ground Database Software (GDS)

- ORbit Determination Software (ORDS)
- Objects Tracking Software (OTS)
- Conjunction Analysis Software (CAS)

The main purposes of these modules are the collection and organization of data from On-Board SPOT into a ground database, the successive orbit determination for the detected RSOs, the propagation and tracking of the objects' trajectories and, finally, the computation of the collision probability for each couple of objects in the SPOT catalogue.

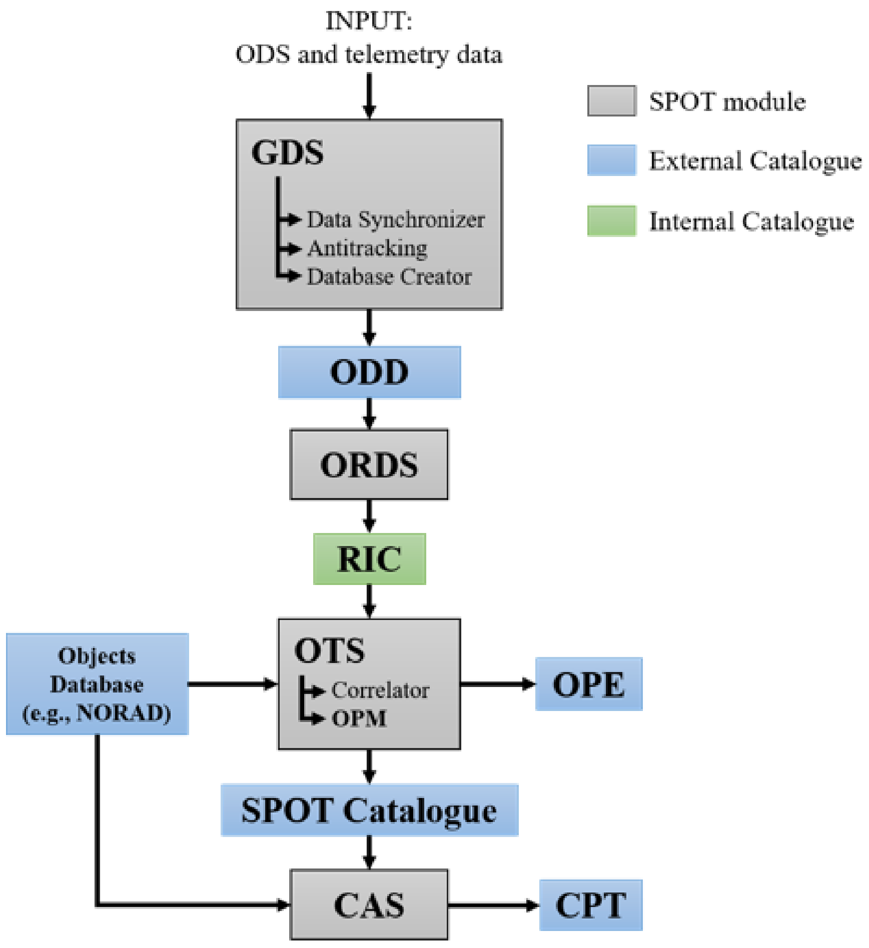


Figure 7.4. Ground SPOT architecture.

GDS

GDS is the first ground module. It has to organize the information coming from the ground station (On-Board SPOT data and telemetry) into a database called Object Detection Database (ODD). The input data are processed by the Data Synchronizer sub-module. It evaluates all the available telemetry data by using spline interpolation at the epoch of detected RSOs. Then, the ground *Antitracking* module removes stars leaving only RSOs by means of a star catalogue reduced to magnitude 6.5. The synchronized data are then used as input by the database creator module. It collects the data related to the same object and the corresponding values for the interpolated telemetry data. For each detected object, RSO's position in Earth Centered Inertial reference frame (ECI), RSO's magnitude, observer's position, velocity, quaternion and angular velocity are collected along with the corresponding epoch expressed in Universal Time Coordinated (UTC). As explained before, the observer data are synchronized by interpolation from the available data from the satellite operator.

ORDS

ORDS module performs the IOD for each RSO provided in the ODD. The module is also designed to evaluate the dimension of the object. The module takes data information from ODD as input. The output returned by this module is collected in an internal catalogue called RIC (RSOs Identification Catalogue) needed by OTS and CAS modules. RIC contains information relative to the Classical Orbital Elements (COE) of the estimated orbit, along with dimension estimate for the corresponding RSO. Since the orbiting platform and the RSO are in relative motion between each other, the observation time may be limited (TSAs). This is a typical case in LEO missions where the relative dynamics may be fast. Thus, to solve IOD problem from TSAs, AI techniques are being investigated. ML (supervised learning) and Metaheuristics algorithms are considered for their possible application to the problem.

OTS

OTS should perform two main tasks. The first one is responsible for the SPOT catalogue generation, in which the observed objects' orbit estimates are correlated and collected. Orbit-to-Catalogue (O2C) and Orbit-to-Orbit (O2O) correlations are performed:

- O2C compares obtained COEs from ORDS with the catalogue COEs (e.g., NORAD);
- O2O compares and correlates orbits (COEs) from different catalogues.

The module should also provide an uncertainty estimation associated to the orbit estimate coming from ORDS. Actually, the ORDS provides a first guess for the orbit, then the OTS is in charge of the correlation of observations related to the same RSO which is detected multiple times in different time intervals. The obtained longer observation permit to improve the orbit estimate and to compute the associated (reduced) uncertainty. This is the reason why the OTS provides the definitive orbit estimation uncertainty. To do this, OTS uses filters (e.g., batch filter) to statistically characterize the prediction reliability. The output structure is called SPOT catalogue and it will contain all the elements of the Two Line Elements (TLEs) adding to them magnitude, dimension, spinning rate (Tumbling), and the observer's position and velocity. The second task that needs to be accomplished by OTS is relative to the propagation of the RSO to estimate the next useful observation, for the same observer or a different one, orbiting or Earth based. This sub-module is called Object Pointing Module (OPM). This function will require the observer position and the type (i.e., Orbiting or Earth based) as inputs. The output will be provided as the temporal evolution of Azimuth, Elevation (or Right Ascension and Declination), and range depending on the epoch. The output data structure is called Object Pointing Elements (OPE).

CAS

CAS module evaluates the collision risk between a target selected by the user and an object collected in the SPOT Catalogue. The Collision Probability (CP) estimation is used to decide an orbital correction maneuver for the target. This module estimates the Time of Closest Approach (TCA), the Distance of Closest Approach (DCA) and the CP considering couples of detected objects collected in the SPOT Catalogue. CAS is divided into two main submodules:

1. No-collision Prediction Algorithm (NPA): it is used to determine which RSOs surely do not collide with the selected target.
2. Probability of Collision Algorithm (PCA): it permits to estimate the CP P_c of the objects that will have a close approach with the target.

The module will perform the operations mentioned above for all the objects in the catalogue provided by the OTS module, calculating the CP for each objects couple. The output of this module will be the Collision Probability Table (CPT).

7.1.3 On-Board SPOT Feasibility Tests

The conducted tests are based both on the images from an observation campaign of Italian Space Agency Matera Observatory and simulated images for testing the On-Board SPOT system with fictitious small objects using a HFSTS.

On-Board SPOT Unit Test with Real Images

The goal was testing On-Board SPOT software using real images of the night sky with satellites in the FOV. GEO satellites have been observed using inertial pointing of the telescope, providing a realistic scenario in order to simulate an on-board application of SPOT. The GEO satellites in the FOV appear as moving streaks in consecutive images. On-Board SPOT can process the received images to detect objects (both RSOs and stars). Only foreground pixels are considered as useful information and merged in clusters according to some specific filters (length, distance and density filters). Clusters of consecutive images are compared to assess which of them belong to the same object. In this way, useless information is filtered to reduce the amount of data that will be sent on ground.

Input Image Data Description

The analyzed images were provided to Sapienza School of Aerospace Engineering by Italian Space Agency Matera Observatory. They have been taken with an OS RiFast 400 telescope and processed via TheSkyX Version 10.5.0 Build 11086. The detector was a ProLine PL16803. Their characteristics are showed in Tables 7.1 and 7.2. Despite the moving satellites, the fixed stars pattern on the background is showed in Figure 7.5:

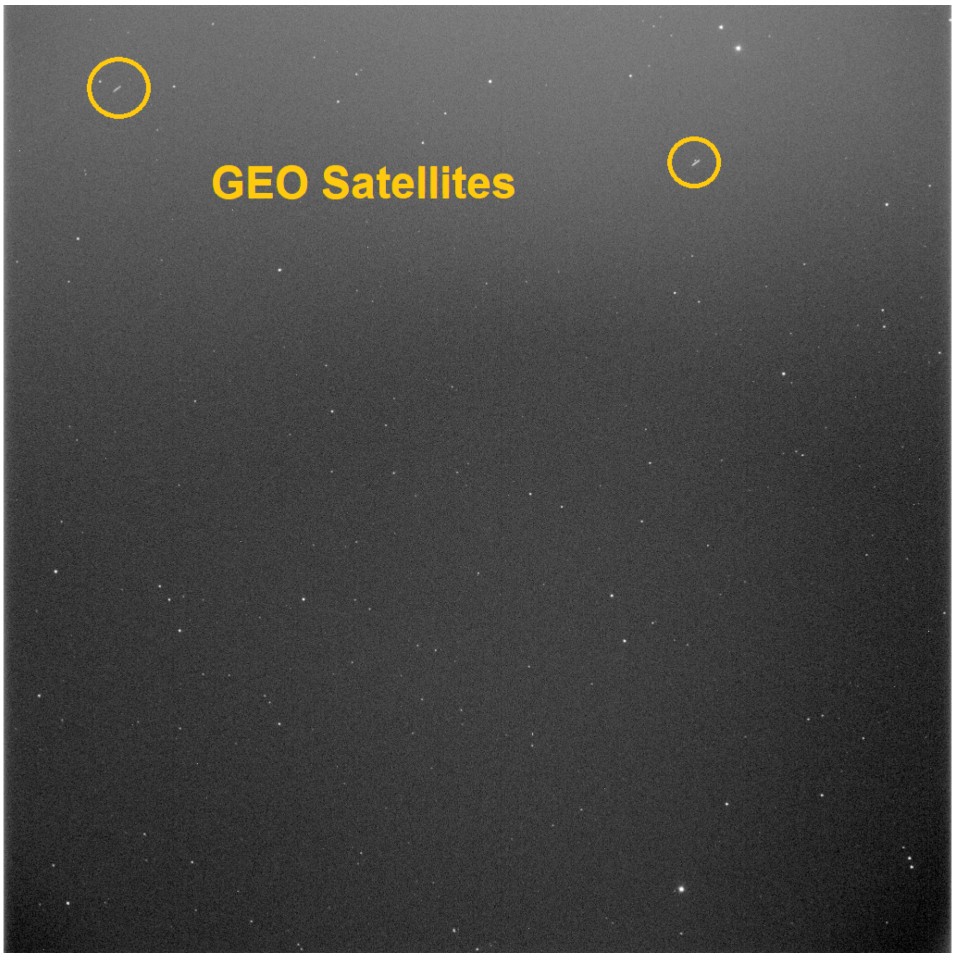


Figure 7.5. Image sample from ASI Matera Observatory containing two GEO satellites.

The satellites are the two streaks moving from top to bottom in the upper part of the image. It must be noticed that there is a sensible straylight coming from the lower part that affects the whole image.

From the stars pattern on the background, stars identification and attitude determination were possible. The only modification applied to the algorithm was updating the Star catalogue up to max magnitude 11 to adapt the code to the ground telescope observer performances. Thus, the attitude information was available for the On-Board SPOT unit. Just five stars of the Hipparcos catalogue are within the FOV and are showed in the following image Figure 7.6.

Identified Hipparcos Catalogue stars were the following:

1. HIP 13231 Mg 7.4;
2. HIP 13309 Mg 9.7;
3. HIP 13334 Mg 9.1;

Table 7.1. Detector Specification.

Parameter	Value
Image Size	4096×4096
Pixel Size	$9\,\mu m$
Binning	2×2
Horizontal FOV	1.39°
Diagonal FOV	1.96°
Focal length	$1520mm$
Pointing	Sidereal

Table 7.2. Image specification from .fit file.

Parameter	Value
Acquisition Epoch	$20:17:16.071\ UTC$
Acquisition Date	23/10/2020
Latitude (Acquisition Site)	40°38' 58.04" N
Longitude (Acquisition Site)	16°42' 12.10" E
Exposure Time	$3\,s$
bore-sight Direction Pointing (RA)	+123.921°
bore-sight Direction Pointing (DEC)	+23.996°

4. HIP 13616 Mg 9.9;

5. HIP 13476 Mg 7.2.

They were used for quaternion computation via Singular Value Decomposition (SVD) method. With this quaternion information it was possible to perform the *Antitracking* module. Each centroid (x,y) that reaches this module is used to compute the associate UV in the ECI or PRF frame $\hat{v}_{ECI}$. From this information, the scalar product with every UVs of the Hipparcos catalogue stars $\hat{v}_{HIP}$ is computed. By knowing of this scalar product, it is possible to compare this value with a certain tolerance.

For every Hipparcos star catalogue element a check is performed:

$$\hat{v}_{ECI} \cdot \hat{v}_{HIP} > cos(tol) \tag{7.5}$$

where the tolerance that is chosen is equal to $tol = \frac{1}{100}\,\theta_{FOV}$ (diagonal FOV). This angle is equal to 15 pixels of the 1024 × 1024 image. Condition 7.5 is the rationale behind the *Antitracking* module of SPOT to filter clusters associated to Hipparcos stars. This operation is made for every cluster, a circle of radius equal to to is built around each of them and if a star from the Hipparcos star catalogue is in this searching area, it is removed.

The image coordinates (px) of the previous and removed stars were (data from On-Board SPOT MATLAB implementation):

1. $[793.3; 47.3]$;

2. $[842.2; 398.9]$;

3. $[525.1; 83.1]$;

4. $[56.0; 612.4]$;

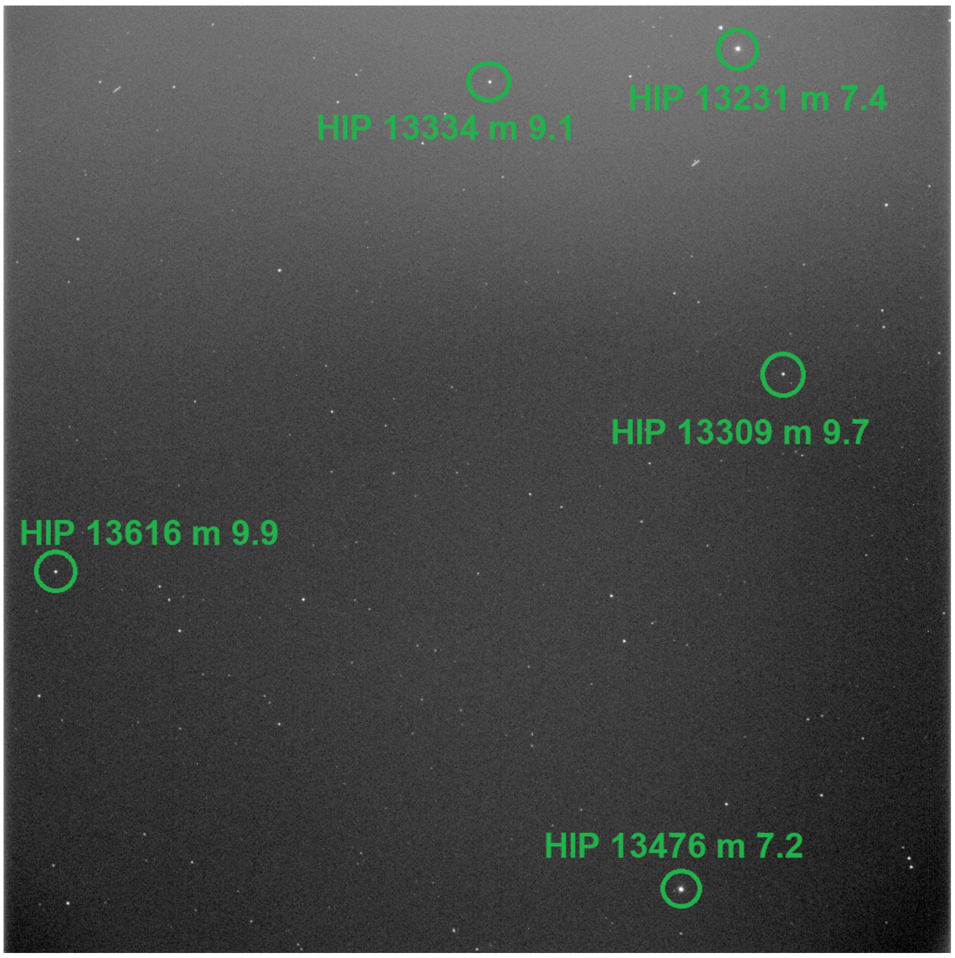


Figure 7.6. Star Pattern Recognition performed on the Matera image.

5. [731.8; 955.6].

These stars were successfully removed by On-Board SPOT *Antitracking* module because they are part of the Hipparcos Catalogue.

After the whole system run, On-Board SPOT MATLAB implementation was able to track satellites and not removed stars. They are shown in the .txt file screenshot in Figure 7.8.

The previous stars image coordinates are referred to the real objects inside the FOV and to a star of another catalogue (GAIA, not used by STs generally). Observer's position, observer's velocity, and angular velocity in the ECI and relative epochs have not been included to provide a reduced and clearer presentation of the outputs. The complete structure of these outputs is shown in Table 7.3.

Their content is composed of RSOs main data:

1. ST ID: Star Tracker ID;

2. RSO Number: Number of the Resident Space Object;

3. Image Number: Number of the Image;

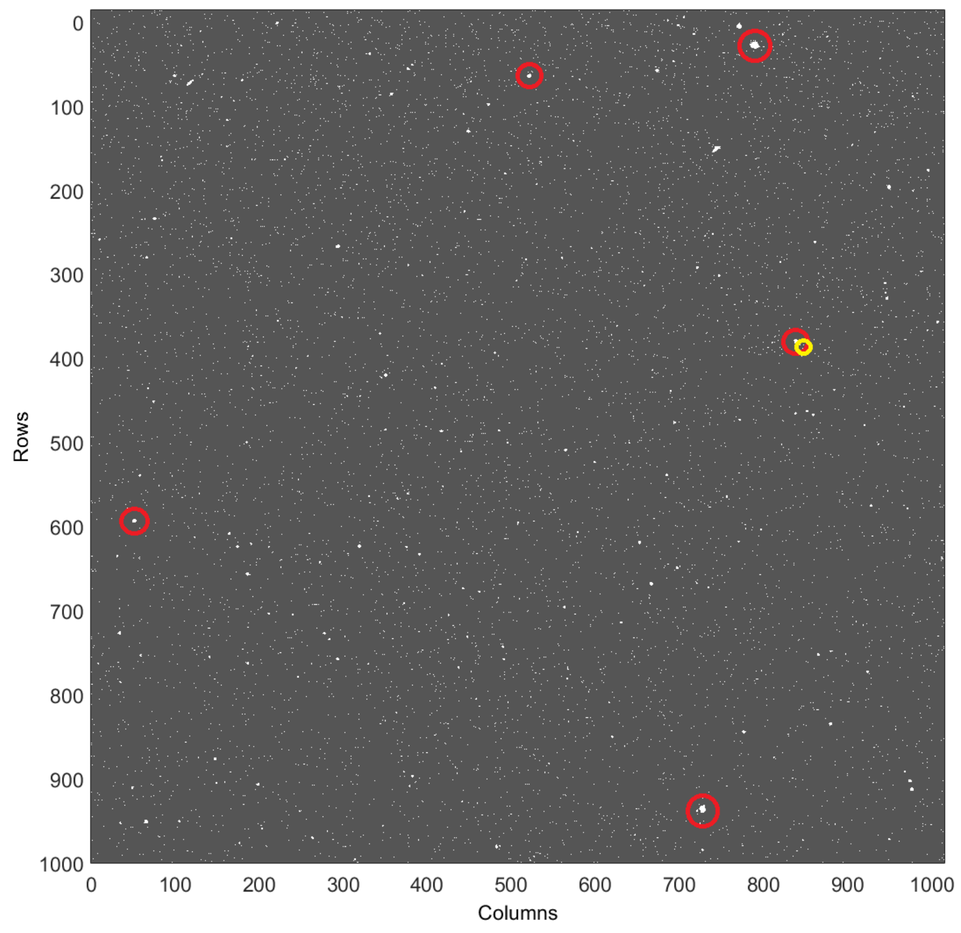
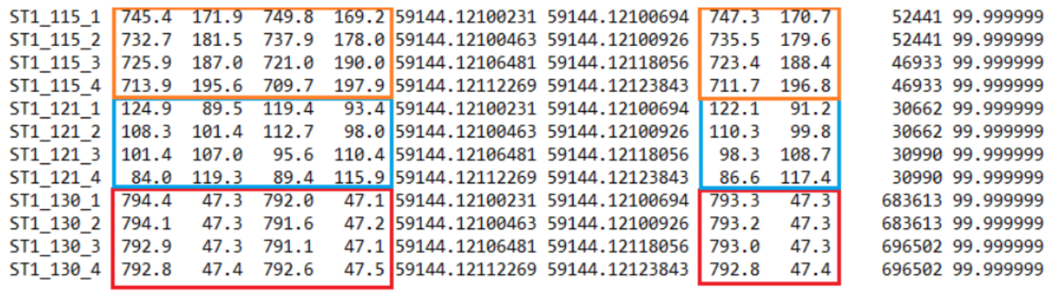


Figure 7.7. MATLAB pre-processed Image with removed stars (red) and the object (yellow).

4. Two Extremal Points' (EPs): EPs' coordinates;
5. Two EPs' Epochs expressed in Modified Julian Date (MJD): MJD format for the epochs associated to the EPs;
6. Centroid: centroid coordinates;
7. RSO energy: Energy associated to the RSO.

Ancillary Data:

1. Observer's quaternion: Platform quaternion;
2. Observer's quaternion epoch expressed in MJD: Platform quaternion epoch;
3. Observer's Position: ECI position of the platform;
4. Observer's Position Epoch expressed in MJD: Platform's position epoch;
5. Observer's Velocity: ECI velocity of the platform;



```
ST1_115_1 745.4  171.9  749.8  169.2 59144.12100231 59144.12100694 747.3  170.7      52441 99.999999
ST1_115_2 732.7  181.5  737.9  178.0 59144.12100463 59144.12100926 735.5  179.6      52441 99.999999
ST1_115_3 725.9  187.0  721.0  190.0 59144.12106481 59144.12118056 723.4  188.4      46933 99.999999
ST1_115_4 713.9  195.6  709.7  197.9 59144.12112269 59144.12123843 711.7  196.8      46933 99.999999
ST1_121_1 124.9   89.5  119.4   93.4 59144.12100231 59144.12100694 122.1   91.2      30662 99.999999
ST1_121_2 108.3  101.4  112.7   98.0 59144.12100463 59144.12100926 110.3   99.8      30662 99.999999
ST1_121_3 101.4  107.0   95.6  110.4 59144.12106481 59144.12118056  98.3  108.7      30990 99.999999
ST1_121_4  84.0  119.3   89.4  115.9 59144.12112269 59144.12123843  86.6  117.4      30990 99.999999
ST1_130_1 794.4   47.3  792.0   47.1 59144.12100231 59144.12100694 793.3   47.3     683613 99.999999
ST1_130_2 794.1   47.3  791.6   47.2 59144.12100463 59144.12100926 793.2   47.3     683613 99.999999
ST1_130_3 792.9   47.3  791.1   47.1 59144.12106481 59144.12118056 793.0   47.3     696502 99.999999
ST1_130_4 792.8   47.4  792.6   47.5 59144.12112269 59144.12123843 792.8   47.4     696502 99.999999
```

Figure 7.8. On-Board SPOT partial output format: Satellites (orange and pale blue boxes) and star (red box).

6. Observer's Velocity Epoch expressed in MJD: Platform's velocity epoch;
7. Observer's Angular Rate: Angular velocity of the platform in ECI;
8. Observer's Angular Rate Epoch expressed in MJD: Epoch of the Angular velocity of the platform.

What is shown in Figure 7.8 is the format of the On-Board SPOT outputs related to the two GEO satellites (N_{RSO}= 115 and 121) and a fixed cluster inside the FOV (N_{RSO}= 130). It can be seen from centroids pixel coordinates that for the object 130 the coordinates are stationary due to the sidereal tracking (at least the unit digit varies due to noise). This means that object 130 has not been filtered by the antitracking because it is not comprised in the Hipparcos catalogue. For the objects 115 and 121, image coordinates are not stationary with the image number and this indicates their RSO nature.

Table 7.3. On-Board SPOT File format (first part above and second part below).

ST_ID_NRSO_Nimg	EP_1	EP_2	$Epoch_{start}$	$Epoch_{end}$	Centroid	Energy	$\vec{q}$
$\vec{q}_{epoch}$	$\vec{x}$	$\vec{x}_{epoch}$	$\vec{v}$	$\vec{v}_{epoch}$	$\vec{\omega}$	$\vec{\omega}_{epoch}$	

On-Board SPOT Unit Test with Simulated Images

In this test the capability of SPOT system in detecting objects of 5 cm in radius is tested. The observer platform is a LEO circular orbit with h=400 km. The ST is oriented towards the radial direction. Different contiguity times intervals and pause analysis are investigated in terms of On-Board SPOT working. At initial simulation time, 4 spherical objects of 5 cm in radius at 20 km of bore-sight distance are initialized inside the FOV. Their orbits are chosen with respect to several constraints among which:

- Semimajor axis $\leq$ 42164 km;
- Perigee height $\geq$ not lower than 200 km.

Two couple of images have been taken into account. They are enough to prove the working or not of On-Board SPOT modules. Actually, because of their proximity with respect to the observer after few seconds of simulation every object disappears in FOV so, the simulation of an entire orbit is unnecessary. For every object the following data have been reported:

- True centroid position of detected clusters on the image;
- Estimated centroid position of detected clusters on the image;
- Energy of detected clusters.

In this way the correct working of *Pre-processing*, *Clustering*, *Cluster Fusion* and *Cluster Growth* can be proved.

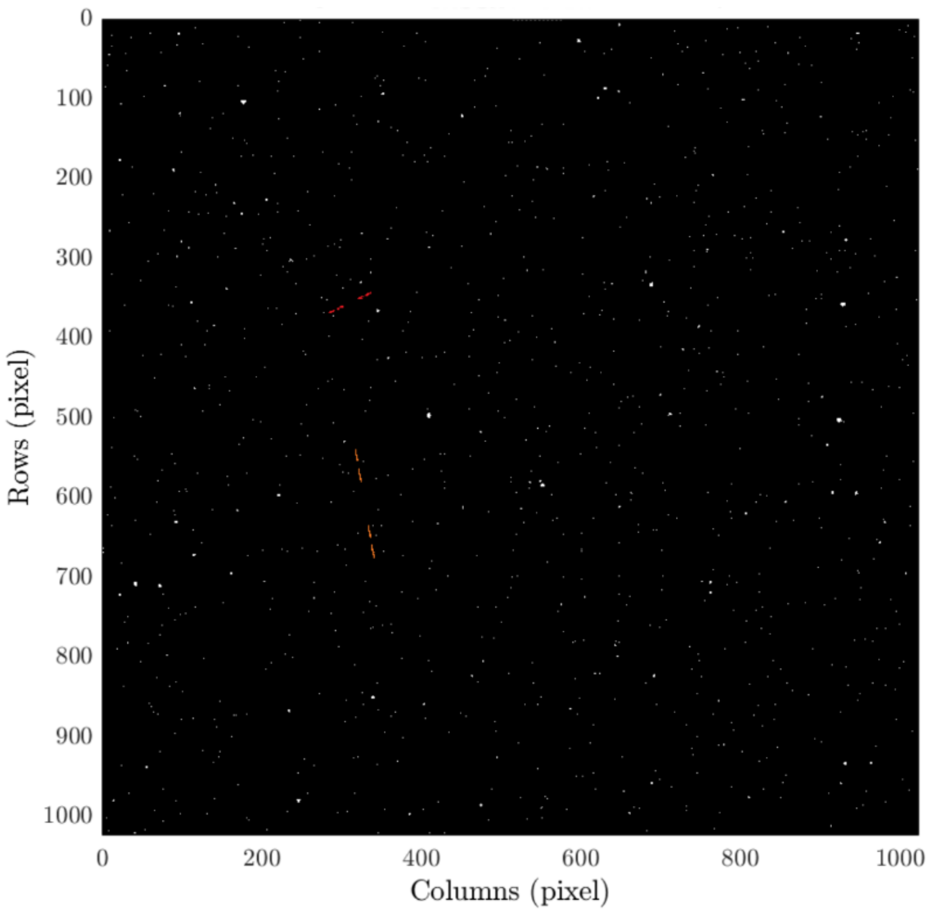


Figure 7.9. Time evolution of the detected objects inside the FOV.

In Figure 7.9 the time evolution of the objects within the FOV for all the consecutive 4 images is shown. Of the initialized 4 objects just two of them are detected. This is due to the different orbit initial conditions and not due to a different phase angle (light illumination is the same for each of them) or particular attitude of the object itself (they are spherical). By looking at the image, it is possible to appreciate the nonexistent pause between consecutive images ($\delta t = 0\,ms$) within every couple and the pause between a couple of them and the following couple ($\Delta t = 600\,ms$) while the exposure time is $T_{exp} = 200\,ms$. The acquisition philosophy with the roles of δt and Δt are better explained by Figure 7.10.

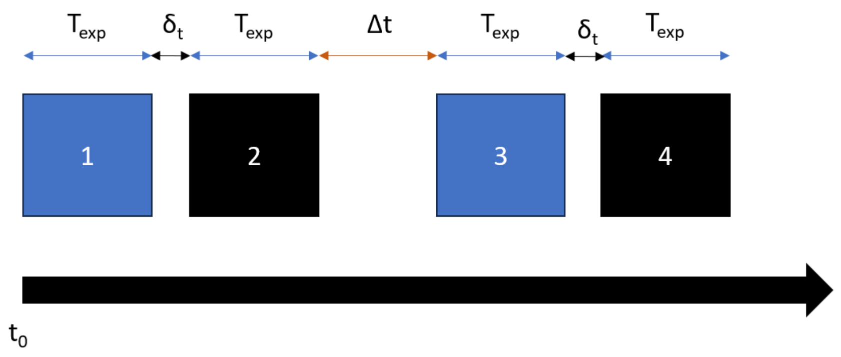


Figure 7.10. Image acquisition timing philosophy.

Table 7.4. Simulated Images test results: Energies evolution.

	$Image1$	$Image2$	$Image3$	$Image4$
Energy Obj1 $[e^-]$	1303	1418	1402	1410
Energy Obj2 $[e^-]$	1243	1394	1410	1320

7.1.4 On-Board SPOT Hardware Implementation

This section shows the software/hardware implementation of SPOT modules on a Zynq-7000 SoC that integrates a dual-core ARM Cortex-A9 processor associated with an FPGA (Table 7.6). This represents a further step for this ambitious project because a hardware implementation is needed for on-board application and for proving real time working of the algorithm. The complexity of the SPOT algorithm is due to a combination of ordinary image processing and unusual data elaboration with extensive use of mathematics. This requires the selection of a space-qualified board capable of speeding up calculations with affordable power consumption and real-time requirements. Therefore, the Z7000-P3 Onboard Computer (OBC) was selected based on its ability to perform heavy calculations in real-time. This OBC is based on Zynq SoC technology and for development purposes, it was better to use a cheaper computer with the same technology level. Actually, a development board from the same family called ZYBO (ZYnq BOard) was selected. It is a digital circuit development platform built around Zynq-7000 SoC that can host a whole system design. The on-board memories, USB, SD slot, and six Pmod ports, make designing of the final flight software SPOT up-and-ready, putting it on an easy growth path. Figure 7.11 shows the general system configuration where ZYBO is connected both to the camera to get the image stream and to the computer for commands and data transmission.

The co-design approach was used for SPOT software/hardware integration: it makes software routines and hardware work together to share the processing load. It enables, moreover, high-speed image flow processing by increasing the execution speed of SPOT algorithms approximately by 70 times with respect to the MATLAB implemented solution (results are shown below).

In co-design methodology the design flow was divided across the two implementation platforms, FPGA and microprocessor with the intention of benefiting from each of their strengths to share the processing load:

Table 7.5. Simulated Images test results: True (T) and Estimated (E) Centroids comparison.

	$Image\,1$		$Image\,2$		$Image\,3$		$Image\,4$	
	$Col[px]$	$Row[px]$	$Col[px]$	$Row[px]$	$Col[px]$	$Row[px]$	$Col[px]$	$Row[px]$
T Obj1	324.1	581.1	321.1	563.4	309.1	492.6	306.1	474.9
E Obj1	324.1	581.2	321.2	563.6	309.3	492.5	306.2	475.0
T Obj2	300.7	361.2	294.2	364.4	268.2	377.2	261.7	380.4
E Obj2	300.8	361.2	294.1	364.5	268.3	377.3	261.6	380.3

Table 7.6. ZYBO's FPGA specification.

Features	Values
Look-Up-Tables	53200
Flip-Flops	106400
Block RAM	630 kB

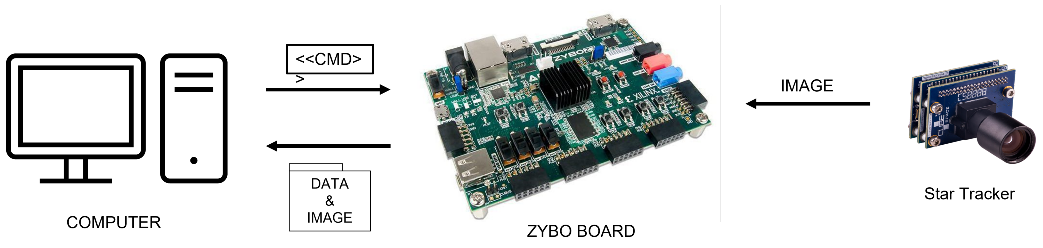


Figure 7.11. General System with ZYBO

- The parallelism nature of the FPGAs enables the capability to run several processing elements in parallel. This makes possible to process the data during few clock periods. This feature makes FPGAs suitable for numerous high-performance applications that require intensive and high-speed computations like image processing;
- Microprocessors perform well with managing and controlling data as well as decision making.

In this case, the microprocessor is used as a "master" configuration unit to direct the flow of data to the "slave" FPGA device. The "master" is called Processing System (PS) and besides the microprocessor, it contains multiple controllers including an off-chip memory controller to store the images and data into DDR SDRAM (Dual Data Rate Synchronous Dynamic Random Access). The "slave" or Programmable Logic (PL) contains not only the FPGA but also a wide range of resources: Block Random Access Memory (BRAM) and Digital Signal Processing (DSP) blocks which are mainly used for SPOT design. PL and PS are connected by several interfaces. The interface that plays a key role in accelerating image and data transfer to avoid SPOT processing overhead is named AXI (Advanced eXtensible Interface) and is shown in Figure 7.12. AXI can work as memory-mapped or stream. AXI-Stream is used for high data transfer to send images from PS to PL, while Memory-mapped AXI (AXI-Lite) is used to set/read-back the configuration parameters and to send/receive commands and flags to or from PL by reading and writing operations on the associated memory address.

In next lines the FPGA implementation of SPOT routines will be highlighted hiding the micro-controller programming which goes beyond the scopes of this research activity.

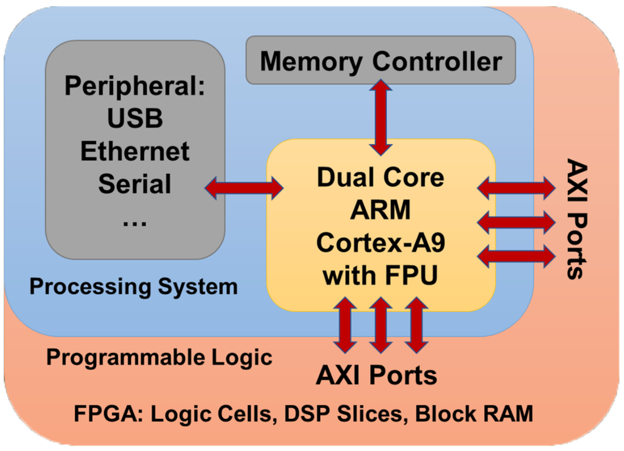


Figure 7.12. Zync Hardware Architecture.

7.1.5 FPGA

The FPGA must contain enough programmable logic blocks (Figure 7.13) for processing and enough memory for data holding to be able to support the processing of all SPOT units. These algorithms involve multiple layers of filters that are relatively expensive in terms of computing latency, limiting their applicability to real-time processing on serial processors such as Central Processing Units (CPUs). SPOT filters are well suited for a hardware implementation on FPGAs which can dramatically increase performance per watt in comparison to the equivalent software implementation (e.g. MATLAB) taking the advantage of their parallelism in application execution. This is of primary importance in order to meet the real-time requirements needed where high-speed parallel data processing is requested. With parallel computing, multiple processing can be executed at the same time, allowing to run many functions at once. FPGAs contain an array of programmable logic blocks, let their interconnection and possibility to be reprogrammed. This is needed to implement different logic functions at any stage during and after the design process. What has been done here is customizing FPGA resources to serve different purposes. This involves modeling the program instructions using configurable logic blocks to perform complex functions, or basic logics like "OR", "AND" and dedicated multiplexers.

Implementation Approach

As shown in Figure 7.14 and Figure 7.15, the algorithms are developed and implemented following the co-design methodology where the design flow is divided across the two implementation platforms: FPGA (PL) and Microprocessor (PS).

Figure 7.15 shows the design flow of SPOT system using different software tools. Implementing SPOT algorithms using Hardware Description Language (HDL) requires thousands of coding lines which is impractical and time-consuming. An alternate solution is using Xilinx System Generator coupled with MATLAB-Simulink graphical interface. Also Vivado Design Suite software was used to synthesize,

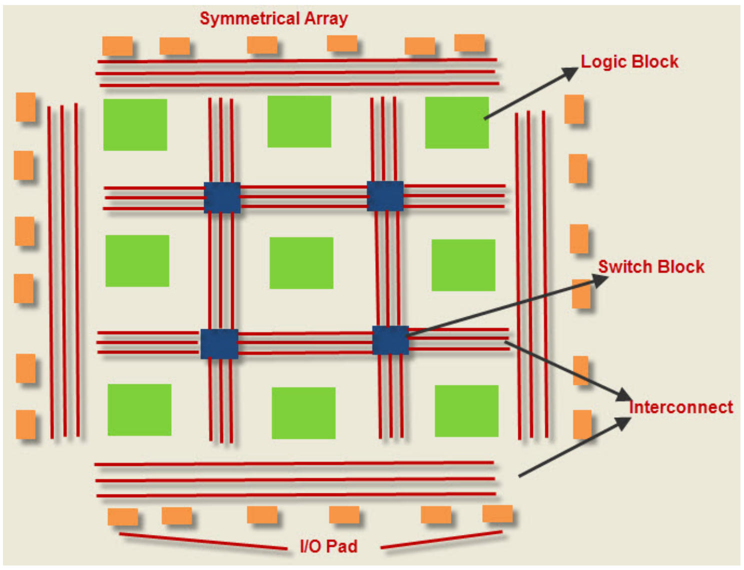


Figure 7.13. FPGA architecture description.

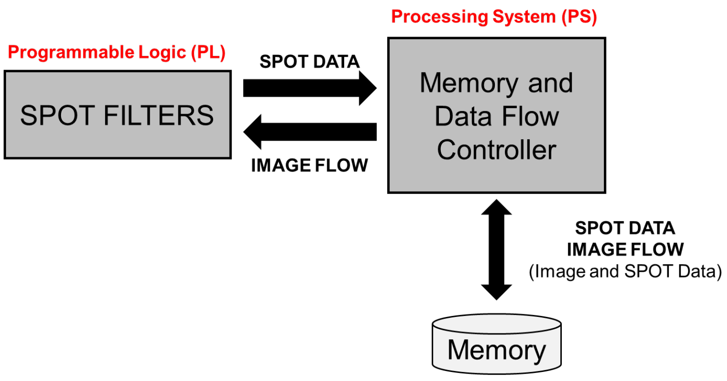


Figure 7.14. Design Strategy on ZYNQ.

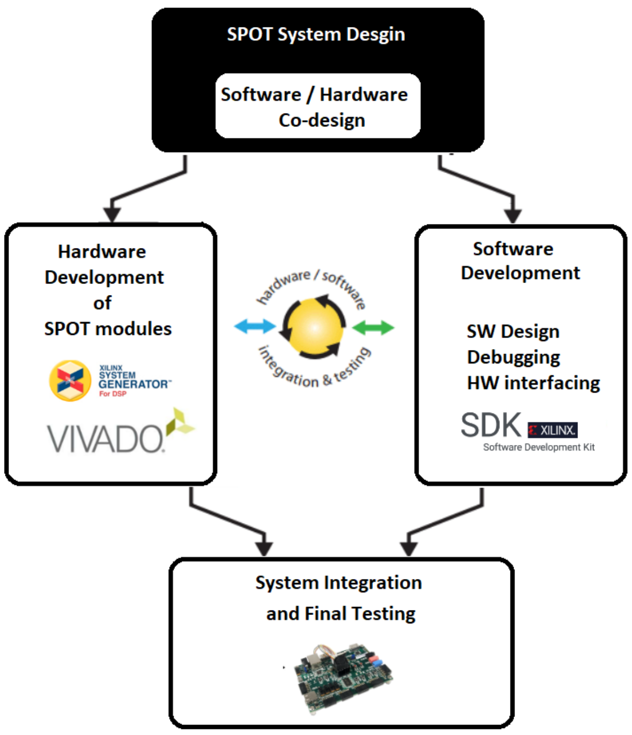

SPOT System Desgin
Software / Hardware Co-design
Hardware Development of SPOT modules
XILINX SYSTEM GENERATOR For DSP
VIVADO
hardware / software
integration & testing
Software Development
SW Design
Debugging
HW interfacing
SDK XILINX Software Development Kit
System Integration and Final Testing

Figure 7.15. SPOT Design Flow for Zynq SoC.

implement and analyze the hardware designs. Figure 7.16 shows the SPOT design in Vivado:

- The PL section (Figure 7.16) hosts SPOT overlay, which contains all the modules (Figure 7.17) described previously and includes an AXI DMA (Direct Memory Access) to provide high-bandwidth direct memory access between PL-section and AXI-Stream off-chip memory (DDR).

- The PS section (Figure 7.16) contains a representative block to configure via a user interface the clock frequency, the interfaces and interrupts between FPGA and microprocessor.

Xilinx Software Development Kit (SDK) was used to develop an embedded application on microprocessor that handles data flow within the SoC and external devices. The SDK is the first application IDE (Integrated Development Environment) to deliver homogenous and heterogeneous multi-processor design, debugging and performance analysis. Additionally, the SDK contains multiple drivers (PS) to facilitate interactions with the hardware SPOT overlay (PL) implemented in the Vivado Design Suite environment.

SPOT Algorithms Implementation

All the SPOT modules were developed separately under the System Generator software tool and exported as IP (Intellectual Property) cores to Vivado software. The modules are linked together using a modular approach, which facilitates debugging

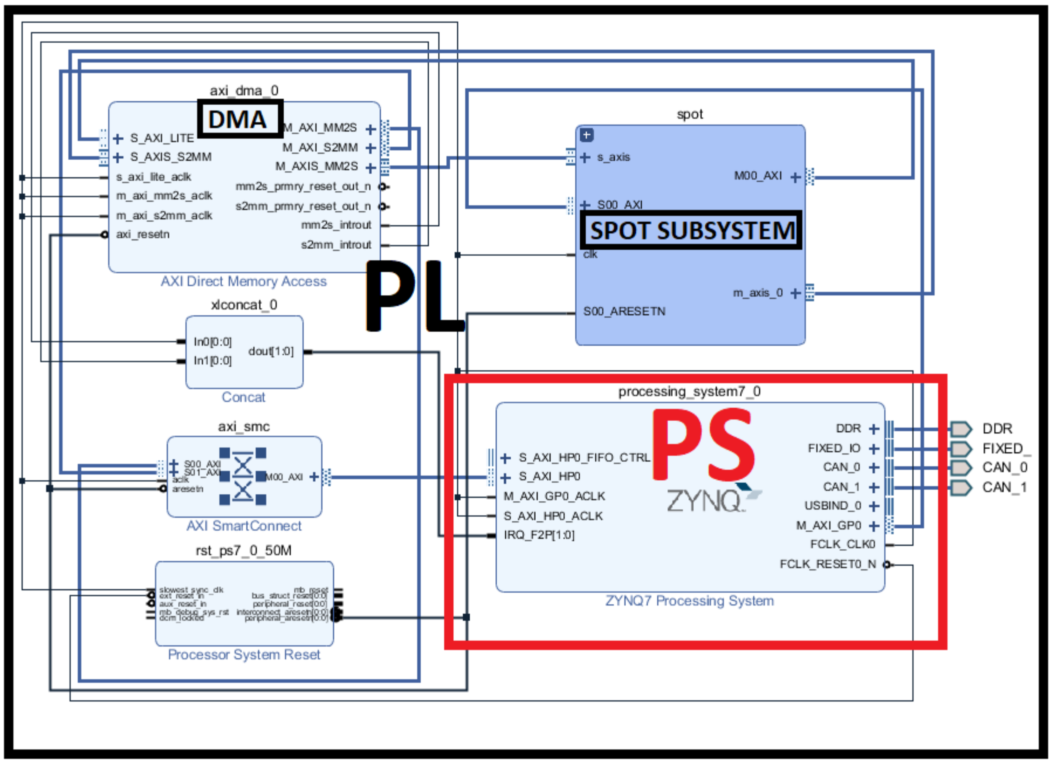


Figure 7.16. SPOT Architecture on Vivado.

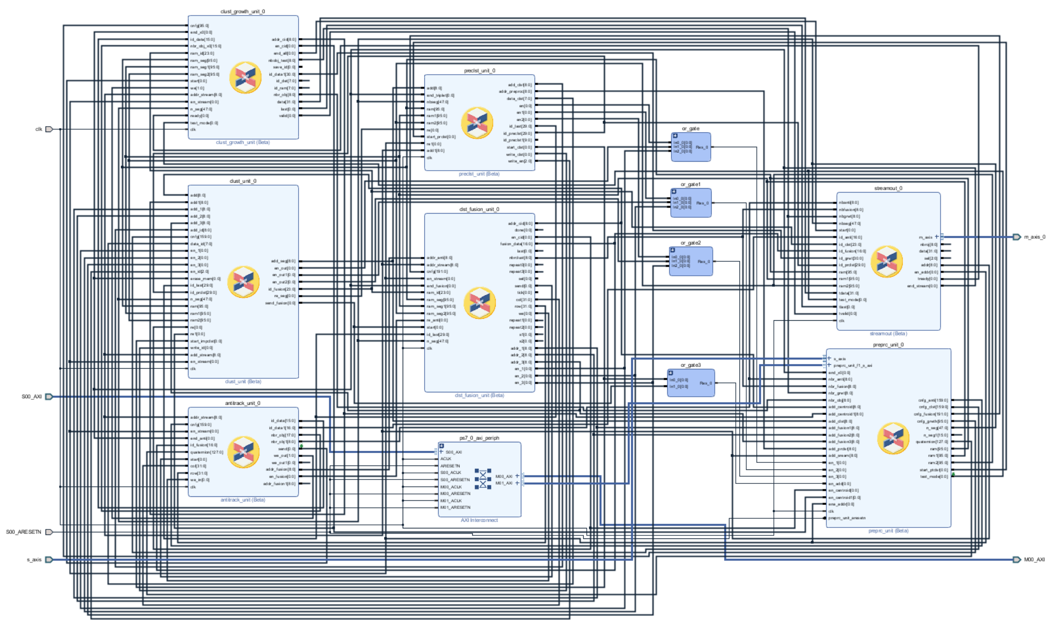


Figure 7.17. Modules of SPOT Subsystem Vivado

and allows any modifications in the future. Programming an FPGA is a process of customizing its resources to serve different purposes which involves modeling the program instructions using basic configurable logic blocks, RAM blocks, DSPs, and IP cores to perform complex functions.

To increase the efficiency and productivity of sequential codes, several ideas have been used to take advantage of the parallel processing capabilities of the FPGA. For instance, a parallel "for loop" has been implemented where the statements in the loop to run in parallel on separate processors.

End-to-End Data Flow

The diagram in Figure 7.18 shows the end-to-end data flow. The image stream is stored in the local memory of the camera to ensure a short time interval between successive images. Immediately, as a certain number of images are received, the memory controller inside the PS is triggered to move them from the camera to the DDR through the Controller Area Network (CAN) bus with a data rate of 1MB/s.

Shortly afterward, the memory controller moves the images from DDR to PL using the AXI-Stream interface that guarantees a high data transmission rate because it does not rely on memory addressable data transfer, allowing for an unlimited data burst size. Unlike AXI-Stream, AXI-Lite is used to grant the configurations and remotely check the on-board system status. Due to the limited on-chip memory, there is a constraint to work with triplets only. Hence the modules' local memories are updated after processing the entire triplet.

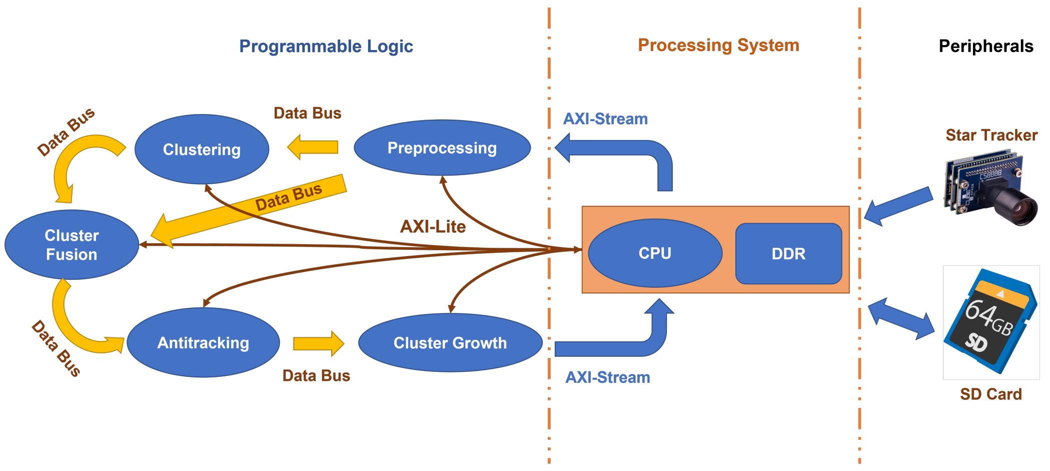


Figure 7.18. Implementation Data Flow

The raw image is transmitted in zigzag to the *Pre-processing* module. Within *Pre-processing* there is a receiver module that acquires pixels in which the counter is incremented by one step for each pixel received. In addition, the module contains a processing unit to retrieve and store, in local memory, all the data necessary for future processing. The *Clustering* module starts immediately, while the *Pre-processing* continues to process the other two subsequent images. After processing data from three images, they are sent in pairs to the Cluster Fusion module. For example, the previous module sends the pair 1-2 (image-1 and image-2), followed by pair 2-3 (image-2 and image-3). This module takes the consistent objects in the three images to calculate and store their centroids. Then, the object centroids values are moved to the *Antitracking* module to filter out the stars and keep only the detected RSOs. Finally, the outputs are processed through the *Cluster Growth* module to keep tracking moving objects in subsequent triplets. The triplets based acquisition philosophy differs from Figure 7.10. This is due to mission's constraints related to the selected ST which is developed with this acquisition philosophy. Contiguity time and pause analysis maintain the same role but now the contiguity images in the triplet are separated of δt while the triplets are separated of Δt. Initially, the SPOT implementation receives three images and process them, then it starts receiving couples of images. Each couple is separated of Δt and within a couple the time interval is δt. Moreover, the time constraints for these two values are: $\delta t \leq 200\,ms$ and $\Delta t \leq 2.9\,s$.

The final data are stored in non-volatile memory (SD card) and retained until they are requested by the ground station. Note that the design of SPOT overlay presented in the Figure 7.17 is based on the description discussed above.

Implemented Modules

Pre-Processing

The raw image from the ST gets into the *Pre-processing* module (Figure 7.19). The "APSover Unit" distinguishes the high-energy pixels belonging to stars' or RSOs' signals from the low-energy pixels (noise). The selected groups of foreground pixels are called *segments*. Each segment may contain one or more pixels. The threshold is a configurable parameter set by the CPU (in PS) through the AXI-lite interface. The function of the *Coordinate Generator Unit* is to assign the pixel coordinates of the received image. The information generated by the two units is enough to extract all the data related to the segments by using the *Processor Engine Unit.* The extracted outputs are the pixels' coordinates, energy, weighted energy, and length of each detected segment. The *Data Controller Unit* takes the processed data from the *Processor Engine* and saves them on the RAM blocks (Figure 7.20), which results in significant off-chip memory access reduction leading to low latency caused by the bandwidth limitation of external memory. The RAM block is replicated to store the up-coming processed data of the images, as shown in the figure 7.20. Each module contains its own local memory.

Clustering

The *Neighborhood Segments Check Unit* (Figure 7.21) has to detect all the neighboring pixels belonging to individual objects and to produce the *primitive clusters.* After this step, all the primitive clusters composed of one pixel are discarded.

Under dynamic conditions, objects in the FOV can generate dashed streaks. This means that pixels associated with the same object can be spread out into several small, distinct groups. For this reason, the *Filters Unit* (Figure 7.21) was developed in order to connect the same object (star or RSO) to different primitive clusters of the broken streak. The *Filters Unit* is based on the previous described filters: *Minimum distance filter*, *Increasing length filter* and *Density filter.*

IDs Assigning Unit (Figure 7.21) is used to allocate an identifier number to every segment. It re-allocates a new identifier for the segments associated with the same

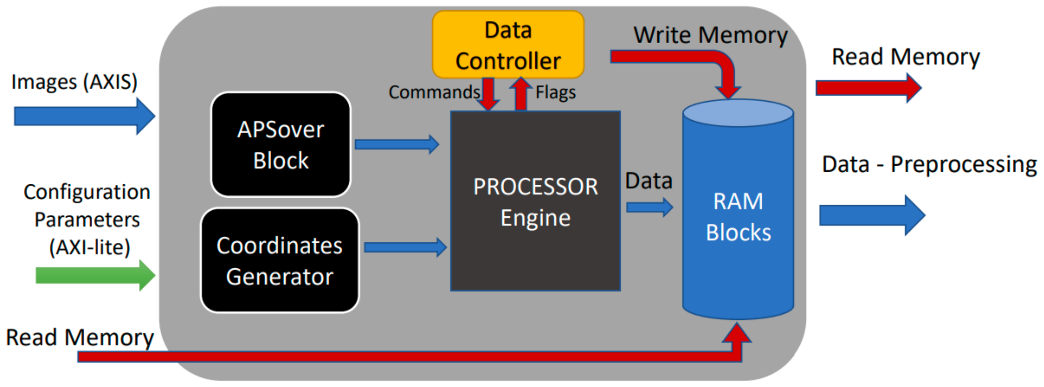


Figure 7.19. Pre-processing Scheme

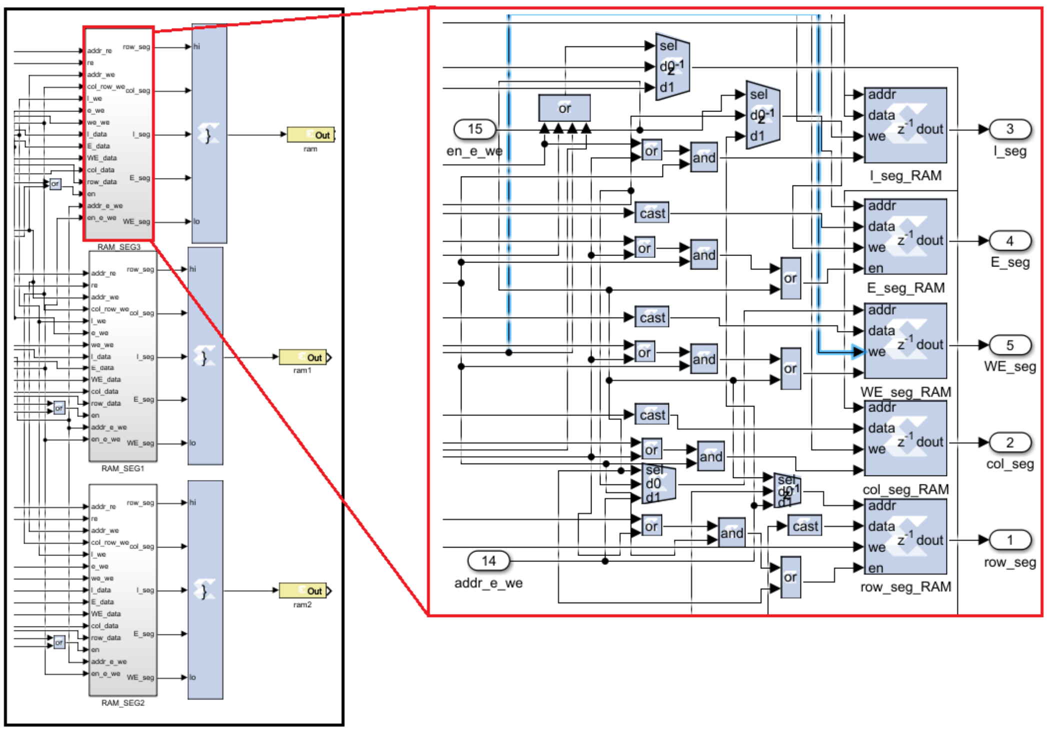


Figure 7.20. Triplet Pre-Processing Block RAMs

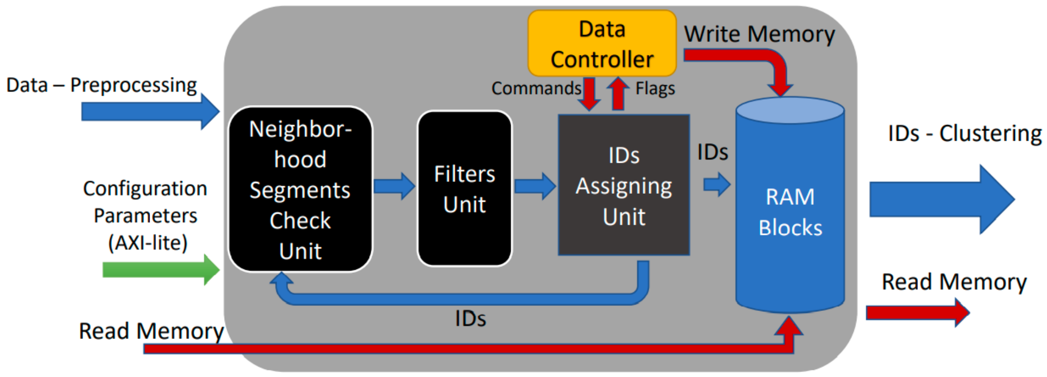


Figure 7.21. Clustering Scheme

object. Referring to Figure 7.22, the segments with identifiers equal to 17 and 18 (left part of Figure 7.22) are grouped together with segment identifiers equal to 15 and then they are re-assigned with the lowest identifier value. Consequently, a large cluster with an identifier equal to 15 is formed, as shown in the right part of Figure 7.22. The other clusters which are not merged still preserve their original IDs.

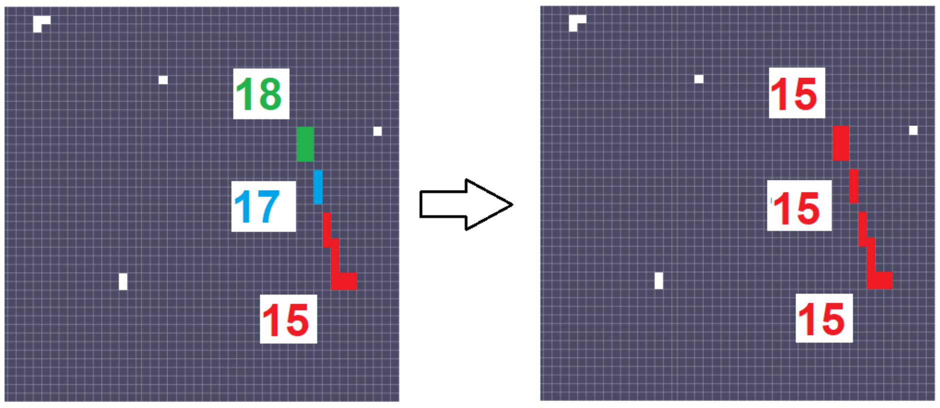


Figure 7.22. Clustering effect: before (left) and after (right)

Cluster Fusion

Neighborhood Segments Check Unit (Figure 7.23) takes two consecutive data of *Pre-processing* module and IDs of the *Clustering* module as inputs to compare the clusters. Comparison means a check on the neighboring pixels in two successive images. If neighboring pixels are present and close enough between them, then the two clusters are merged and belongs to the same objects. For the *Filters Unit* (Figure 7.23) and *IDs Merging Unit*, the operations are very similar to the one described for the *Clustering* module. The main difference is that the *Clustering* acts with clusters coming from the same image while *Cluster Fusion* acts with clusters coming from two consecutive images (δt separation).

The *Centroids Calculation Unit* processes the IDs of the merged clusters along with the *Pre-processing* outputs to calculate the cluster's centroids.

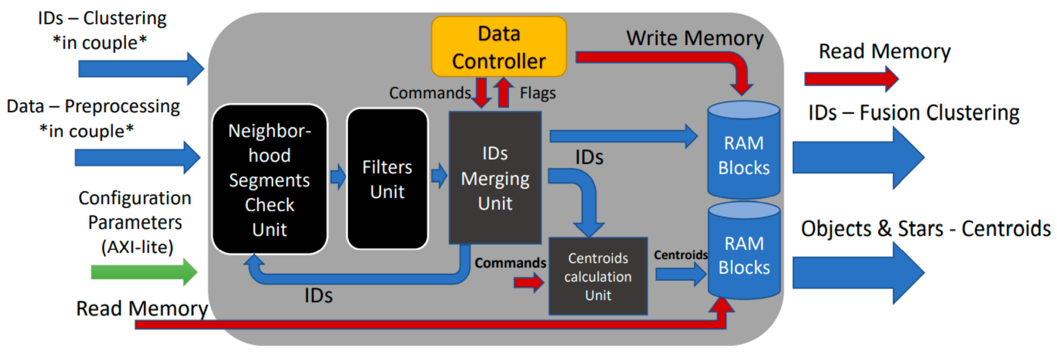


Figure 7.23. Cluster Fusion Scheme

It is useful to remember that the *Cluster Fusion* output contains only the centroids of the merged clusters, which are the clusters that appear in both consecutive images (Figure 7.24). As a result, it is possible to identify the persistence of objects in the FOV of the sensor.

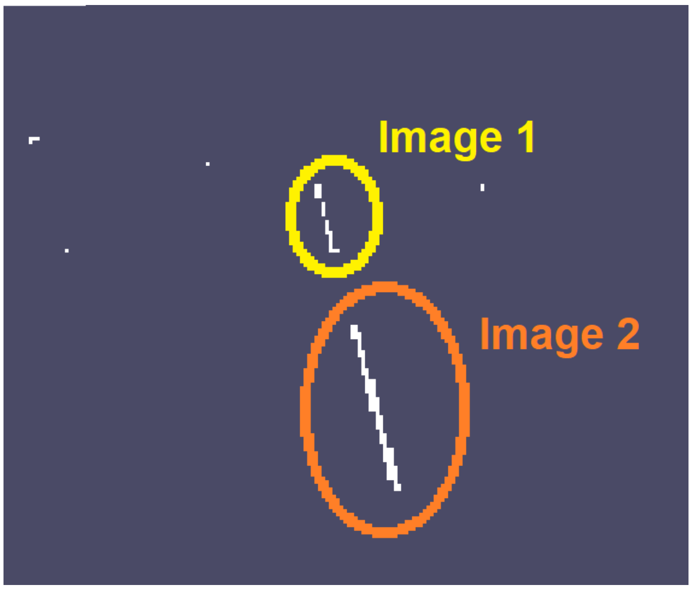


Figure 7.24. The moving object is shown in two subsequent images

Antitracking Module

The *Antitracking* module (Figure 7.25) is required to remove stars from the set of detected objects using the *Star Filtering Unit.* This unit requires the Rotation Matrix (RM) converted from the current attitude quaternion, a star catalogue (Hipparcos Catalogue) and the sensor's characteristics as inputs. All the objects recognized as stars are filtered out.

Cluster Growth Module

The *Cluster Growth* module (Figure 7.26) is used to deduce the direction of the

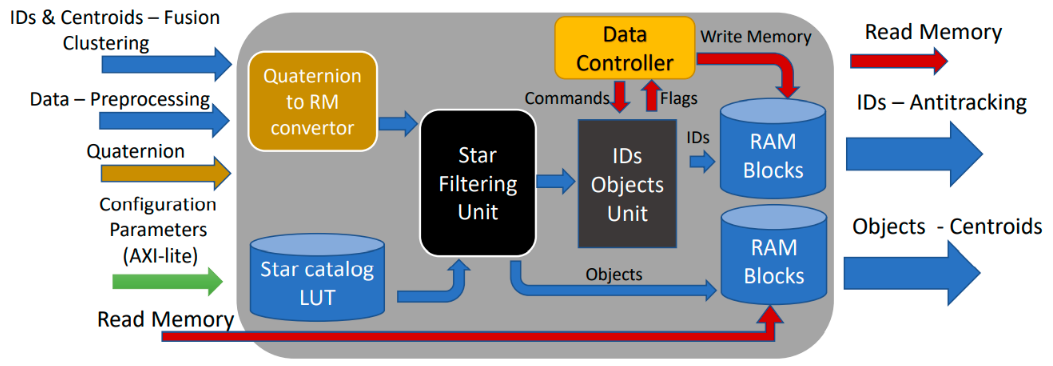


Figure 7.25. Antitracking Scheme

moving objects in the star sensor's FOV. This module is applied to build an on-board database of tracked objects. It processes successive couples of images. The goal is to track RSOs during their time spent in the FOV and to record the positions and time instants of each couple of images. The output of this module represents the final output of the on-board SPOT unit. Figure 7.44 shows the tracked object that appears in the three merged images.

The on-board data must be sent to the PS side and stored in the external memory (SD card). When the data is requested, it is transmitted to ground station for post-processing and orbit determination.

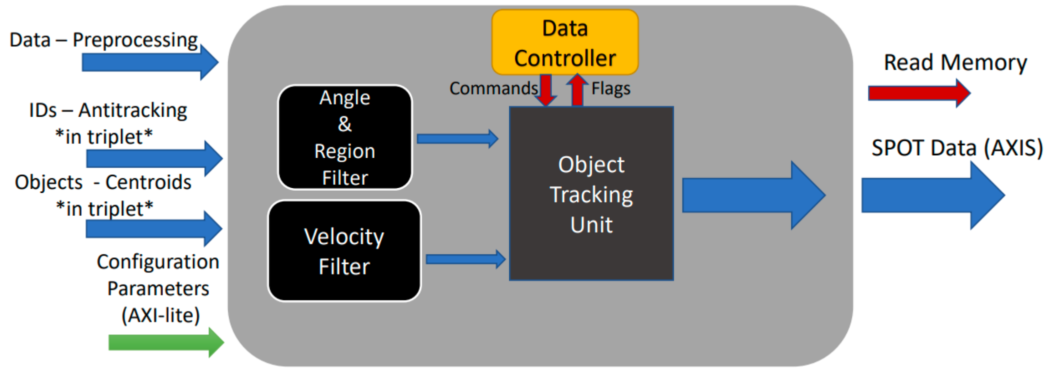


Figure 7.26. Cluster Growth Scheme

Implementation Techniques

System Generator

Generally, FPGAs are programmed with a HDL: Verilog, SystemVerilog, or Very High Speed Integrated Circuits HDL (VHDL). To implement SPOT algorithms using HDLs requires thousands of coding lines, which is impractical and time-consuming. An alternate solution is using Xilinx System Generator, coupled with a graphical interface under the MATLAB-Simulink that enables the use of the MathWorks model-based Simulink design environment for FPGA design. It makes the implementation very easy to work with in comparison to the other software for hardware description. The library of Xilinx System Generator includes many building blocks, allowing faster prototyping and design from a high-level programming standpoint. As a result, designers can define an abstract representation of a system-level design and easily transform the algorithms into a gate-level representation. Another benefit of using the Xilinx System Generator for the hardware implementation is that it allows the FPGA module to be co-simulated with the test vectors provided by MATLAB Simulink blocks. In software co-simulation, all Xilinx blocks are connected between *Gateway In* and *Gateway Out* blocks, which respectively behave as input and output for the hardware design.

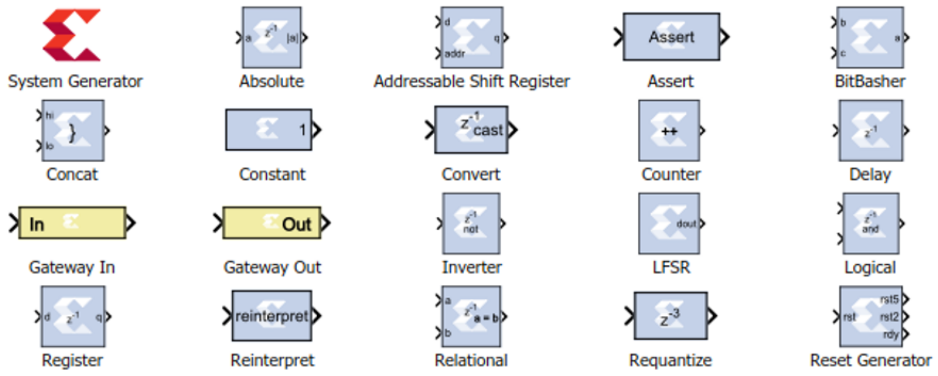


Figure 7.27. Xilinx blockset in simulink

Fixed Point Arithmetic

Normally, floating-point implementations require larger amounts of FPGA resources. This increased use of resources results in higher energy consumption and, ultimately, in an increase of the overall cost of implementing a design. Therefore, the reduction in the use of FPGA resources inherently leads to lower power consumption and enables massively increased computing capabilities within the FPGA. Again, converting from a floating-point design to a fixed-point design can significantly save power and area efficiency while maintaining the same level of precision and comparable performance. In some cases, the results can even be improved. To meet these challenges, it is necessary to thoroughly evaluate the lower precision (fixed point) implementations of MATLAB codes before targeting the FPGA implementation.

Pipelining

An important objective to be taken into account during the design is to increase the clock rate and throughput of an FPGA. This is achieved by pipelining design mechanism. Pipelined designs take advantage of the parallel processing capabilities of the FPGA to increase the efficiency of sequential code. Pipelined implementation foresees that the code is divided into several small parts, which are separated with the use of registers. With Pipelining, the processor can start executing a new input without waiting for the previous one to complete.

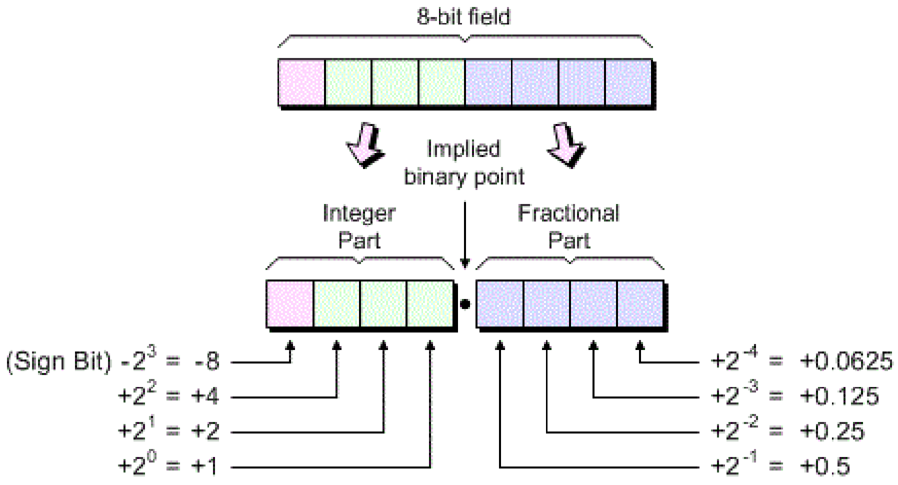


Figure 7.28. Fixed point arithmetic description.

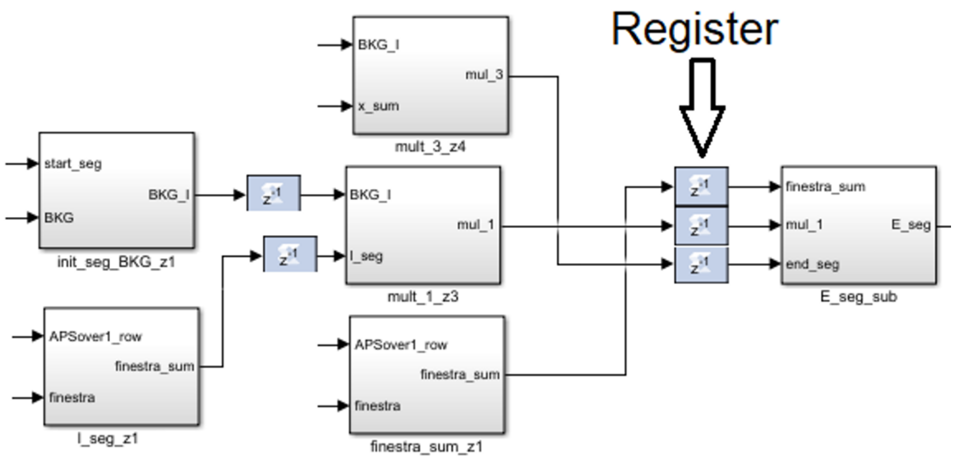


Figure 7.29. Design with Pipelining approach.

The different processors are separated by buffer registers, which are all linked to the clock signal. At each clock cycle, the registers become accessible for writing, which causes the data to pass to the next step.

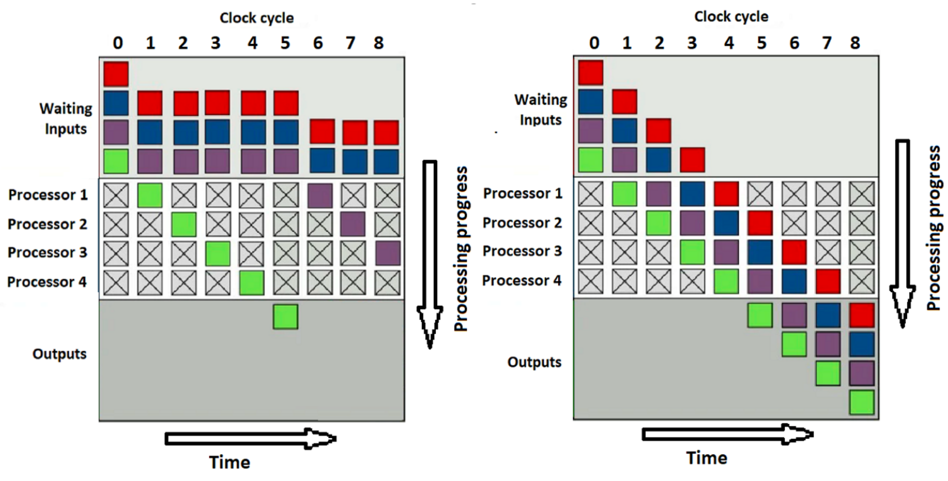


Figure 7.30. Non-Pipelining process (left) and Pipelining process (right)

In the non-pipelined design (Figure 7.30), all four inputs will take twenty clock cycles to be fully processed. This happens both because each input occupies all four processors untill the output is produced and each processor takes one clock cycle. In the pipeline design (Figure 7.30), the input accesses the processor row as soon as it is free. Therefore, the output is produced for each clock pulse starting from the fourth clock cycle. Actually, each input must pass through four registers during its processing before reaching the output. All four entries will take six clock cycles to be processed. The example in Figure 7.30 indicates that the pipelined design significantly increases the frequency and throughput compared to the non-pipelined design.

Shared Memory and Resources

Since the SPOT design contains units that have a similar architecture, it is useful to share the same architecture, possibly adding a few adjustments, rather than duplicating them for each unit. These adjustments greatly optimize the use of area resources on the FPGA. In addition, shared resources can reduce complexity, increase productivity, reduce maintenance costs, and use the additional resources available for additional functionality. An additional step used to reduce area usage is memory sharing. In FPGA, shared memory refers to block and distributed RAMs accessible, in a clock cycle, by several different processing units within a parallel computation, which allows a very fast and efficient implementation.

Reliability

Reliability is an important theme in space applications. It regards both SW and HW and every space system needs to prove high levels of it (according to the mission

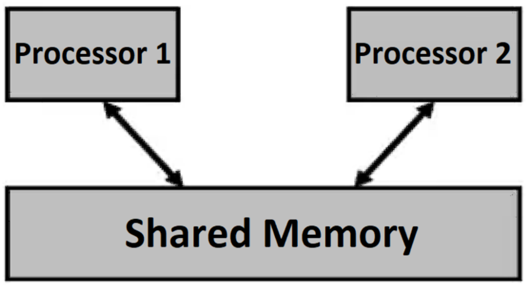


Figure 7.31. Shared memory strategy.

context). In particular, Soc/FPGA based HW's reliability is threatened by many actors like the ionizing radiation. Static Random Access Memory (SRAM)-based FPGAs like the Xilinx are highly susceptible to the ionizing radiation environment in space, typically in the event of a nuclear explosion. They generate photocurrents through the semiconductor material, causing memory cells to switch and transistors to change the logical state randomly. A SEU typically occurs by the change of logical state of a memory cell under the effect of a charged particle. It is a transient effect that will be erased by rewriting the affected memory cell. Any electronic circuit which has memory cells is likely to experience SEUs. This erroneous signal may remain in the digital system and can even propagate to other digital modules resulting in a failure. To mitigate space radiation effects in both their configuration and user memory, Triple Modular Redundancy (TMR) and algorithms for error detection and correction are applied to reduce the susceptibility of the implemented SPOT algorithms to space radiations, increasing the system's reliability.

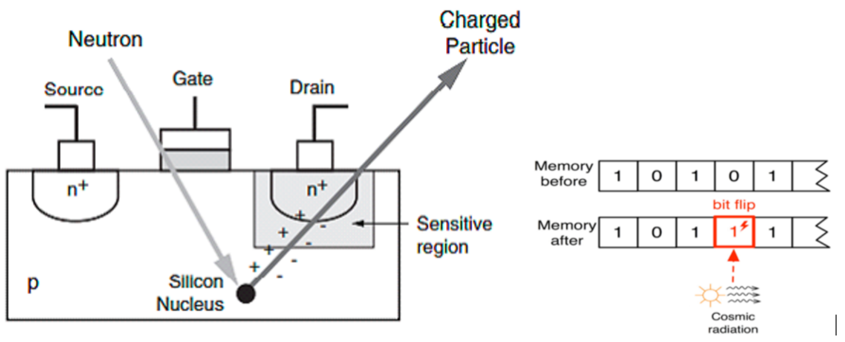


Figure 7.32. Radiation effect over SRAM.

For best results a manually design for TMR was done. It involves the implementation of three instantiations of the Processor Element (PE) with the majority voting upon the outputs. Typically, a TMR implementation requires spatial separation of the logic within the FPGA to ensure that a SEU does not corrupt more than one of the three instantiations. Another way to mitigate the effect of SEUs is to exploit the Soft Error Mitigation (SEM) Intellectual Property (IP) cores provided by Xilinx to perform SEU detection, correction and classification for configuration memory. The

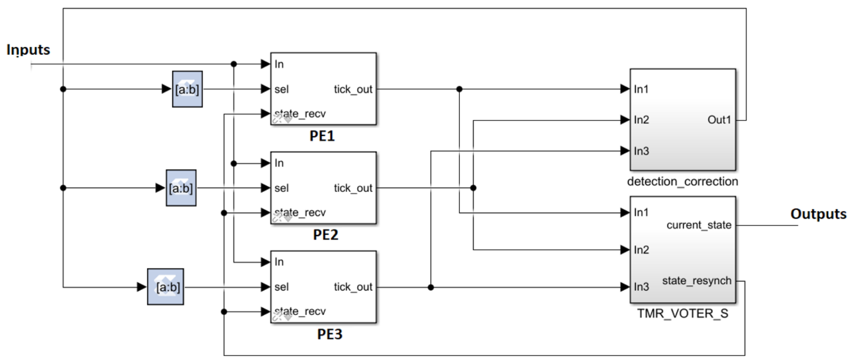


Figure 7.33. Triple Modular Redundancy with error detection and correction.

cores utilize device primitives such as Internal Configuration Access Port (ICAP) and FRAME_ECC (Error Correction Code) to the clock and observe the Readback CRC (Cyclic Redundancy Check) feature to continuously scan the configuration cells. For SEUs correction, the IP cores perform the necessary operations to locate and correct errors.

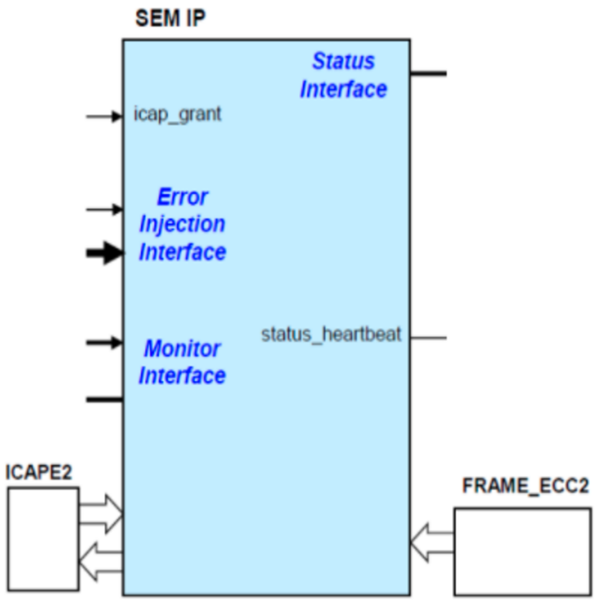


Figure 7.34. Software error mitigation IP.

The SEM IP cores also perform emulation of SEUs by injecting errors into configuration memory. The error injection feature provides a means to evaluate and test the SEU mitigation capabilities of the IP cores without the need for expensive test time at a radiation effects facility.

7.1.6 Test and Results for Hardware/Software SPOT Integration

The Hardware In the Loop (HIL) set-up which is used for the whole SPOT integrated system is shown in Figure 7.35.

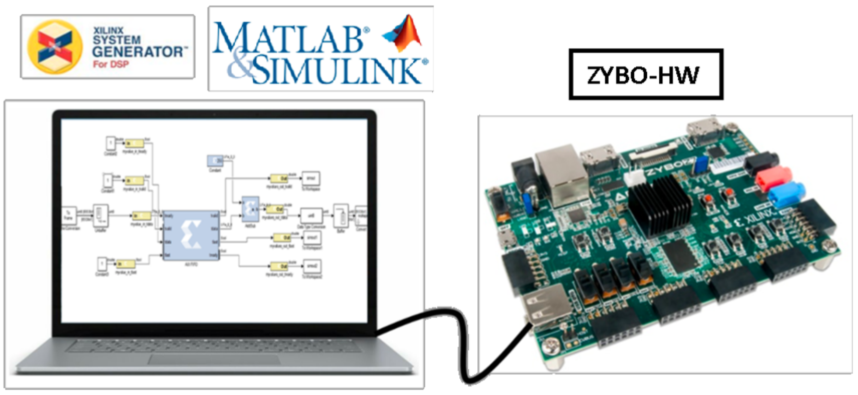


Figure 7.35. HIL set-up.

It uses System Generator to verify the performance and robustness of the control algorithms and to simulate critical scenarios. System Generator includes a HIL-simulation tool that can be used for simulating together with FPGAs. The FPGA is connected to a computer by JTAG cable and inputs-vectors are simulated on the computer(Simulink) and sent to the FPGA. The FPGA performes the calculation then it produces the outputs to be sent back to the computer (Simulink). The personal computer is also in charge of storing the real and simulated images which constitute the inputs for the ZYBo implemented SPOT solution. The images do not come from a camera because there is no interest here in verifying the camera-board communication protocols.

Preliminary test campaign is organized in a stack of tests where on-board modules are implemented and tested alone. Each module received as inputs the produced outputs from the previous module:

1. *Pre-Processing* Unit Test;
2. *Clustering* Unit Test;
3. *Cluster Fusion* Unit Test;
4. *Antitracking* Unit Test;
5. *Cluster Growth* Unit Test.

To assess the goodness of results, the SPOT integrated solution outputs will be compared with the results from a MATLAB copy of the SPOT code to validate the successful implementation of the integrated SPOT. The work-flow used for every target code is described into steps:

- Inputs are taken from their specific folder;
- Run the MATLAB copy of the code;

- Collect and store the outputs;
- Run the Implemented code using the same Inputs;
- Collect and store the outputs;
- Board and MATLAB outputs are compared and discussed.

Results

Pre-Processing Unit Test

In this test, six real images were considered and *Pre-Processing* module was applied for each of them. This was carried out both with FPGA and MATLAB codes and then results were compared. Results of *Pre-Processing* unit are segments and energies related to the over thresholds pixels of the image. An example of input and related reconstructed output is shown in Figure 7.36 just for the most relevant part of the image.

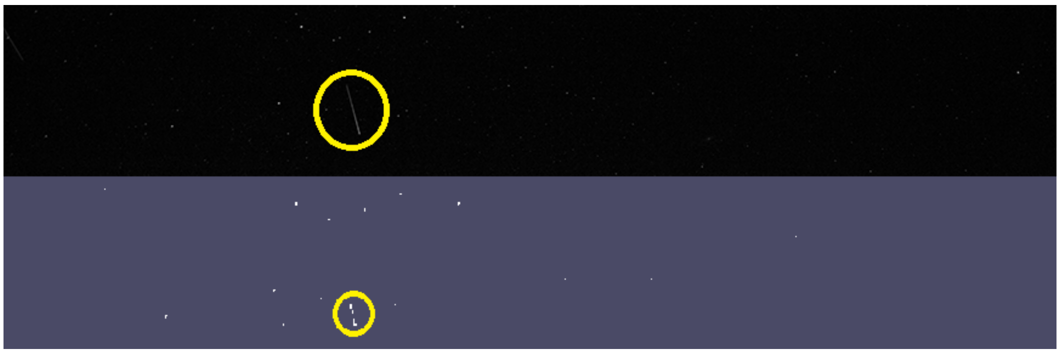


Figure 7.36. Detail of *Pre-Processing* module input (top) and output (bottom).

To compare results of both MATLAB and FPGA module we used performance indices described in [106]. These indices are defined to assess the similarity between two masks produced by different segmentation algorithms. These masks are matrices of binary numbers. When they are summed element by element, they give a two dimensional array of 0, 1 and 2 values. By recording the number of 0 elements ($cntr_0$), 1 elements ($cntr_1$) and 2 elements ($cntr_2$) it is possible to compute two indices: G and G_0. The third index is defined as the ratio between the number of active pixel in mask 1 ($cntr_{m1}$) and mask 2 ($cntr_{m2}$). By evaluating each index for each tested image, the following values were obtained by comparing MATLAB and ZYBo SPOT versions:

- $G = \frac{cntr_0 + cntr_2}{cntr_0 + cntr_1 + cntr_2} \times 100 = 100\,\%$
- $G_0 = \frac{cntr_2}{cntr_1 + cntr_2} \times 100 = 100\,\%$
- $M = \frac{cntr_{m1}}{cntr_{m2}} = 1$

These values mean that there is a perfect superposition between MATLAB and ZYBo codes' output masks for what concerns the localization of over threshold and under threshold pixels ($cntr_1 = 0$). Moreover, another comparison has been done also with segments' energies which shows a complete success of the FPGA *Pre-Processing* Unit implementation.

Clustering Unit Test

In the implementation of the *Clustering unit*, the same pre-processing test of the image was considered and its outputs were injected to the *Clustering* unit.

Segments' positions, energies, weighted energies, and lengths for the *Clustering* unit were provided to both MATLAB and FPGA. The output data are the coordinates, energies, and dimensions of the *Clustering*. To compare the output data, the difference (ϵ) between the output data from different sources (y_{FPGA} and y_{MATLAB}) was evaluated:

$$\epsilon = y_{FPGA} - y_{MATLAB} \tag{7.6}$$

In Figures 7.37-7.38 these differences are related to the *Primitive Clustering* Part. Here the blue color is related to the difference distribution while the red color is the mean value, and the black color is used for the $\pm\sigma$ curves. "Col" and "Row" represent the x and y centroids' coordinates in the image plane reference frame, respectively.

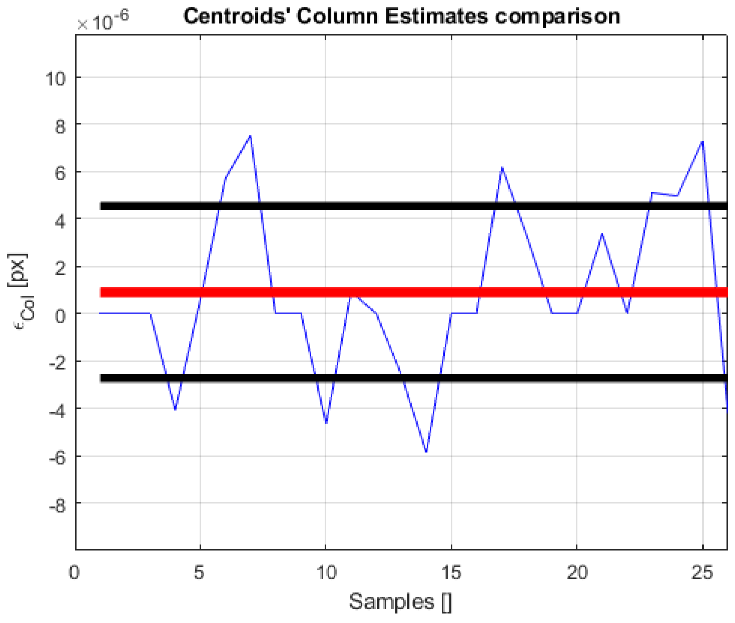


Figure 7.37. Cluster centroids' columns estimate differences over the whole samples set.

The centroids column and row plots show a predicted trend with an average value of zero, considering the interval $\pm 3\sigma$ the differences are still very far from the maximum tolerance of 10^{-1}. When comparing clusters' dimension and energies,

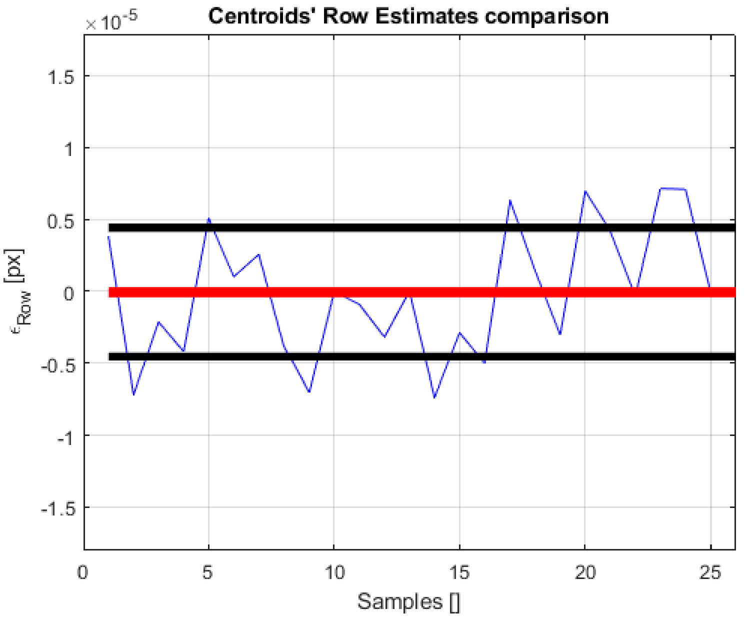


Figure 7.38. Cluster centroids' rows estimate differences over the whole samples set.

the output data from FPGA and MATLAB are matched. This happens because truncation and rounding are not applied and they have an integer type nature.

To see the effects of the *Improved Clustering* module, Figure 7.39 is considered.

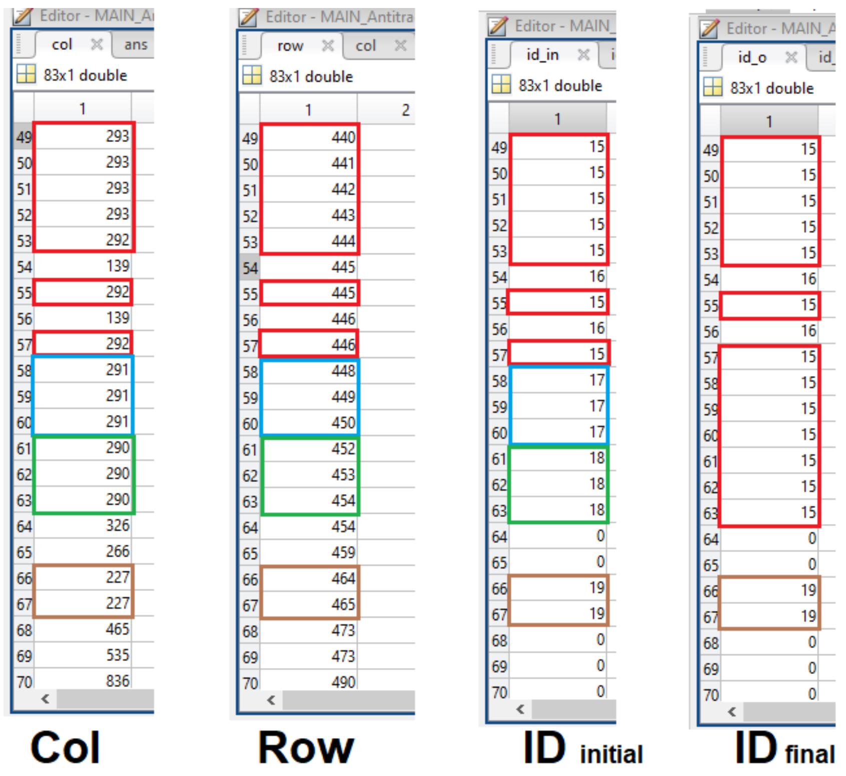
In Figure 7.39, the first two columns from left are the coordinates of the segments that compose primitive clusters. As can be seen on the last two columns on the right, a change in the segments corresponding clusters identifier (ID) appears.

Before *Improved Clustering* , each segment has an associated cluster identifier.

As a result of applying several filters and identifier re-assigning, the primitive clusters ID= 17 and ID= 18 are grouped with clusters ID=15. Hence, a large cluster of ID= 15 was formed (red boxes). The other clusters which are not merged still preserve their original index as can be seen for cluster ID= 19 (brown box). The three primitive clusters 15, 17, 18 which are merged in the end are the streak fragments associated with the ISS cluster Figure 7.40 (Cluster 19 is not present in the image detail).

Cluster Fusion Unit Test

In the *Cluster Fusion* module test, two consecutive images input were provided to the target unit. Clusters of the first and second images are compared using the previously described filters in the architecture section and the considered images are merged and displayed (Figure 7.41). The results provided by ZYBo have been compared with MATLAB and they both show the same results: five clusters have been correctly merged. Four of them are related to stars while the remaining one is the ISS. In particular, Figure 7.42 shows centroids relative to the ISS in the first and second images of the considered couple, along with the final structure resulting



Editor - MAIN_A
col | ans
83x1 double

	1
49	293
50	293
51	293
52	293
53	292
54	139
55	292
56	139
57	292
58	291
59	291
60	291
61	290
62	290
63	290
64	326
65	266
66	227
67	227
68	465
69	535
70	836

Editor - MAIN_Antitra
row | col
83x1 double

	1	2
49	440	
50	441	
51	442	
52	443	
53	444	
54	445	
55	445	
56	446	
57	446	
58	448	
59	449	
60	450	
61	452	
62	453	
63	454	
64	454	
65	459	
66	464	
67	465	
68	473	
69	473	
70	490	

Editor - MAIN_
id_in | i
83x1 double

	1
49	15
50	15
51	15
52	15
53	15
54	16
55	15
56	16
57	15
58	17
59	17
60	17
61	18
62	18
63	18
64	0
65	0
66	19
67	19
68	0
69	0
70	0

Editor - MAIN_A
id_o | id_
83x1 double

	1
49	15
50	15
51	15
52	15
53	15
54	16
55	15
56	16
57	15
58	15
59	15
60	15
61	15
62	15
63	15
64	0
65	0
66	19
67	19
68	0
69	0
70	0

Figure 7.39. *Improved Clustering* effects on cluster identifier evolution.

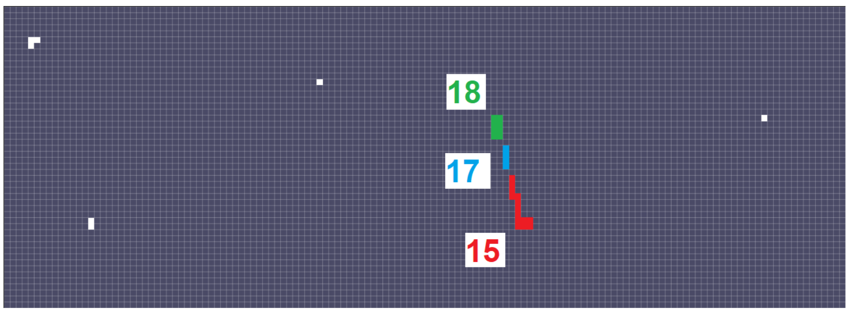


Figure 7.40. ISS' streak primitive clusters (Image Detail).

from the fusion operation.

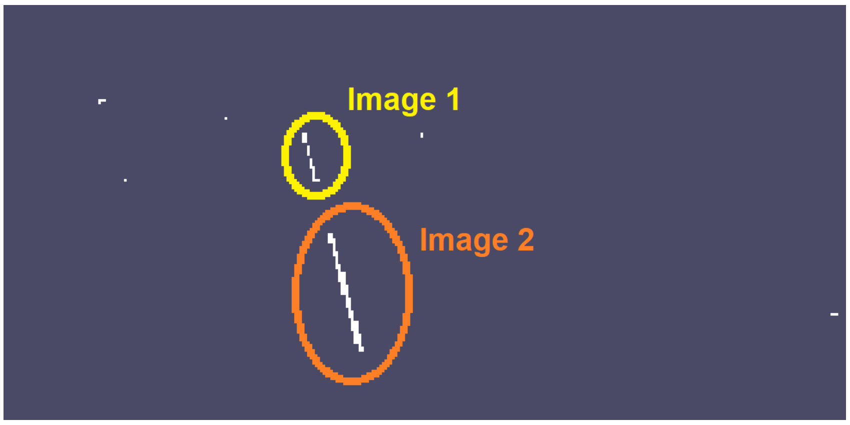


Figure 7.41. ISS' fused streaks by Cluster fusion module.

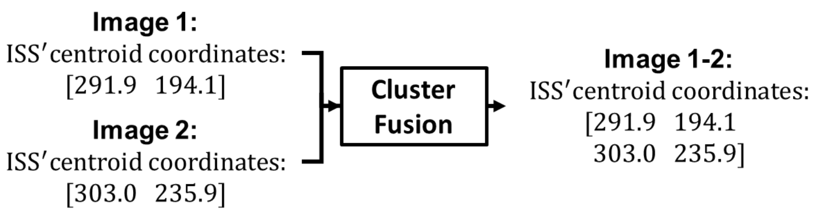


Figure 7.42. ISS' fused centroids by Cluster fusion module.

Antitracking Unit Test

For the *Antitracking* test, a single image with its associated sensor position was considered (Figure 7.43).

In particular five clusters were detected, of which four stars and one object (ISS). The huge FOV and low resolution made all the other stars being segmented as single pixels which were correctly filtered by the *Clustering* module. Just the brightest stars resulted big enough not to be considered as noise. After applying the *Antitracking*, all the objects recognized as stars (yellow circles in the figure) are filtered out using the Hipparcos catalog, while the ISS is not deleted (red circle in the figure) and is included in the *Antitracking* outputs to be processed by the *Cluster Growth* unit.

The used quaternion in this test was:

$$q = [-0.526829; 0.222090; -0.342451; -0.745556] \tag{7.7}$$

Objects centroids before antitracking are:

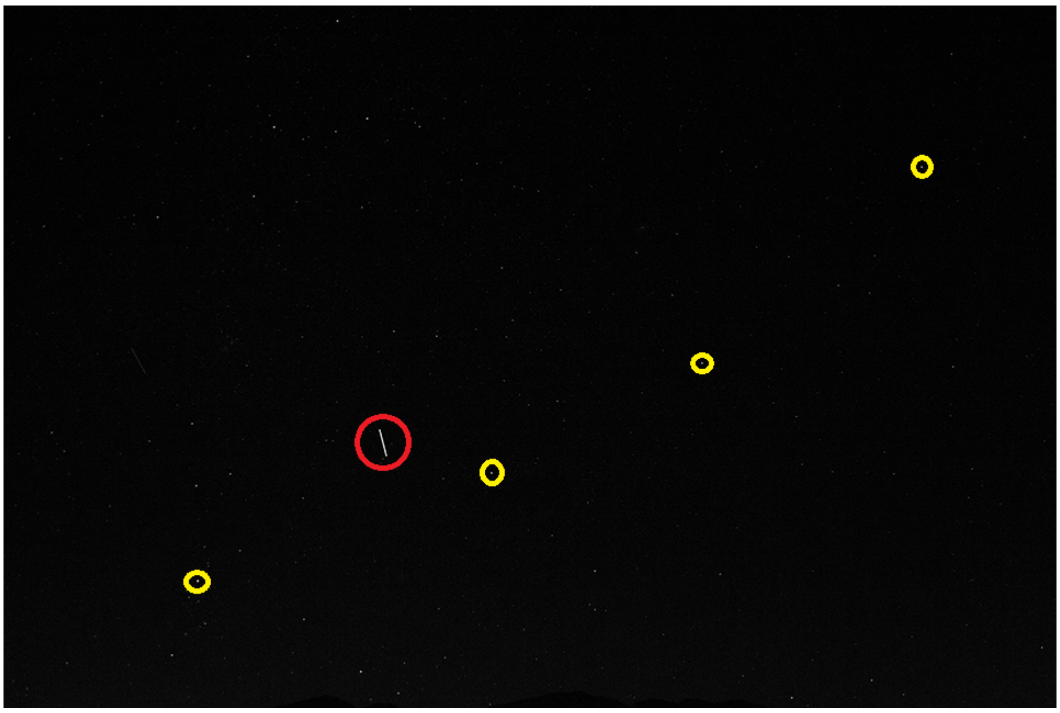


Figure 7.43. ISS (red circle) and stars (yellow circle)

$$centroids = \begin{bmatrix} 178.2 & 116.4 \\ 445.5 & 215.3 \\ 637.4 & 315.5 \\ 837.5 & 494.6 \\ 345.5 & 399.0 \end{bmatrix} \tag{7.8}$$

where, the first column represents the detected centroids' x-coordinate in the image plane and the second column represents the detected centroids' y-coordinate in the image plane, after the *Antitracking* module, clusters relative to stars are recognized as stars and deleted only the cluster relative to the ISS remains:

$$centroids = \begin{bmatrix} 345.5 & 399.0 \end{bmatrix} \tag{7.9}$$

Here again, MATLAB and FPGA results perfectly match.

Cluster Growth Unit Test

For the last test, all the six consecutive images processed in three couples are used. The entire sequence is shown in Figure 4.31. By Cluster Growth results, the unique object in the FOV was tracked (the ISS). Its centroid coordinates evolution is shown in Table 7.7. MATLAB and FPGA outputs are numerically the same, considering the first digit of the decimal part. This is because, by project analysis, the *Cluster Growth* centroids are rounded to the first digit.

In the end, this ISS sequence was the same used both for testing YOLOv3, YOLOv4 detection algorithms with the RSOs tracking module. Results for YOLOv4

Table 7.7. ISS centroids in the image sequence

Image	Column (px)	Row (px)
1	291.9	194.2
2	303.0	235.9
3	315.1	281.9
4	326.3	324.9
5	336.4	363.6
6	345.5	399.0

were shown in Table 4.19. Here the Object 1 x and y coordinates correspond to ISS. By comparing Table 4.19 frames 2-7 coordinates with Table 7.7 values it is possible to compare the AI-based approach and traditional/hardware integrated system outputs. Rounding to the first decimal digit the AI-based algorithm outputs, differences between centroids coordinates are below 1.1 px and this is a good result. This small difference is due to the different image segmentation and centroids computation process. Results are anyway encouraging if we consider that this was just a preliminary comparison

HW Implementation Technical Results

A performance value for real time application is the *Processing time.* It includes the entire path from the PS-side in which three consecutive images are streamed into PL-side, the processing of images inside the PL-side and then the returning of the outputs to PS-side. The average time achieved of processing three successive images with size 1024 $\times$ 1024 was around 143 ms with clock frequency of the PL-side equal 40 Mhz compared to the software solution on the computer (based on Intel CPU) that takes 10 s with 3 Ghz clock frequency. The average time depends mainly on the clock frequency. Thus, by increasing the system clock frequency of PL-side the processing time decreases. Increased clock frequency, anyway, can lead to the so-called "timing violation" which occurs when the execution time requested by sections of code is shorter than execution time the bitfile achieves after compiling. To avoid this issue, the design should be separated by adding registers between combinational logic modules. Anyway, this solution will give rise to more resources usage on the FPGA and increases its power consumption. The best practice is to make, instead, a trade-off between processing time and resource usage and check if the real-time requirement is met. Other parameters that can have minor effects on the processing time are the number of segments, objects detected, and the size of the image. Considering the Pause Analysis requirement to be satisfied, the 143 ms value is lower than 2.9 seconds and this proves that the following hardware implementation approach is in compliance with the project time constraints.

The three image sequence on which this time was measured is shown in Figure 7.44. The design was implemented on the Xilinx Zybo Board Zynq-7000 FPGA using Vivado 2018.3, System Generator, and SDK. A summary of FPGA resources usage is shown in the Table7.8. DSP is highly consumed due to the high use of arithmetic operations. Regarding the Figure 7.45, the FPGA, where the SPOT algorithms have been implemented, shows a power consumption around $0,397\,W$ which is the

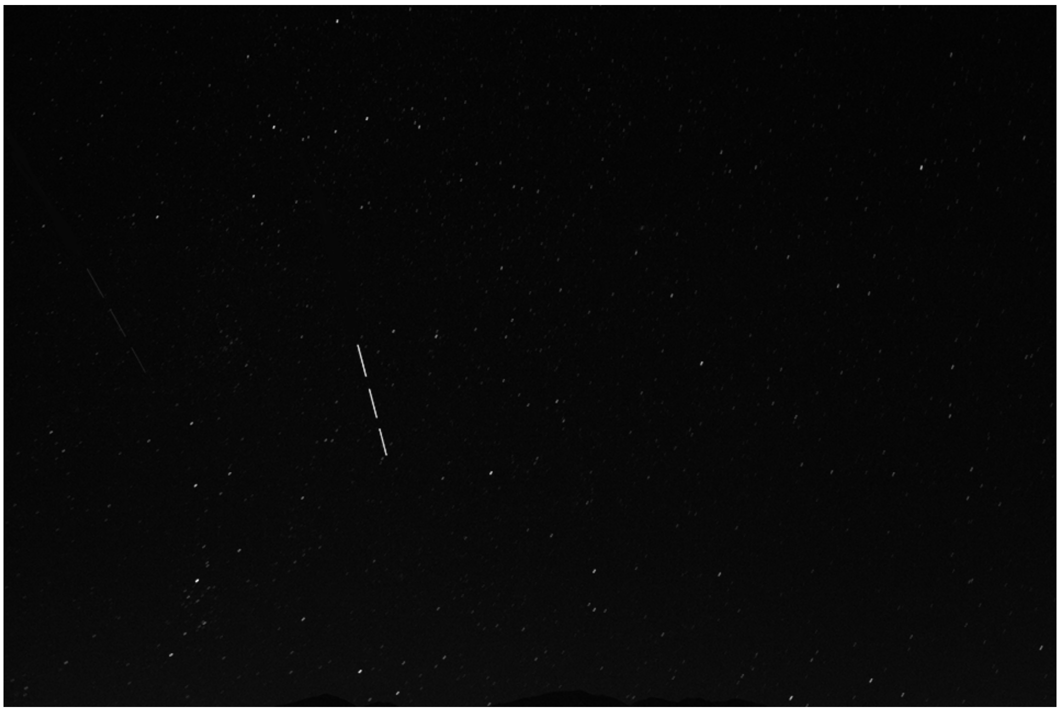


Figure 7.44. Three consecutive and merged images of a real passage. ISS is the brightest object while the weakest is a Starlink (on the left) .

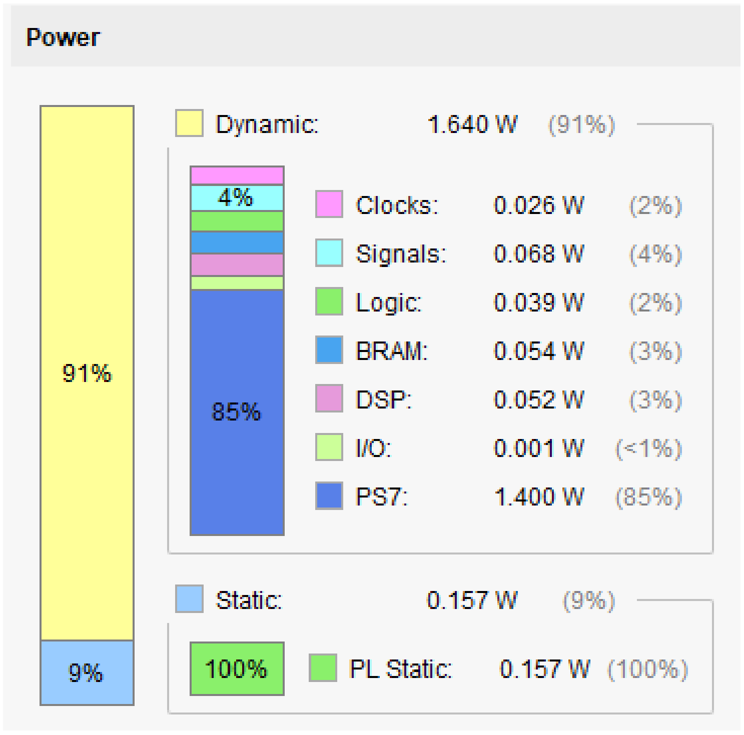
summation of PL-Static power ($0.157\,W$) plus PL-Dynamic power ($0.240\,W$).

Table 7.8. Resources Utilization

Resource	Utilization	Available	Utilization (%)
LUT	18786	53200	35.31
LUTRAM	1125	17400	6.47
FF	16822	106400	15.81
BRAM	76	140	54.29
DSP	187	220	85.00

Summary of the SPOT Activity

In this chapter SPOT project has been presented and described, individuating the context and the need of such a mission in the SSA and SST panorama. The description and high level design of both the units have been shown. In particular a deeper level of description was done for the on-board part which I personally lead as Co-PI of this project together with a team of five people. The on-board unit was entirely described with support figures and tested: both on an Intel-based personal computer and on a Soc/FPGA Zynq-7000 based hardware for real time applications and On-Board SPOT unit validation. Tests were conducted both using simulated ST images and real images from an equivalent Reflex camera (equivalent in terms of CMOS technology). Test results show that On-Board SPOT is capable of tracking objects while being compliant with time constraints (mission requirements) on the target hardware. The implementation approach using System Generator



Power

🟨	Dynamic:	1.640 W	(91%)
🟪	Clocks:	0.026 W	(2%)
🟦	Signals:	0.068 W	(4%)
🟩	Logic:	0.039 W	(2%)
🟦	BRAM:	0.054 W	(3%)
🟪	DSP:	0.052 W	(3%)
🟩	I/O:	0.001 W	(<1%)
🟦	PS7:	1.400 W	(85%)
🟦	Static:	0.157 W	(9%)
🟩	PL Static:	0.157 W	(100%)

Figure 7.45. Power consumption

made it more practical and faster in comparison to the other software solutions. The techniques applied for the implementation show a significant effect in terms of area usage and the data throughput of the FPGA, resulting in a suitable solution for the use in real-time missions.

The V&V of the hardware implemented algorithm against the theoretical results is proved. Moreover, in the ISS passage tests with real images it was possible to compare On-Board SPOT results against an AI-based RSOs detection and tracking design.

In the future the implementation of State Machine on the Microprocessor part of the SoC for controlling data flow and decision making will be carried out. These will include the implementation of a CAN protocol interface to connect it to the on-board camera as well as other communication protocols to exchange data with the on-board computer.

7.2 Activity Status and Next Steps

At the moment of writing the SPOT project went up a change of mission platform and target hardware for the implementation. These last months several activities of integration on the new target hardware were performed and the removal of *Pre-processing*, *Clustering* and *Cluster Fusion* units was carried out, being them already implemented on the selected ST model. In this way, with just the implementation of *Antitracking* and *Cluster Growth*, it was possible to select a new SoC/FPGA Hardware with less resources which proved to be suitable for the real time working of this new On-Board SPOT version.

The selected hardware is PHOEBE Board [117] (Figure 7.46). It was used by School of Aerospace Engineering for the STECCO (Space Travelling Egg-Controlled Catadioptric Object) mission. PHOEBE Board Features include the following:

- Microcontroller to handle data and tasks;
- FPGA for fast image data processing;
- Low power consumption: up to 0.3 W;
- Small Satellites Protocol: CAN, I^2C and RS485;
- Low Mass: 54 g;
- Dimensions: $99 \times 19 \times 29$ mm^3.

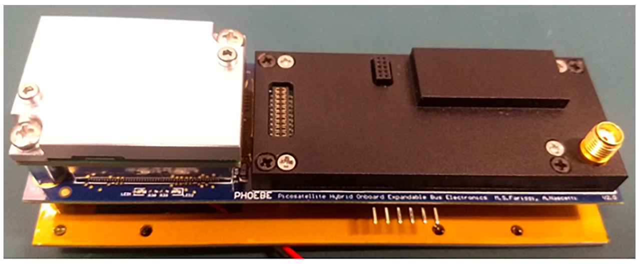


Figure 7.46. PHOEBE board, (Credits to School of Aerospace Engineering).

The PHOEBE system on Chip/Field Programmable Gate Array (SoC/FPGA) hardware offers parallel computation capabilities of the FPGA which boost the data processing time for real-time applications.

The selected ST is SAGITTA Star Tracker from ARCSEC SPACE (Figure 7.47). Its most important features for these purposes are the following:

- FOV: 24.8 deg (squared);
- Sensor Size: 2048×2048 px
- Cut-Off Magnitude: 7;
- Working Frequency: Up to 10 Hz;
- Accuracy: 2 arc seconds (1 sigma) cross-bore-sight, 10 arc seconds (1 sigma) around bore-sight.

With dimensions of $45 \times 50 \times 95$ mm^3, a mass of 275 g and a low power consumption of 1.4 W, SAGITTA is compact and suitable for microsatellites and cubesats missions. This device can perform image segmentation and clustering functions retrieving just the centroids and rough magnitudes of the objects in the FOV.

The integrated *Antitracking* and *Cluster Growth* modules are being tested with real images. Some night sky acquisitions were performed in the last two months

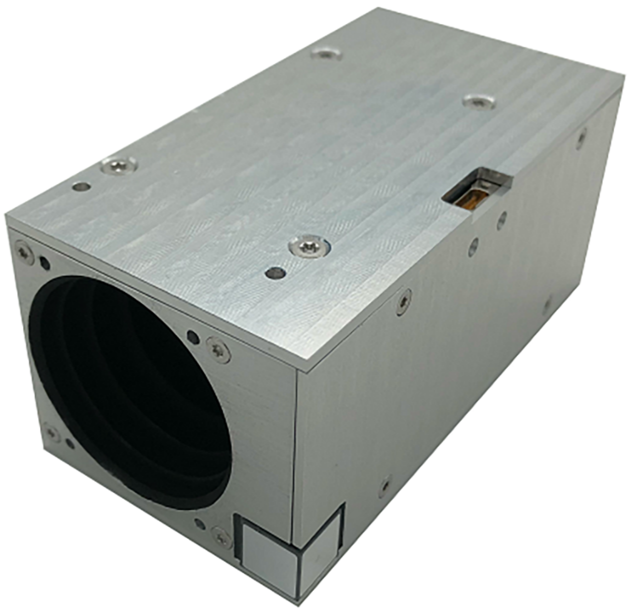


Figure 7.47. Sagitta ST, (Credits to ARCSEC SPACE).

to learn in handling and commanding the SAGITTA ST in the proper way to get objects centroids and a reliable quaternion information. These tests made me and my team starting a remote correspondence to change the ST firmware for improving the amount of output centroids to a useful number for the SPOT mission requirements. In parallel quaternion retrieval tests, centroids download tests both from single image and from a sequence of images have been carried out. *Antitracking* and *Cluster Growth* have been tested with these centroids and quaternion information. Now the next step is the interface protocol implementation between PHOEBE and SAGITTA for data and commands exchange. In the future, real time acquisition and processing tests will be carried out on the field to assess the correct working of the integrated SAGITTA-PHOEBE system. Then the whole payload will be mounted on a small satellite from School of Aerospace Engineering for its IOV mission. IOV mission activity is being defined in these months. In case of a successful IOV mission the design of a dual-purpose ST with the flight proven SPOT units will be pursued in parallel with the proposal of an IOV mission for the AI-based dual-purpose streaks detection algorithm.

Chapter 8

Conclusions

This work condenses the most important achievements during my PhD activity in these years. It has brought me to the right maturity for the conceptualization of a wider and future activity which is the development of the TRIDAENT ST. This work shows step by step the evolution of my PhD project. It starts from the urgent problem of space debris tracking and surveillance to be solved in the Space Domain Awareness panorama. It evolves into a research activity based on the AI technology for space debris detection starting from optical sensor images. My work reaches, then, the development of a dual-purpose AI-based algorithm for RSOs tracking and HAR estimation. This constitutes the starting point to realize a real space product whose preliminary design takes place within this work.

To face this activity I relied on my ability in real night sky images acquisition and post-processing in order to create all the material I needed for CNNs training and test. Moreover, I led acquisition campaigns which aimed to record into sequences the RSOs passages. I organized and classified these data both for Training and Validation Datasets, as well as for algorithm tests. Besides this, I used the HFST to generate simulated images for the datasets' dimension extension. This was needed to increase CNNs training accuracy. Datasets were created both for images segmentation and detection tasks. After this data organization and bibliography research, the realization of a U-Net based segmentation algorithm was possible. The result was the creation of a segmentation product which shows good and generalized segmentation properties for what concern the night sky images. This algorithm was named BOSS and was compared with the state of the art showing its robustness against several noise levels without requiring any calibration activity after its design. At this point the target moved on achieving the detection capabilities of RSOs which appear as streaks in most of the cases. I identified YOLOs NNs starting from the v3 and evolving into v4 to achieve streaks detection purposes. In order to train YOLO models, a combination of real night sky binarized images and a MATLAB streaks simulator was used to produce the needed datasets. Several kind of datasets were created with this own made dataset generation method for YOLOs TL processes. This was needed to increase the NNs robustness against the high fragmented streaks, achieving a satisfying mAP of 86% in streaks detection. Images segmentation and detection capabilities were successfully tested on real images before developing the Tracking algorithm, based on the minimization of a proposed cost function.

The entire algorithm chain was extensively tested and improved with the introduction of filtering actions to increase the reliability of tracked objects while avoiding the loss of precious information. Every process of building up streaks detection and tracking algorithm has been described, justified and detailed within this work and within related published papers.

The streaks detection capabilities have been exploited in another context such as the case of spinning optical sensors to test the algorithms output quality for HAR estimation. This was possible by coupling the developed algorithm with a least-square based module for angular rate determination. Moreover, the HAR estimation investigation brought me to design and propose a metric to solve stars' streaks from RSOs' streaks. The presented metric is just a starting point with very encouraging preliminary results. Extensive tests were carried out for HAR estimation in a wide range of angular speeds [5°/s, 15°/s] and for several angular rate directions showing the non controllable effect of initial attitude, angular rate direction and module on the number of tracked stars and the consequent quality of the HAR estimate.

This algorithm then, paved the way to the development of the AI-based ST idea for dual-purpose application. The idea is shown in Chapter 6, the main work-flow and modules are presented, described and the suggested solutions for each module implementation are highlighted. The main part of these solutions is developed in the previous chapters.

After the development of AI-based algorithms for SSA and SN, several points still need to be analyzed and studied in the future:

- Heuristic optimization of *Cost Function* weights through extensive tests (Monte Carlo simulations) can be performed to select the right weights for achieving the correct streaks tracking once the sensor features have been specified;

- Monte Carlo simulations to statistically evaluate the performances of the HAR estimation algorithm against initial attitude, angular rate directions and angular rate module;

- Improvement of the geometric based metric for resolving RSOs and stars with the introduction of a velocity based contribute to resolve moving RSOs parallel to stars trajectories;

- Streaks centroids' uncertainties characterization for the investigation of a statistical approach to solve the tracking problem;

- Study of the image segmentation problem through BlendMask and comparison w.r.t. U-Net based algorithm;

- Study of the streaks and points detection problem through YOLOv4 and v7, comparing their performances;

- Implementation of the previous NNs on a dedicated HW as NVIDIA Jetson or Raspberry Pi 5 coupled with a Neural Compute Stick or NCS (Intel) to get a real time HW deployed version of the AI-based algorithm to perform on the field preliminary ground based tests.

Last but not least, during this PhD activity I led and carried on the SPOT project through its preliminary and final design from architecture, SW, HW selection and documentation point of view. The experience and know-how gained as a Co-PI of the on-board SPOT project since 2019 helped me a lot during my PhD activity. The project was close related to my PhD research and while SPOT is in the final stage of implementation, my work represents its future in an AI-based way.

Bibliography

[1] Esa debris. `https://www.esa.int/ESA_Multimedia/Images/2020/05/ESA_2019_report_on_space_debris_-_evolution_in_all_orbits`. Accessed: 2020-05-07.

[2] Esa about space debris. `https://www.esa.int/Space_Safety/Space_Debris/About_space_debris`. Accessed: 2023-09-28.

[3] Victor Jungnell. Guidance methods for earth observation satellites, 2012.

[4] C.C. Liebe. Accuracy performance of star trackers - a tutorial. *IEEE Transactions on Aerospace and Electronic Systems*, 38(2):587–599, 2002.

[5] Benjamin B. Spratling and Daniele Mortari. A survey on star identification algorithms. *Algorithms*, 2(1):93–107, 2009.

[6] Jens Utzmann and Axel Wagner. Sbss demonstrator: A space-based telescope for space surveillance and tracking.

[7] Dario Spiller, Edoardo Magionami, Vincenzo Schiattarella, Fabio Curti, Claudia Facchinetti, Luigi Ansalone, and Alberto Tuozzi. On-orbit recognition of resident space objects by using star trackers. *Acta Astronautica*, 177:478–496, 2020.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] What is padding in neural network? `https://www.geeksforgeeks.org/what-is-padding-in-neural-network/`. Accessed: 2023-05-04.

[10] Rajalingappaa Shanmugamani. *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing Ltd, 2018.

[11] R Abay, S Gehly, S Balage, M Brown, R Boyce, et al. Maneuver detection of space objects using generative adversarial networks. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2018.

[12] Roberto Furfaro, Richard Linares, and Vishnu Reddy. Shape identification of space objects via light curve inversion using deep learning models. In *AMOS Technologies Conference, Maui Economic Development Board, Kihei, Maui, HI*, 2019.

[13] James Allworth, Lloyd Windrim, James Bennett, and Mitch Bryson. A transfer learning approach to space debris classification using observational light curve data. *Acta Astronautica*, 181:301–315, 2021.

[14] Stefano Silvestrini and Michèle Lavagna. Neural-aided gnc reconfiguration algorithm for distributed space system: development and pil test. *Advances in Space Research*, 67(5):1490–1505, 2021.

[15] Lorenzo Pasqualetto Cassinis, Robert Fonod, Eberhard Gill, Ingo Ahrns, and Jesús Gil-Fernández. Evaluation of tightly-and loosely-coupled approaches in cnn-based pose estimation systems for uncooperative spacecraft. *Acta Astronautica*, 182:189–202, 2021.

[16] David Rijlaarsdam, Hamza Yous, Jonathan Byrne, Davide Oddenino, Gianluca Furano, and David Moloney. Efficient star identification using a neural network. *Sensors*, 20(13):3684, 2020.

[17] Haris Iqbal. Harisiqbal88/plotneuralnet v1.0.0, December 2018.

[18] Sairaj Neelam. Yolo for object detection, architecture explained. `https://medium.com/analytics-vidhya/understanding-yolo-and-implementing-yolov3-for-object-detection-5f1f748cc63a`. Accessed: 2023-05-04.

[19] Evaluating detection (intersection over union). `https://www.oreilly.com/library/view/hands-on-convolutional-neural/9781789130331/a0267a8a-bd4a-452a-9e5a-8b276d7787a0.xhtml`. Accessed: 2023-05-10.

[20] Fares Fourati and Mohamed-Slim Alouini. Artificial intelligence for satellite communication: A review. *Intelligent and Converged Networks*, 2(3):213–243, 2021.

[21] Space debris and human spacecraft. `https://www.nasa.gov/mission_pages/station/news/orbital_debris.html`. Accessed: 2021-08-07.

[22] Harold D Black. A passive system for determining the attitude of a satellite. *AIAA journal*, 2(7):1350–1351, 1964.

[23] Craig L Cole and John L Crassidis. Fast star-pattern recognition using planar triangles. *Journal of guidance, control, and dynamics*, 29(1):64–71, 2006.

[24] Vincenzo Schiattarella, Dario Spiller, and Fabio Curti. A novel star identification technique robust to high presence of false objects: The multi-poles algorithm. *Advances in Space Research*, 59(8):2133–2147, 2017.

[25] Space surveillance and tracking - sst segment. `https://www.esa.int/Safety_Security/Space_Surveillance_and_Tracking_-_SST_Segment`. Accessed: 2021-08-07.

[26] Eu-sst. `https://www.eusst.eu`. Accessed: 2021-08-07.

[27] Tim Flohrer and Holger Krag. Space surveillance and tracking in esa's ssa programme. In *Proceedings 7th European Conference on Space Debris, Darmstadt, Germany, https://conference. sdo. esoc. esa. int*, volume 1, 2017.

[28] François Chollet. *Deep Learning with Python.* Manning, November 2017.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[30] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[31] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*, 2012.

[32] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *arXiv preprint arXiv:1805.11604*, 2018.

[33] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[35] Artificial intelligence in space. `https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/Artificial_intelligence_in_space`. Accessed: 2023-05-16.

[36] Satguard artificial intelligence to monitor spacecrafts' health. `https://www.iai.co.il/p/satguard`. Accessed: 2023-05-16.

[37] Shamim Sanisales and Reza Esmaelzadeh Aval. Artificial intelligence techniques for spacecraft health monitoring system - a survey. 03 2023.

[38] Richard Linares and Roberto Furfaro. Space objects maneuvering detection and prediction via inverse reinforcement learning. In *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, volume 46, page 46, 2017.

[39] Alejandro Pastor, Guillermo Escribano, and Diego Escobar. Satellite maneuver detection with optical survey observations. In *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, 2020.

[40] Ben Bradley and Penina Axelrad. Lightcurve inversion for shape estimation of geo objects from space-based sensors. In *Univ. of Colorado. International Space Symposium for Flight Dynamics*, 2014.

[41] Mikko Kaasalainen and Johanna Torppa. Optimization methods for asteroid lightcurve inversion: I. shape determination. *Icarus*, 153(1):24–36, 2001.

[42] Richard Linares, Roberto Furfaro, and Vishnu Reddy. Space objects classification via light-curve measurements using deep convolutional neural networks. *Journal of the Astronautical Sciences*, 67(3):1063–1091, 2020.

[43] David Rijlaarsdam, Hamza Yous, Jonathan Byrne, Davide Oddenino, Gianluca Furano, and David Moloney. A survey of lost-in-space star identification algorithms since 2009. *Sensors*, 20(9), 2020.

[44] Tamer Mekky Ahmed Habib. Artificial intelligence for spacecraft guidance, navigation, and control: a state-of-the-art. *Aerospace Systems*, 5(4):503–521, 2022.

[45] Maksim Shirobokov, Sergey Trofimov, and Mikhail Ovchinnikov. Survey of machine learning techniques in spacecraft control design. *Acta Astronautica*, 186:87–97, 2021.

[46] David Rijlaarsdam, Hamza Yous, Jonathan Byrne, Davide Oddenino, Gianluca Furano, and David Moloney. A survey of lost-in-space star identification algorithms since 2009. *Sensors*, 20(9):2579, 2020.

[47] Ari Silburt, Mohamad Ali-Dib, Chenchong Zhu, Alan Jackson, Diana Valencia, Yevgeni Kissin, Daniel Tamayo, and Kristen Menou. Lunar crater identification via deep learning. *Icarus*, 317:27–38, 2019.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[49] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[50] Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10), 2020.

[51] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[52] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[53] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[54] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[55] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[56] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[59] Yi Zhu and Shawn Newsam. Densenet for dense flow. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 790–794, 2017.

[60] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[61] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[62] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 734–750, 2018.

[63] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[64] Abdullah Rashwan, Agastya Kalra, and Pascal Poupart. Matrix nets: A new deep architecture for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[65] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[67] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[68] Songtao Liu, Di Huang, et al. Receptive field block net for accurate and fast object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 385–400, 2018.

[69] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[70] Wei Xu, Qi Li, Hua-jun Feng, Zhi-hai Xu, and Yue-ting Chen. A novel star image thresholding method for effective segmentation and centroid statistics. *Optik*, 124(20):4673–4677, 2013.

[71] Carl Christian Liebe. Accuracy performance of star trackers-a tutorial. *IEEE Transactions on aerospace and electronic systems*, 38(2):587–599, 2002.

[72] Likai Xu, Jie Jiang, and Lei Liu. Rpnet: A representation learning-based star identification algorithm. *IEEE Access*, 7:92193–92202, 2019.

[73] Zhaokui Wang and Yu-lin Zhang. Algorithm for ccd star image rapid locating. *Chinese journal of space science*, 26(3):209–214, 2006.

[74] Dario Spiller and Fabio Curti. A geometrical approach for the angular velocity determination using a star sensor. *Acta Astronautica*, 196:414–431, 2022.

[75] John Bernsen. Dynamic thresholding of gray-level images. In *Proc. Eighth Int'l conf. Pattern Recognition, Paris, 1986*, 1986.

[76] Dongju Liu and Jian Yu. Otsu method and k-means. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, volume 1, pages 344–349. IEEE, 2009.

[77] W Niblack. An introduction to image processing (pp. 115–116), 1986.

[78] Jiu-Lun Fan and Bo Lei. A modified valley-emphasis method for automatic thresholding. *Pattern Recognition Letters*, 33(6):703–708, 2012.

[79] Kan Jin, Yilun Chen, Bin Xu, Junjun Yin, Xuesong Wang, and Jian Yang. A patch-to-pixel convolutional neural network for small ship detection with polsar images. *IEEE Transactions on Geoscience and Remote Sensing*, 58(9):6623–6638, 2020.

[80] Dong Zhao, Huixin Zhou, Shenghui Rang, and Xiuping Jia. An adaptation of cnn for small target detection in the infrared. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 669–672, 2018.

[81] Zunlin Fan, Duyan Bi, Lei Xiong, Shiping Ma, Linyuan He, and Wenshan Ding. Dim infrared image enhancement based on convolutional neural network. *Neurocomputing*, 272:396–404, 2018.

[82] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[83] Nasser M. Nasrabadi. Deeptarget: An automatic target recognition using deep convolutional neural networks. *IEEE Transactions on Aerospace and Electronic Systems*, 55(6):2687–2697, 2019.

[84] Manshu Shi and Huan Wang. Infrared dim and small target detection based on denoising autoencoder network. *Mobile Networks and Applications*, 25(4):1469–1483, 2020.

[85] Xiaozhong Tong, Bei Sun, Junyu Wei, Zhen Zuo, and Shaojing Su. Eaau-net: Enhanced asymmetric attention u-net for infrared small target detection. *Remote Sensing*, 13(16):3200, 2021.

[86] Danna Xue, Jinqiu Sun, Yaoqi Hu, Yushu Zheng, Yu Zhu, and Yanning Zhang. Starnet: convolutional neural network for dim small target extraction in star image. In *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*, pages 1–7. IEEE, 2018.

[87] Danna Xue, Jinqiu Sun, Yaoqi Hu, Yushu Zheng, Yu Zhu, and Yanning Zhang. Dim small target detection based on convolutinal neural network in star image. *Multimedia Tools and Applications*, 79(7):4681–4698, 2020.

[88] Marco Mastrofini. nightskyunet, 2022.

[89] Georg Zotti and Alexander Wolf. Stellarium 0.19. 0 user guide. Technical report, Technical report. Available online at github. com/Stellarium/stellarium . . . , 2019.

[90] F Curti, D Spiller, L Ansalone, S Becucci, D Procopio, F Boldrini, and P Fidanzati. Determining high rate angular velocity from star tracker measurements. In *Proc. 66th Int. Astron. Congr.*, pages 1–13, 2015.

[91] Fabio Curti, Dario Spiller, Luigi Ansalone, Simone Becucci, Dorico Procopio, Franco Boldrini, Paolo Fidanzati, and Gianfranco Sechi. High angular rate determination algorithm based on star sensing. In *Advances in the Astronautical Sciences Guidance, Navigation and Control 2015, Vol. 154*, page 12. 2015.

[92] Vincenzo Schiattarella, Dario Spiller, and Fabio Curti. Star identification robust to angular rates and false objects with rolling shutter compensation. *Acta Astronautica*, 166:243–259, 2020.

[93] Vincenzo Schiattarella, Dario Spiller, and Fabio Curti. Efficient star identification algorithm for nanosatellites in harsh environment. In *Advances in the Astronautical Sciences, Vol. 163*, pages 287–306. 2018.

[94] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[95] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, 12(2):13–21, 2007.

[96] Laila Kazemi, John Enright, and Tom Dzamba. Improving star tracker centroiding performance in dynamic imaging conditions. In *2015 IEEE Aerospace Conference*, pages 1–8. IEEE, 2015.

[97] Rafael C Gonzalez, Richard E Woods, et al. Digital image processing, 2002.

[98] Daniel Chapman, Asad M Aboobaker, Derek Araujo, Joy Didier, Will Grainger, Shaul Hanany, Seth Hillbrand, Michele Limon, Amber Miller, Britt Reichborn-Kjennerud, et al. Star camera system and new software for autonomous and robust operation in long duration flights. In *2015 IEEE Aerospace Conference*, pages 1–11. IEEE, 2015.

[99] Nasa-juno star reference unit camera. `https://www.jpl.nasa.gov/images/pia24436-high-energy-and-junos-stellar-reference-unit`. Accessed: 11-01-2022.

[100] Marco Mastrofini, Gilberto Goracci, Ivan Agostinelli, and Mohamed Salim. Resident space objects detection and tracking based on artificial intelligence. In *Proceedings of the Astrodynamics Specialist Conference AAS/AIAA, Charlotte, NC, USA*, pages 7–11, 2022.

[101] Daniel L Oltrogge and Salvatore Alfano. The technical challenges of better space situational awareness and space traffic management. *Journal of Space Safety Engineering*, 6(2):72–79, 2019.

[102] Nikolaus Salvatore and Justin Fletcher. Learned event-based visual perception for improved space object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2888–2897, 2022.

[103] AI Bobrovsky, MA Galeeva, AV Morozov, VA Pavlov, and AK Tsytsulin. Automatic detection of objects on star sky images by using the convolutional neural network. In *Journal of Physics: Conference Series*, volume 1236, page 012066. IOP Publishing, 2019.

[104] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[105] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[106] Marco Mastrofini, Francesco Latorre, Ivan Agostinelli, and Fabio Curti. A convolutional neural network approach to star sensors image processing algorithms. *Astrodynamics Specialist Conference AAS/AIAA, Big Sky, Montana, USA, August 9-11*, 2021.

[107] Marco Mastrofini. Test on 20 image sequences. `https://www.youtube.com/watch?v=RGZbLHRshSI`. Accessed: 2022-06-22.

[108] Giancarmine Fasano, Giancarlo Rufino, Domenico Accardo, and Michele Grassi. Satellite angular velocity estimation based on star images and optical flow techniques. *Sensors*, 13(10):12771–12793, 2013.

[109] Dario Spiller and Fabio Curti. A geometrical approach for the angular velocity determination using a star sensor. *Acta Astronautica*, 196:414–431, 2022.

[110] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-yuan Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 07 2022.

[111] Yolov7: The most powerful object detection algorithm (2023 guide). `https://viso.ai/deep-learning/yolov7-guide/`. Accessed: 2023-09-15.

[112] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. Blendmask: Top-down meets bottom-up for instance segmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8570–8578, 2020.

[113] Malcolm David Shuster and S D_ Oh. Three-axis attitude determination from vector observations. *Journal of guidance and Control*, 4(1):70–77, 1981.

[114] F Landis Markley. Attitude determination using vector observations and the singular value decomposition. *Journal of the Astronautical Sciences*, 36(3):245–258, 1988.

[115] Hanspeter Schaub, Lee EZ Jasper, Paul V Anderson, and Darren S McKnight. Cost and risk assessment for spacecraft operation decisions caused by the space debris environment. *Acta Astronautica*, 113:66–79, 2015.

[116] Dario Spiller and Fabio Curti. A geometrical approach for the angular velocity determination using a star sensor. *Acta Astronautica*, 2020.

[117] Phoebe, school of aerospace engineering. `https://sites.google.com/uniroma1.it/stecco-sia/home`, 03 2021.