STRATEGIC: telescope movement prediction and control software for unknown orbiting object tracking

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Giulia Vitale**

Student ID: 953414
Advisor: Prof. Pierluigi Di Lizia
Co-advisors: Riccardo Cipollone, Andrea De Vittori
Academic Year: 2021-22

To my students of 5Y and to new beginnings

Abstract

Space debris problem increases year by year due to the constant increment of satellites launches for commercial purposes, which overcrowd the most operative orbits: LEO and GEO. Hence, new instruments for space surveillance, required to cope with the constant alert operative satellites, must avoid fatal collisions. Debris must be tracked, and unknown objects must be observed and added to existing catalogues, to plan manoeuvres and reentries in a safe way without risking further collisions and fragmentations. Optical ground telescopes are not the most powerful tools for precise orbit debris tracking, but they are low cost and they still provide useful information for unknown object orbit parameters estimation. STRATEGIC is a software planned to be installed on the PoliMi optical telescope. It is able to control telescope operation of image shooting and mount moving. It automatizes the process of object tracking through a convolutional neural network called YOLO which applies a detection on the acquired shots, and an algorithm of movement prediction called TEMPO. The latter analyzes the output of YOLO and predicts the camera frame for the telescope repositioning; thus pinpointing the targeted object over time. TEMPO applies a linear estimation on the satellite velocity and uses it to rapidly approximate the next position where the object is going to be found with respect to the camera frame. Due to the time consumption given by the movement of the mount and the AI detection, the most probable satellites or debris to be detected are the ones occupying LEO and MEO orbits.The process has been implemented on a virtual machine called “VM Ware” with Ubuntu operative system and runs on a 11th Gen Intel(R) Core(TM) i7-1165G7 at 2.80GHz, using Python as programming language and Pycharm as ide. The result show STRATEGIC is totally able of tracking objects for their whole visible passage and retrieve a complete set of their position in the sky during the observation time.

Keywords: Space debris, Optical Telescopes, Telescope control, Object tracking, Movement prediction.

Abstract in lingua italiana

Il problema dei detriti spaziali è in costante ascesa a causa dell'aumento negli anni di lanci di satelliti per fini commerciali, che affollano le orbite più importanti dal punto di vista operativo: quella geostazionaria, GEO e l'orbita bassa, LEO. Per questo motivo è necessaria l'implementazione di nuovi strumenti capaci di far fronte alla crescita esponenziale di allerte di imminenti collisioni che i satelliti si trovano ad affrontare per evitare danni collaterali agli strumenti di bordo. I detriti devono essere tracciati il più possibile e bisogna aumentare al massimo la catalogazione di oggetti sconosciuti, osservandoli e correlandoli con i cataloghi già disponibili. Queste operazioni possono essere fondamentali per consentire ai satelliti operativi di programmare manovre apposite di evitamento dei detriti che potrebbero causare avarie ai sistemi o manovre di rientro sicuro che evitino collisioni che sarebbero causa di ulteriori frammentazioni e detriti in orbita. I telescopi ottici presenti in stazioni a terra non sono gli strumenti più potenti a disposizione per fare osservazione di detriti spaziali, ma possono comunque dare un enorme contributo per via del loro basso costo, che permette un'osservazione di minor qualità, ma in maggiore quantità, di oggetti che orbitano intorno alla Terra. STRATEGIC è un software programmato per essere installato sul telescopio ottico presente al PoliMi ed è capace di controllare le operazioni del telescopio che consentono di scattare immagini e modificare il puntamento della fotocamera nel corso di un'osservazione. Inoltre, automatizza il processo di tracciamento di oggetti nel cielo attraverso una rete neurale convoluzionale chiamata YOLO, che applica un rilevamento della traccia lasciata dagli oggetti sulle immagini scattate dal telescopio. Inoltre, deduce informazioni utili a STRATEGIC che, tramite un algoritmo chiamato TEMPO, è in grado di effettuare una previsione sulla futura posizione nel cielo dell'oggetto in esame e seguirlo per tutta la durata del suo passaggio. TEMPO esegue una stima lineare della velocità del satellite in osservazione e la usa per approssimare rapidamente la prossima posizione dell'oggetto rispetto al campo visivo della camera. Il processo è stato implementato su una macchina virtuale chiamata "VM Ware", con Ubuntu come sistema operativo. A causa del tempo necessario alla montatura del telescopio per muoversi e al rilevamento di tracce nelle immagini eseguito dalla IA, gli oggetti che STRATEGIC saprà tracciare si troveranno principalmente nelle orbite LEO e MEO.

Il PC utilizzato è un Intel(R) Core(TM) i7-1165G7 di undicesima generazione con processore a 2.80GHz. Il linguaggio di programmazione utilizzato è stato Python con PyCharm come IDE. I risultati dimostrano che STRATEGIC è perfettamente in grado di raggiungere lo scopo per cui è programmato: ovvero di tracciare oggetti orbitanti durante il loro passaggio nel cielo ed ottenere un set completo di informazioni riguardante la loro posizione nel corso dell'osservazione.

Parole chiave: Detriti spaziali, Telescopi ottici, Controllo, Tracciamento di oggetti, Previsioni di movimento

Contents

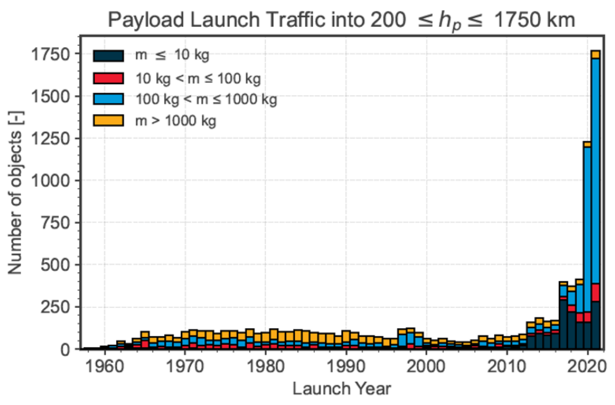
1 | Introduction

Earth pollution is a main theme affecting new generations, due to an excessive usage of natural resources for artificial purposes. Recently, even space is rapidly becoming more and more contaminated by space debris, caused by fragments of decommissioned satellites. These objects cause collateral damages to operative systems, increasing the amount of space junk and worsening the overcrowding conditions of the main orbital regimes. At PoliMi, a project has been implemented to help monitoring this situation. A space sourveillance telescope has been assembled with a Rowe-Ackermann Schmidt Astrograph optical tube [1] and an AI has been trained to detect debris passing in front of the telescope camera. STRATEGIC software, namely "Space debris TRacking Algorithm for TElescope Guidance and Instantaneous Control", main purpose is to connect the telescope control with the detection chain, allowing POSST to track even unknown space debris, actively contributing to debris catalogues updating.

1.1. Space debris problem

Over the last years, the launch of satellites has drastically increased for commercial purposes and Fig. 1.1 clearly shows the exponential increment of launched satellites in the last years with respect to the past decades. Furthermore, today's technologies foster the concept of smaller satellites in wider constellations rather than a smaller amount of larger systems. For this reason, orbit overpopulation and space pollution are becoming serious issues for new generations, resulting in an increasing number of collisions and discarded objects that generate orbital debris around the Earth. This definition includes any human-made object in orbit around the planet that no longer serves a useful function, such as nonfunctional spacecraft, abandoned launch vehicle stages, mission-related debris, and fragmentation debris [2].

Nowadays, the awareness related to space debris increment is spreading, and many precautions are adopted when launching new satellites. As an example, most (but not all) rocket bodies launched today are safely placed in compliant disposal orbits or removed from low-Earth orbit to avoid further fragmentations. Nonetheless, this trend is unsus-



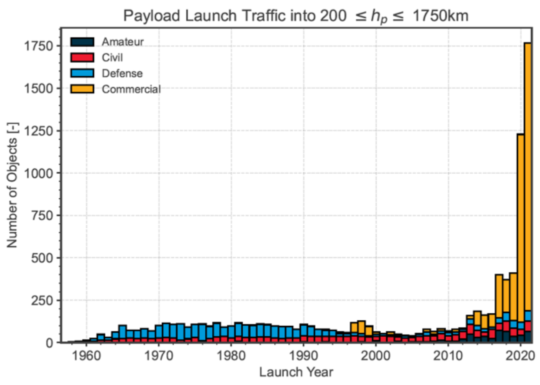


Figure 1.1: Number of launched satellites over years[3].

tainable in the long term since not enough satellites are removed from crowded orbits at the end of their life. As a direct result, each year active satellites need to dodge out of their way a growing number of derelict objects that have wondered in space for more than a decade [3]. These operations are usually performed manually, but recently many Space Safety programs carried by NASA and ESA are developing Automated Collision Avoidance technologies based on Artificial Intelligence that could improve space debris tracking and decrease the number of fragmentation events. Fig. 1.2 shows how unidentified (UI) objects exhibit a steady year-by-year growth, clearly showcasing why it is fundamental to enhance current technologies that spot and track fragments of space debris.

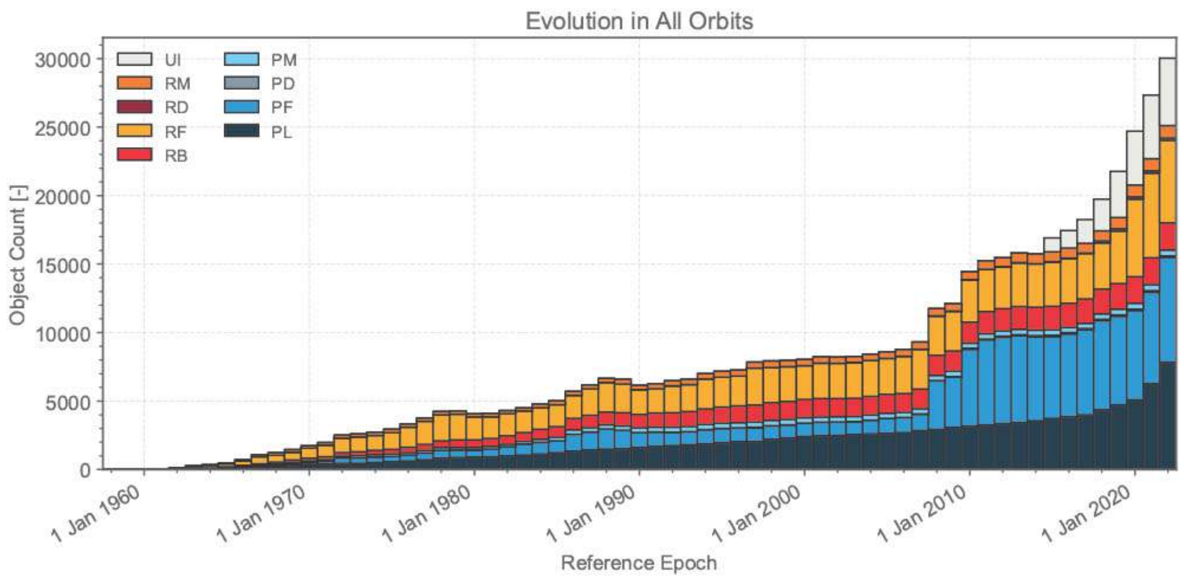


Figure 1.2: Number of space debris objects by year and origin: blue ones come from payloads, red and orange from rocket bodies [3].

1.2. Space debris surveillance

As of the date of this work, scientific models estimate that the number of objects considered as debris orbiting around the Earth are:

- about 29,000 for sizes larger than 10 cm;
- around 670,000 for sizes larger than 1 cm;
- more than 170 million for sizes larger than 1 mm.

Any of these objects are harmful for operational satellites. A 10-cm object could collide with a spacecraft causing a catastrophic damage, 1-cm debris can penetrate ISS shields and 1-mm objects can strongly reduce the effectiveness of an operational subsystem [4]. As an example, Figure 1.3 shows Hubble's solar panels that were replaced and taken back to Earth by astronauts. The cells are damaged by small holes originated by collisions with debris. Due to their high velocity, small objects act as projectiles on solar panels, destroying solar cells and drastically reducing power efficiency.

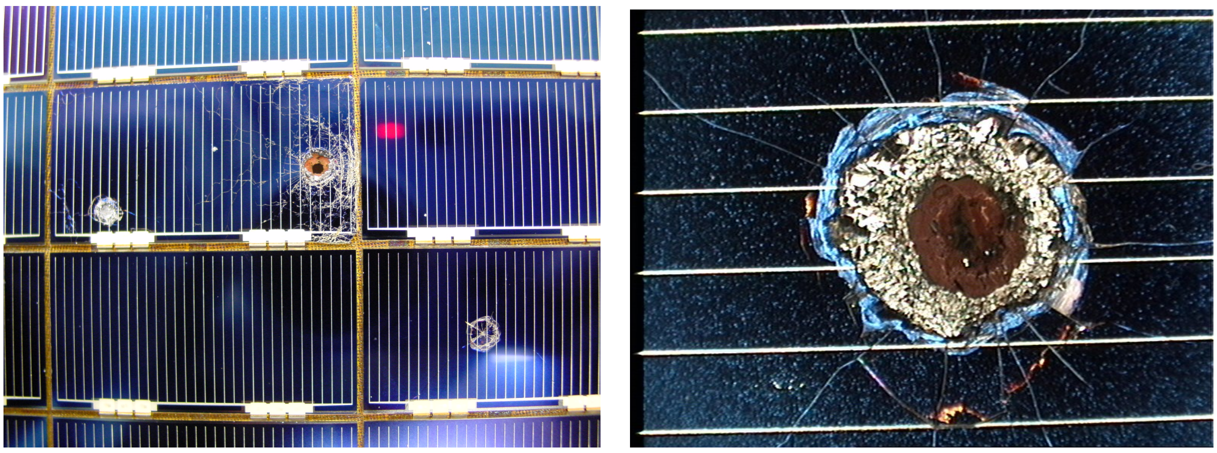

(a) Entire panel. (b) Detail.

Figure 1.3: Space debris collision damage on Hubble's solar panels[5].

Moreover, orbits which are affected the most by space debris are also the most overcrowded ones:

- Low Earth Orbit (LEO): includes all satellites orbiting between 160 and 2000 kilometers of altitude. Most of the instruments for Earth surveillance, meteorological purposes and military observations are located here, with velocities of around 7 km/s;

- Geostationary Orbit (GEO): located at 35 000 km from the Earth's surface, this orbit hosts satellites for telecommunications, meteorology and military defense, among others. Here, collisions could take place at relative velocities up to 4 km/s, due to the existence of GEO-crossing debris in eccentric orbits [6].

Clearly, it is of vital importance to take action before collisions occur in order to avoid catastrophic events such as fragmentation or explosions of operative spacecrafts. For this reason, Space Surveillance and Tracking systems (SST) are constantly upgraded by all the mayor space agencies over the world. Generally, SST systems retrieve data from ground and space-based optical and radar stations. Meteorological satellites also serves for unidentified objects monitoring. Data from sensors are processed to update a catalogue containing information about all the observed objects, i.e. orbit parameters and dimension. With a complete catalogue, SST systems can provide all the services related to the safety of operational spacecraft:

- *Conjunction Prediction System* offers collision alerts whenever an observed debris is expected to hit an operative system;
- *Reentry Prediction System* helps planning a safe reentry for decommissioned spacecraft, preventing the increment of space junk;
- *Fragmentation Analysis System* studies fragmentation events and helps build statistical models;
- *Catalogue Query System* is updated and available for all satellites. In addition, it can be used to train new automated collision avoidance technologies.

The whole process is summarized in Fig. 1.4.

In order to obtain and maintain a complete catalogue, it is necessary to grant access to as many observable objects as possible. Space-based optical sensors are significantly more effective than ground-based ones; some of their advantages are, in fact, the independence from daytime/nighttime observation, weather change or atmosphere, but also their proximity to the space environment. On the other hand, in spite of their limitations, any ground telescope could potentially contribute to space debris observation and orbit determination. In this regard, the main purpose of this work is to provide a pipeline for automatic real-time space object tracking with optical telescopes.

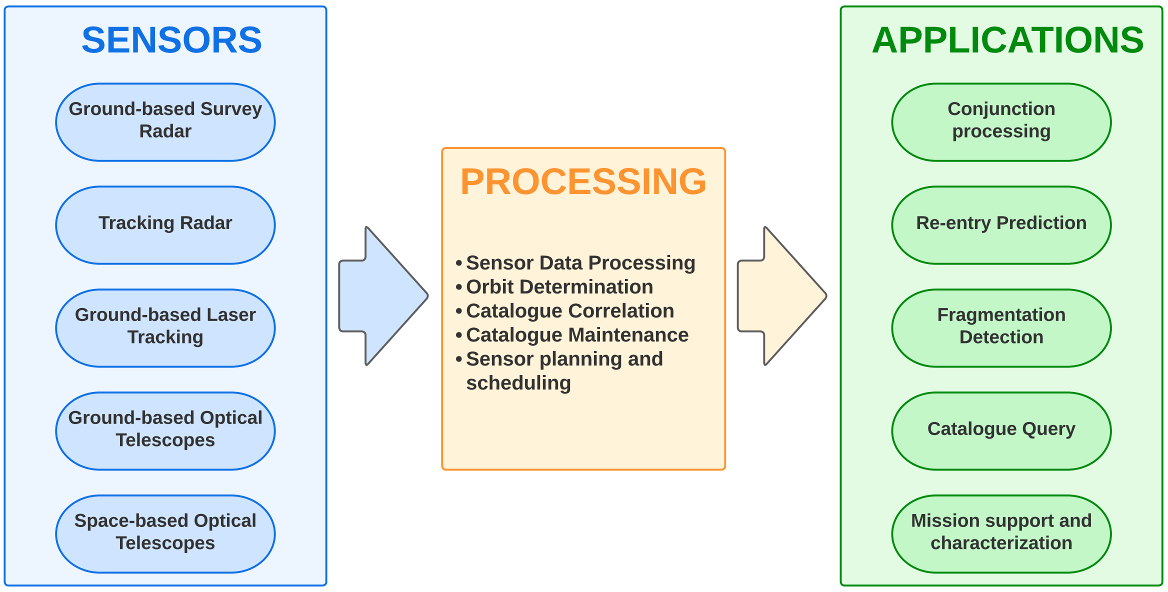


Figure 1.4: Processing pipeline of Space Surveillance and Tracking Systems [7].

1.3. State of the Art

Nowadays, the main instruments used for space debris observation and detection are radar and optical telescopes, both ground-based and space-based.

1.3.1. Optical telescopes

These instruments detect space debris using the visible spectrum, gathering photons coming from the orbiting object on CCD or CMOS cameras. This way, objects down to 10 cm can be detected. Due to this working principle, many constraints must be satisfied in order to successfully execute a detection [8]:

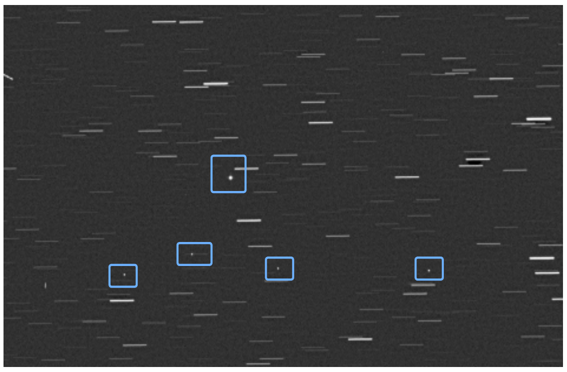
- sensitivity and Signal-to-Noise Ratio (SNR) must stay within an acceptable range. The object needs to be distinguishable from the background; hence, the telescope must stay in the dark whereas the debris must be illuminated. This particular condition only happens a few hours after twilight or before dawn with a low amount of light pollution;
- observation is strongly affected by weather conditions;
- as previously mentioned, the greatest amount of space debris is concentrated in LEO and GEO orbits. LEO objects can only be seen by sensors located at low relative latitudes, i.e. the difference between the observer's latitude and the object's must

be less than 20°. Debris in GEO can be observed only at low latitudes.

Optical telescopes operate mainly in two different modes:

Chasing mode: the telescope moves at the object's angular velocity. As a result, the debris appears on the image as single dots, while stars leave streaks on the sensor. This mode is typical for GEO-orbiting debris. An example is given in Fig. 1.5a.

Staring mode: the telescope's frame moves at the Earth's rotational speed. Specifically, stars appear as dots, whereas the debris leaves a track on the sensor. Known the exposure time and the length of the streak, the relative object's angular velocity can be inferred. A visual representation is shown in Fig. 1.5b.


(a) Chasing mode. Courtesy of [9].

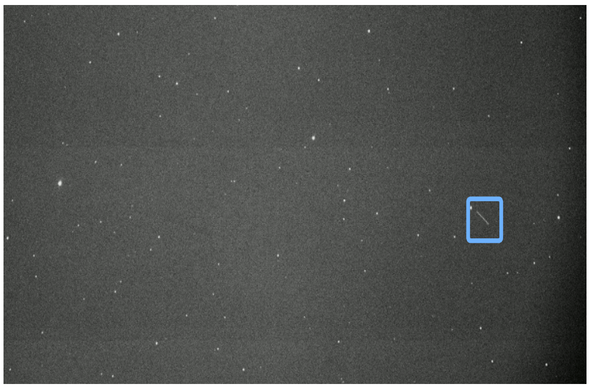

(b) Staring mode[10].

Figure 1.5: Optical telescope images. Blue boxes highlight the observed objects.

As a first approximation, the orbital parameters are deduced by measuring the object's angular velocity and assuming circular orbits. The images are then processed using Astrometry and Photometry techniques, which allow to obtain further information regarding the object's orbit and dimensions. Finally, the orbit is compared to an existing catalogue of known objects in a process called "correlation". If the debris already appears in it, the orbit parameters are updated with new data. If the object does not have any match with the available data (i.e. it is an unidentified object) an estimation of the next passage is carried out for a precise orbit determination. As of the date of this thesis, uncatalogued objects are difficult to follow, especially for staring modes, due to the lack of knowledge about their orbital parameters.

An operative ground-based telescope is mounted inside the Castelgrande observatory (Switzerland) [9]. The telescope owned by GAUSS srl is involved in space debris observation and tracking in cooperation with ROSCOSMOS. Fig. 1.5a shows the images shot by the telescope in chasing mode. Fig. 1.5b is about a picture shot in staring mode by the optical telescope owned by the Italian Air Force at Pratica di Mare (Italy).

CCD and CMOS cameras

Regarding the detector technology usually employed in observation campaigns, the most widespread are CCD and CMOS cameras. A brief description of the two instruments is disclosed hereafter. The PoliMi telescope mounts for sky image acquisitions a CMOS camera, even though the software implemented during this thesis uses a CCD camera with adequate parameter settings for resembling a CMOS device in a simulation process.

- a CCD camera - namely Charged Couple Device - usually ensures high quality and low noise level, and it provides a high dynamic range. This device is made out of a grid of pixels, each one containing a potential well. During the exposure, as light strikes the sensor, this potential well collects photons for the whole exposure time and releases electrons. The electrons, in turn, move down each row of the CCD after the end of acquisition, generating an electrical current. The current flows through an amplifier and converts into a voltage value proportional to the amount of light collected by each pixel. [11]
- a CMOS camera is a metal oxide semiconductor sensor. Even CMOS are constituted by a grid of pixels with a potential well; nonetheless, each pixel is read out in parallel and not row by row, as shown in Fig. 1.6. Thus, CMOS sensors can be much faster than CCD cameras and output low noise shots due to multiple acquisition system, instead of the single one used by CCDs. Furthermore, their power consumption is lower than CCD devices.

1.3.2. Radar Tracking

Radar telescopes are more effective than optical telescopes in detecting debris in LEO. They can operate with no dependence on the weather or day/night condition with a spatial resolution down to 2 cm from a distance of 1000 km [13]. For space-object tracking, radar telescopes usually operate in two different modes:

Target-Directed Observation generally has a priori information available, such as some orbital elements and the approximate size of the object (radar cross-section). In this sense, the radar's telescope beam is oriented to observe an object during a transit and collect measurements for orbital parameters update and debris radar signature. As a result, it is able to collect further precise orbital parameters, as

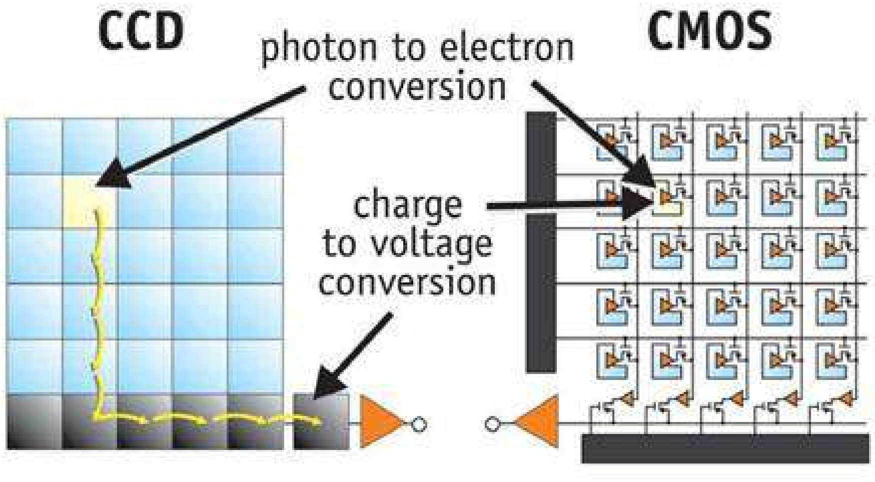
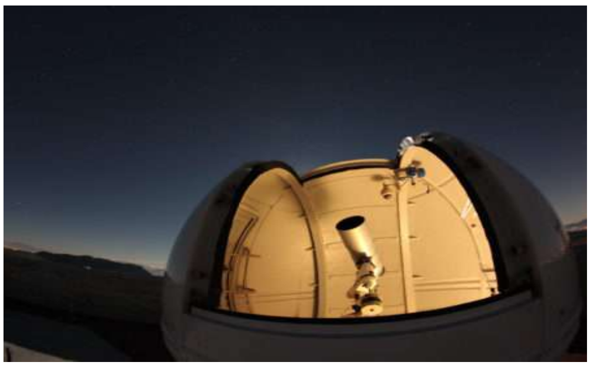


Figure 1.6: CCD and CMOS sensors. Courtesy of [12].

well as the debris radar signature. The latter comes from the registered back scatter of the object to the microwaves sent by the transmitter, providing information such as debris dimension, rotation or tumbling rate. This mode is used when an object is known but still with an unacceptably high uncertainty, for instance in case of collision-avoidance manoeuvres or reentry predictions for potentially dangerous objects.

Beam-park experiments When this mode is used, the radar beam is kept in a fixed direction in space, so that the beam scans a 360-degree portion of the sky on a daily basis. The debris' back scatter is then measured and the object's orbital parameters and dimensions are determined. This mode is intended for statistical model validation and to retrieve information about unknown orbiting debris.

The ESA owns a Radar Telescope for Space debris Observation and Tracking called TIRA, which is shown in Fig. 1.7b.


(a) CastelGAUSS Optical Observatory at Castelgrande. Courtesy of [9].

(b) TIRA Radar Telescope in Wachtberg, Germany[14].

Figure 1.7: Ground-based telescopes.

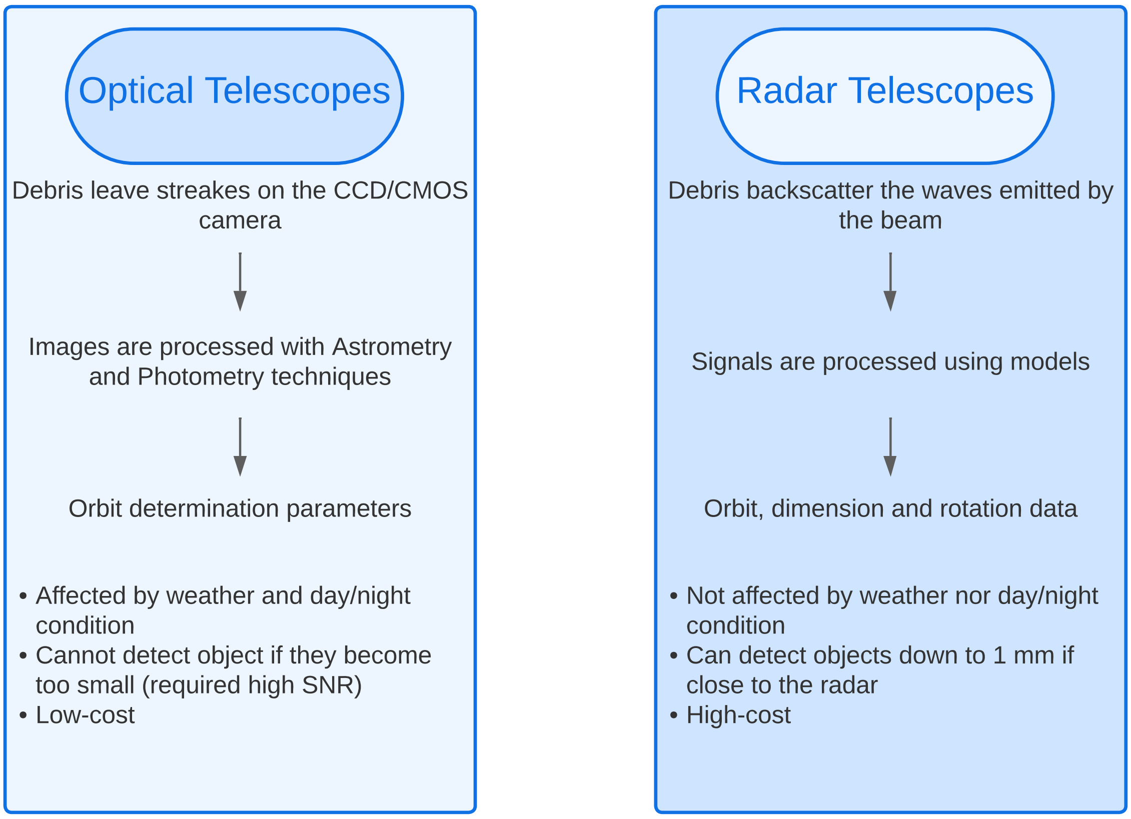

Optical Telescopes
Debris leave streakes on the CCD/CMOS camera
Images are processed with Astrometry and Photometry techniques
Orbit determination parameters
• Affected by weather and day/night condition
• Cannot detect object if they become too small (required high SNR)
• Low-cost
Radar Telescopes
Debris backscatter the waves emitted by the beam
Signals are processed using models
Orbit, dimension and rotation data
• Not affected by weather nor day/night condition
• Can detect objects down to 1 mm if close to the radar
• High-cost

Figure 1.8: Comparison between optical and radar Debris observation.

1.3.3. New Artificial Intelligence techniques

Up to the date of this work, image processing for astrometric reduction has always been carried out after recollecting all the images and signals from the observation phase. Meanwhile, new artificial intelligence neural networks claim to be faster and more accurate. For this reason, Space Agencies are considering the idea of developing automated collision avoidance manoeuvres. These are conceived to the idea of process the image and to identify the menacing object directly in space, using Machine Learning and Deep Learning tools directly installed on the spacecraft. The same concept could be thought at ground-based level, by tracking and gathering more data even with unknown objects over a single passage. The application of this kind of technique to optical ground telescopes would allow, for example, to exponentially increase unknown space debris observation thanks to this inexpensive instrument.

1.4. Proposed Approach and Thesis Purpose

One of the newest and fastest Neural Networks for image detection is YOLO (You Only Look Once), the first one-stage detector in the deep learning era. The algorithm works by dividing the image into different regions in order to analyze them separately. It creates bounding boxes around the detected object, hence predicting the probability of it belonging to a certain class [10]. At Politecnico di Milano, YOLO has been implemented for debris track detection on astronomical images. Furthermore, PoliMi owns an optical telescope, whose mount has been modified so as to allow for movements at a maximum speed of $4°/s$ [1]. This makes it suitable for following objects orbiting also in LEO. and can be remotely controlled via Python's indiserver libraries. The aim of this thesis is to create a tracking and control software able to shoot multiple images of unknown objects, chasing them the whole observable passage. This software acts as a virtual telescope, shooting synthetic images. The acquisitions are then analyzed by YOLO to localize the target in FoV. If two images show the same object, its relative angular velocity can be estimated and the telescope moves in order to spot the target in real-time.

1.5. Workflow

Figure 1.9 shows how the telescope can follow unknown objects thanks to STRATEGIC software. Figure 1.9 shows the general workflow that an optical telescope should be able to follow to track unknown objects, according to this work. Green blocks state for telescope commands, orange is for detection and blue blocks represent the tracking and control algorithms. The red section, except for "FITS to PNG conversion", is only considered in case of virtual simulations of object tracking. Indeed, the procedure has been simulated in a virtual environment. Hence, some additional passages have been taken into account:

1. Synthetic images must be converted from .FITS to .PNG format;
2. Object tracklets must be printed on the image with the objects angular coordinates over different passages. To do so, an algorithm called TIG has been implemented [15] and it is deeper investigated in Section 2.6. Furthermore, TIG requires a SCOOP file with the object's TLE (Two-Line-Element): these two object dataset are explained in Section 2.5 and Section 2.4;
3. The code must follow a fake time starting from initial observation time.

In a real condition, the "synthetic image generation" block adds an image processing phase to the FITS to PNG conversion.

The Thesis is structured as follows. First, Chapter 2 introduces and explains all the tools used in the development of the virtual telescope environment. Tailor-made Python libraries simulate the telescope movements and generate synthetic night sky images with tracklets. Then, Chapter 3 presents STRATEGIC and how it is able to track unknown objects by fully analyzing each module of which it is composed. Next, Chapter 4 shows and analyzes the obtained results. Finally, conclusions and possible future developments are drawn.

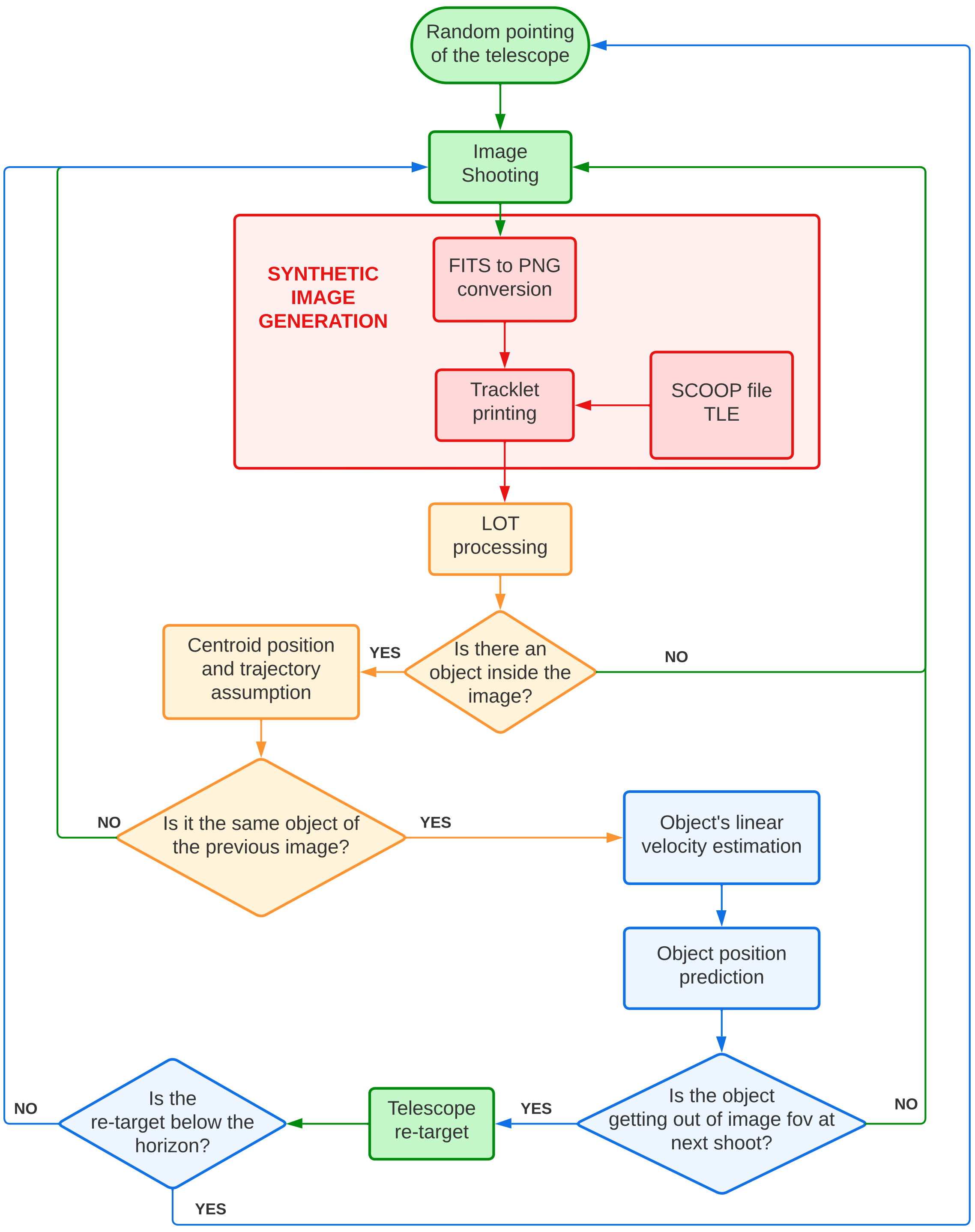

Random pointing of the telescope
Image Shooting
SYNTHETIC IMAGE GENERATION
FITS to PNG conversion
Tracklet printing
SCOOP file TLE
LOT processing
Is there an object inside the image?
YES
NO
Centroid position and trajectory assumption
Is it the same object of the previous image?
NO
YES
Object's linear velocity estimation
Object position prediction
Is the object getting out of image fov at next shoot?
YES
NO
Telescope re-target
Is the re-target below the horizon?
NO
YES

Figure 1.9: STRATEGIC main flowchart. Red section is only considered in a virtual environment.

2 | Fundamentals

This thesis aim is to provide a control software for the PoliMi telescope. In order to accomplish this mission, a particular environment has been utilized; the software runs in a virtual machine with Linux as operative system, since the Python library involved in the telescope control block is not fully developed for Windows. The chapter hereafter explains all the softwares, tools and file formats used to build STRATEGIC and implied in its operations.

2.1. Telescope Simulation

Before moving to Python codes, the frontend graphical interfaces of the telescope simulation softwares have been utilized to understand how to manage the astronomical output files format. This section provides a discussion regarding the sky simulation and the astronomical files properties.

2.1.1. Kstars and Ekos

Kstars is a free astronomy software. It involves a real-time simulation of the night sky and can be adjusted for any location on Earth. It provides an observation planner, a sky calendar tool and an FOV editor. Kstars has also educational purposes by adjusting simulation speeds for long-timescale phenomena[16]. The software displays a simulated night sky dome as in Fig. 2.1, together with the local artificial horizon. If a particular object is selected with the cursor, its celestial coordinates are displayed both in Azimuth-Elevation or Right Ascension-Declination.

Kstars is useful to plan observations of celestial objects, but it also provides a feature called Ekos, a complete astrophotography package with few external dependencies. Ekos is available on Linux®, Microsoft® and Mac® OS. It can communicate with INDI-compatible devices ("Instrument Neutral Distributed Interface") and and it designs Kstars observatory's automation and control. It manages telescopes, CCD (& DSLRs), filter wheel, focuser, guider, adaptive optics unit and any INDI-compatible auxiliary device [17]. Ekos'

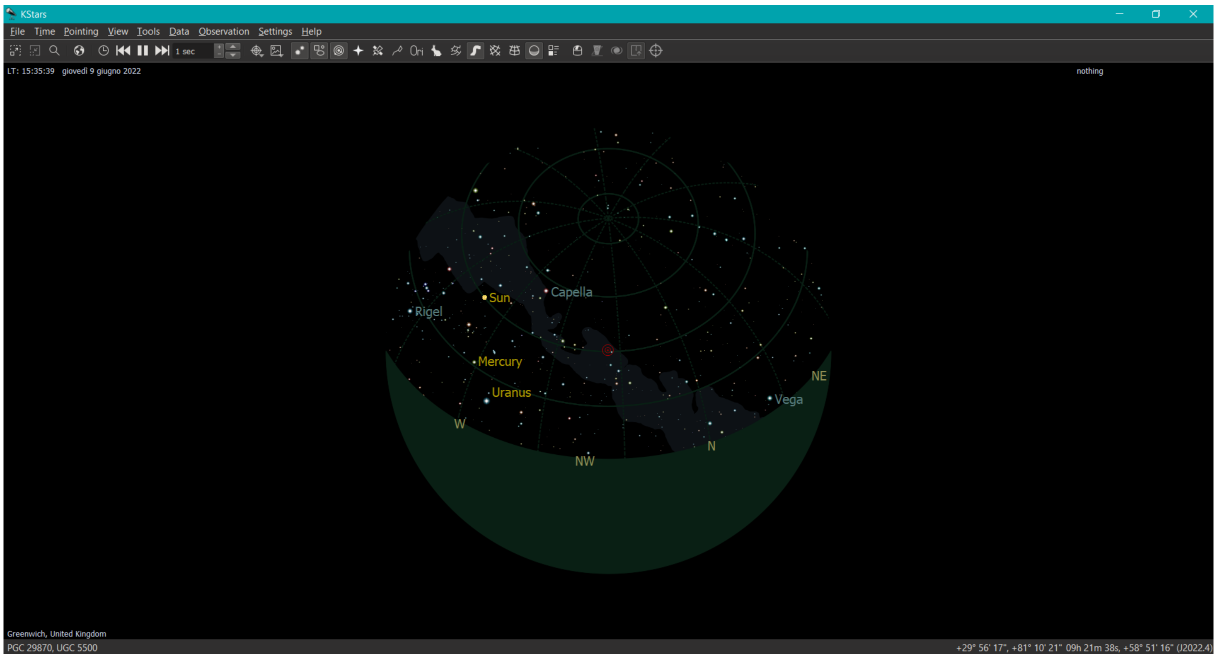
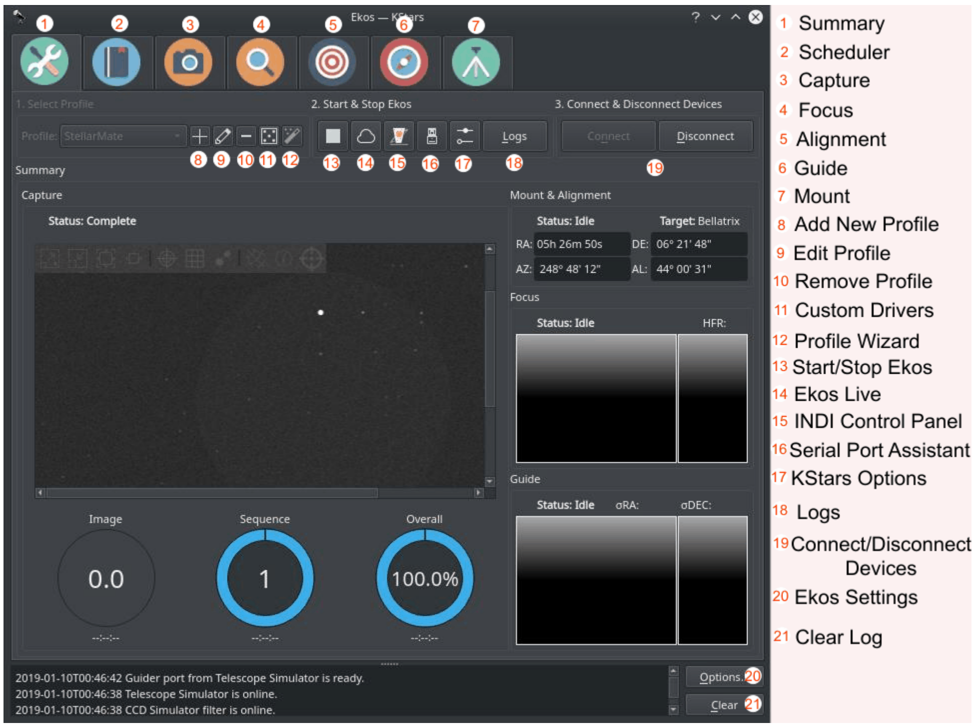

KStars
File Time Pointing View Tools Data Observation Settings Help
LT: 15:35:39 giovedì 9 giugno 2022
nothing
Capella
Sun
Rigel
Mercury
Uranus
NE
Vega
W
N
NW
Greenwich, United Kingdom
PGC 29870, UGC 5500
+29° 56' 17", +81° 10' 21" 09h 21m 38s, +58° 51' 16" (J2022.4)

Figure 2.1: Kstars graphical interface command window.

main features are:

- online and offline highly accurate GOTOs for telescope control;
- load & slew function: if a .FITS image is loaded, Ekos can point the telescope to the image's coordinates;
- polar alignment assistant at the beginning of the observation session;
- automated scheduler to control all the equipment;
- capture and record video streams in ".ser" format, which is used to store astronomical captures about image frames and metadata, such as timestamps, exposure time, instrument, telescope and observer [18].
- automated focus between exposure.

Moreover, Ekos is divided into several *modules*, related to alignment, focus, schedule and mount setups. The settings can be changed from Ekos even when the INDI server is used. The cheatsheet in Fig. 2.2 represents the main commands organised and set for photographs before using the INDI server and starting the implementation of STRATEGIC.



```
2019-01-10T00:46:42 Guider port from Telescope Simulator is ready.
2019-01-10T00:46:38 Telescope Simulator is online.
2019-01-10T00:46:38 CCD Simulator filter is online.
```

Figure 2.2: Ekos graphical interface cheatsheet.

2.1.2. Night Sky Images

The **CAMERA** module in Ekos is the one pertains image capture (number 3 in modules shown in Fig. 2.2). Here, the temperature and filters of the camera can be set to look as close as possible to the user's CCD camera. Even though telescope images are nearly always greyscale, they may contain some color information by setting a particular color filter. In fact, each color filter has a distinct sensitivity to various wavelengths. Furthermore, the size and exposure of the images changes according to the user's desire. If Ekos is not connected to a real camera and telescope, the result is a synthetic frame of the sky, resembling a real shot. This picture is created by means of mathematical modelling computations of compiled data. The difference between a real image and a synthetic one lies in the absence of noise, vignetting and cloud interference. The result is a simple two-tone picture, with black as background and white representing the dot stars. Nonetheless, it could be extremely useful for testing tracking algorithms. Figure 2.3 is an example of synthetic night sky image, taken with Ekos tool from Kstars in FITS format.

Both the real and synthetic images are saved in .FITS - Flexible Image Transport System - format, a common file for astrophotography custom-made for storing images up to 64 bit of color depth and the associated metadata. Not only does it have information about photometric and spatial calibration, but also it comprises size, origin, coordinates,

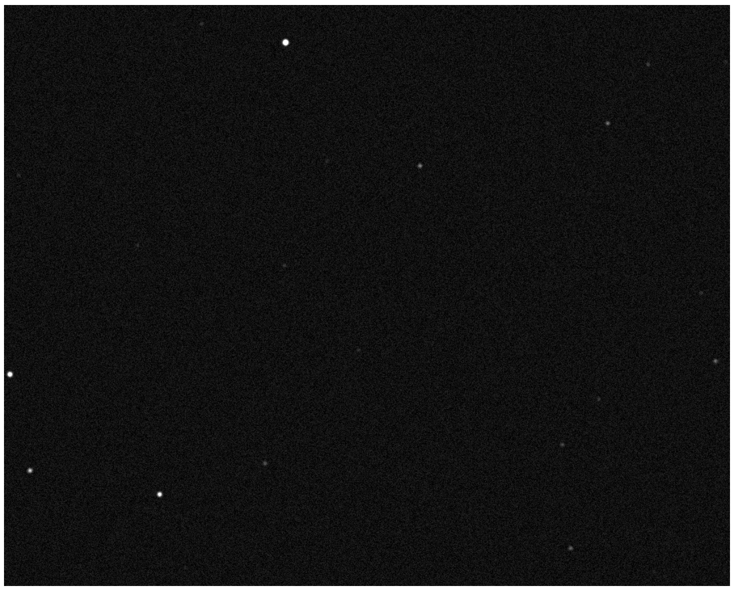


Figure 2.3: Synthetic night sky image from Ekos.

exposure and data history of the shot. The resulting images contain an implicit Cartesian coordinate system that describes each pixel location, with a multiple-dimension table, as outlined in Fig. 2.4.

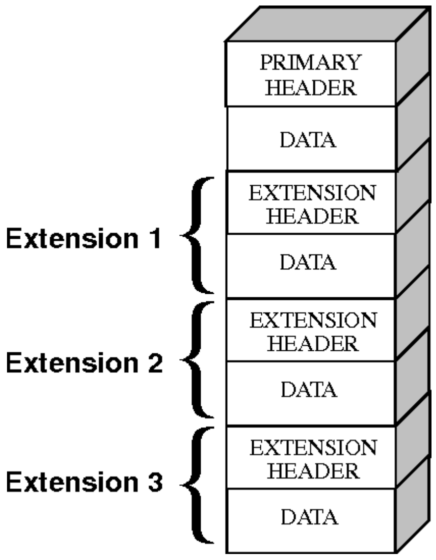


Figure 2.4: FITS image data structure.

Additionally, FITS data allows to save also non-image data, like spectra, photon lists or data cubes. The main software used to visualize this kind of file is **FITSliberator** [19], whose command window is shown in Fig. 2.5. Moreover, this software allows to save a FITS image copy in PNG format, more suitable for computer visualization, as done for Fig. 2.3.

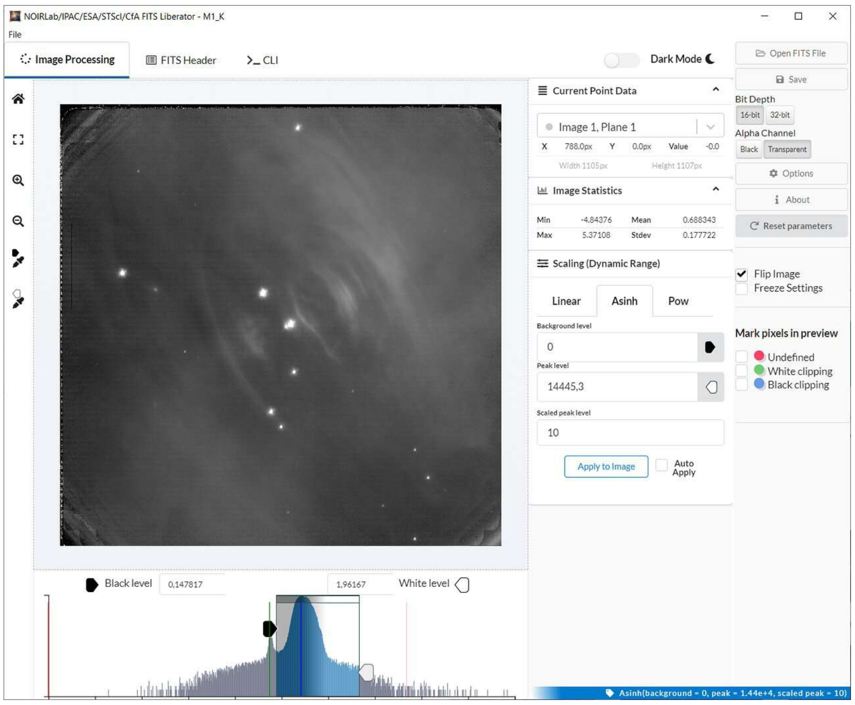


Image 1, Plane 1
X 788.0px Y 0.0px Value -0.0
Width 1105px Height 1107px

Min -4.84376 Mean 0.688343
Max 5.37108 Stdev 0.177722

Background level 0
Peak level 14445,3
Scaled peak level 10
Auto Apply

Bit Depth [✓ 16-bit] 32-bit
Alpha Channel Black [✓ Transparent]
✓ Flip Image
Freeze Settings
● Undefined
● White clipping
● Black clipping

Black level 0,147817
1,96167 White level

Asinh(background = 0, peak = 1.44e+4, scaled peak = 10)

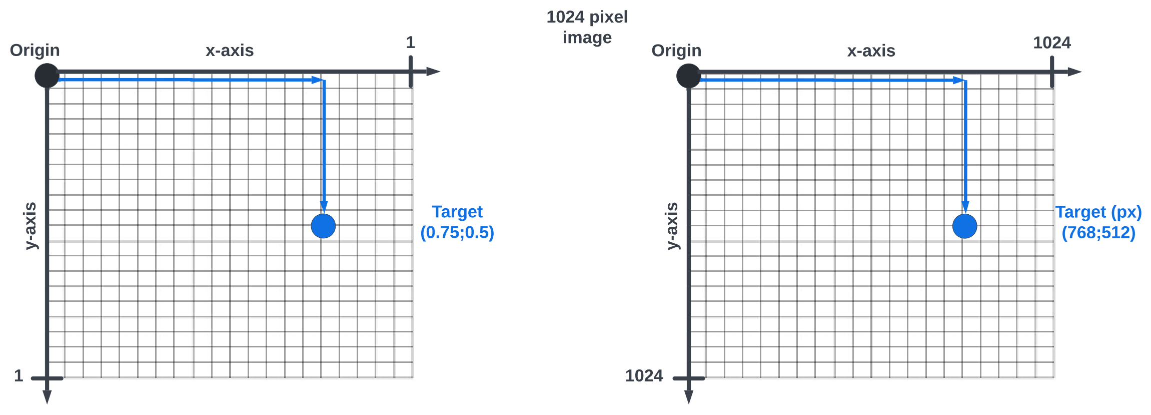
Figure 2.5: FITSliberator command window.

The software can be downloaded for free and it is employed by NASA and ESA. As it is possible to notice, the observable wavelength scale can be adjusted in dependence on the purpose of the image and for aesthetic reasons. Many programming languages are able to read data from .FITS format, i.e. Matlab® or Python, with a dedicated library called Astropy. FITSliberator cannot communicate properly with Python; thus, for the aim of this thesis synthetic FITS images are generated and down-converted to 8-bit PNGs via Python to make them compatible with YOLO (described in detail in Section 2.7).

Images coordinate system

Python reads images as matrices of pixels. Therefore, whenever an image is loaded, any target is identified with an "$x - y$" Cartesian reference system with the origin in the upper left corner, as if it was a 2D matrix. Images are not constrained in a specific size, although a square format of 512 or 1024 pixel is recommended, as it will be explained later on. Image sizing can be chosen when the FITS-to-PNG transformation is applied. As a consequence, the Cartesian reference system for target identification can be made non-dimensional. A representation of the reference frame is given in Fig. 2.6a. This way, any object inside the image field of view is described with x and y coordinates between [0, 1] values. Exceeding these boundaries means being out of the sensor FoV. This kind of reference system comes handy to predict whenever a target is exiting the FoV and for telescope repointing.

By multiplying the non-dimensional value by the resolution, the same coordinates in number of pixels can be retrieved (Fig. 2.6b).



(a) Non-dimensional example. (b) Pixel-dimensional example.

Figure 2.6: Cartesian coordinates used for target positions inside images.

2.2. Virtual Machine Environment

A Virtual Machine (VM) leverages a software instead of a physical computer to run programs and deploy apps. The physical machine is called "host" and can run one or more virtual "guest" machines. Each virtual machine runs its own operating system and functions separately from the physical machine and other VMs, even when they are all running on the same host. More recently, public cloud services are using virtual machines to provide virtual application resources to multiple users at once, for even more cost efficient and flexible compute. The aim of this thesis is to enable remote control for a telescope. For this reason, VM usage is justified by the requirement of using LINUX-Ubuntu and of managing the device remotely. The software for the virtual machine is VMWare workstation 16 player, while the installed operative system is Ubuntu 64-bit.

2.2.1. Pycharm

The PoliMi telescope system is still in development and the chosen programming language is **Python**, particularly versatile since many libraries are available and easy to use. In addition, the Python community is incredibly active: many solutions to eventual errors are available online; new features are constantly developed, and library bugs are being fixed over time. For Python coding, an IDE (Integrated Development Environment) allows multiple functions like building, executing, and debugging tools.

To achieve the purpose of this thesis, PyCharm was chosen as an IDE. Its main advantages are:

- automatic saving and reloading of code files;
- possibility to run directly codes from PyCharm environment and parallel run of two or more codes;
- possibility to run two or more codes simultaneously;
- availability of advanced debugging features;
- syntax highlighting and automatic code formatting.

On the downside, PyCharm is not the fastest IDE available for Python [20]. In this regard, one of the possible future developments for this work could be running the tracking algorithm codes created on PyCharm with faster IDEs or Google Colab. Indeed, it allows anybody to write and execute arbitrary python code through the browser, using more powerful cpu/gpu than the ones in the local PC.

2.3. INDI Library

INDI Library is an open source software to control astronomical equipment. It acts as a bridge between software clients and hardware devices [17]. Currently, Linux, BSD, and OSX are supported, while Windows support is under progress. For this reason, a virtual machine station handling Linux operative system has been employed. INDI library can control any INDI device, exactly as Ekos, but they are used in different ways:

Ekos is the frontend GUI astrophotography tool inside KStars. It performs capture, focusing, guiding, and scheduling. Ekos depends on INDI for device control.

INDI is the backend server used to communicate with astronomical devices and control them.

For the aim of this thesis, INDI server controls a virtual telescope and camera, but generally it allows to communicate also with domes, focusers, adaptive optics, weather stations and raspberry pi.

INDI library is available on many programming languages, such as Java, Python and C.

2.3.1. Architecture and properties

INDI[1] server is the public network access point where one or more clients may contact one or more drivers, as shown in Fig. 2.7. The INDI server connects clients and drivers by means of a "protocol", carrying information from one to the other.

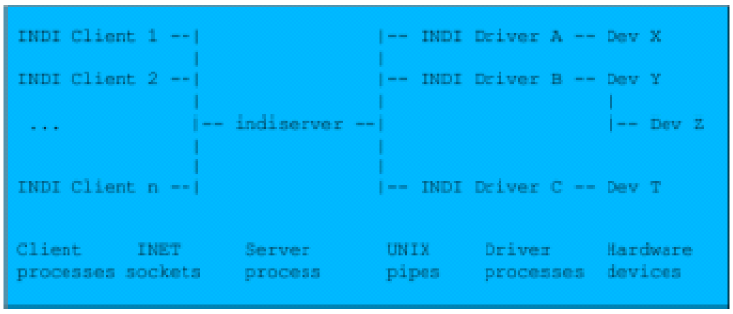


Figure 2.7: INDI main architecture [17].

The main passages the user needs to complete in order to obtain an operational process using INDI server are:

1. Define a list of properties that describe the device operation.
2. Initialize properties initial state.
3. Connect the client.
4. When a client connects, send a list of existing properties.
5. Implement functions that will carry out the operations offered by the properties.
6. Report device status to client if desired.
7. Clean up the driver if the client disconnects.

When the STRATEGIC implementation work started, INDI server codes were available and a first set of shooting and moving tests were inasmuch developed in-house. Even client and device properties were set beforehand. Thus, the operation carried out are related to point 2, 3 and 5.

- Before running the code, the device must be connected with dedicated terminal instructions
- At the very beginning of the operation, client and device properties are set. The file "config.yaml" contains the telescope and CCD simulator's name. Whenever a real

[1]All the information included into this chapter come from the INDI website: indilib.org

telescope needs to be connected, its name must be inserted instead of the current one.

- A Class "Telescope" stores all the main properties of the client and device, such as the telescope, CCD, host and port names and images exposure time.
- In addition, variables that need to be updated during the tracking phase can be easily recalled and replaced with new values during operations, i.e. telescope Right Ascension and Declination and image counter indicator. Furthermore, two functions were implemented: the first one, `Scope_shooting()`, gathers the commands required for shooting and saving a synthetic night sky image, and it is called whenever the telescope takes a new picture. Later on in this thesis, this order is identified on block diagrams by means of a camera icon. The second function, `Scope_moving()`, moves the telescope to the desired Right Ascension and Declination and is represented in block diagrams shown in next chapters with a telescope mount icon.

It is worth pointing out that any of the operation carried out by the INDI server simulates the characteristic times of the telescope routines. For instance, the function `Scope_shooting()` will require that amount of time before ending the process.

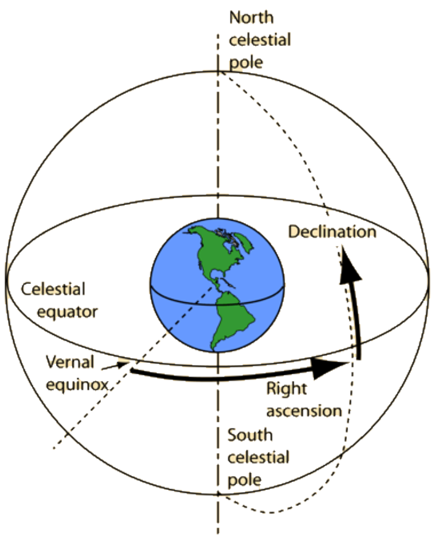
INDI server coordinates

The INDI server operates the telescope using the Ra-Dec coordinate set, a widely used reference system for astronomical observation. It is an Earth-centered equatorial coordinate system, based on:

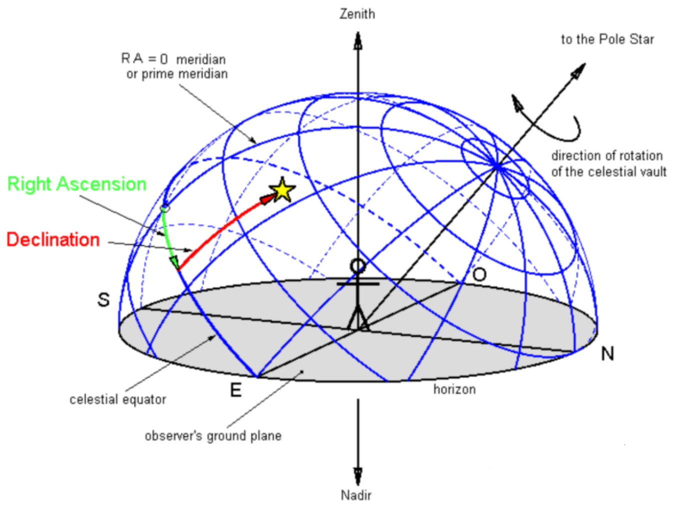
Right ascension : goes from 0 deg to 360 deg. The 0 deg point is called "Vernal equinox" and it coincides with the intersection between the equatorial plane and the ecliptic (see Fig. 2.8a).

Declination : goes from -90 deg to 90 deg, where the positive and negative sign indicates respectively a location above or under the equatorial plane.

Fig. 2.8 shows with a graphical representation how Right Ascension and Declination are defined. On the left, the picture is Earth-centered and equatorial plane is highlighted, while on the right, a representation of the coordinates from a local observer point of view is given, typical of ground telescope pointing.



(a) From Earth [21].



(b) From a local observer [22].

Figure 2.8: Right Ascension-Declination.

2.4. TLE

A TLE file is usually a txt file containing information about an orbiting space object. TLE stands for "Two-Line Orbital Element Set", and it retrieves the Ephemeris of any catalogued object with respect to a desired reference system. TLEs, devised by NASA, are extracted by an online catalogue and constantly updated.

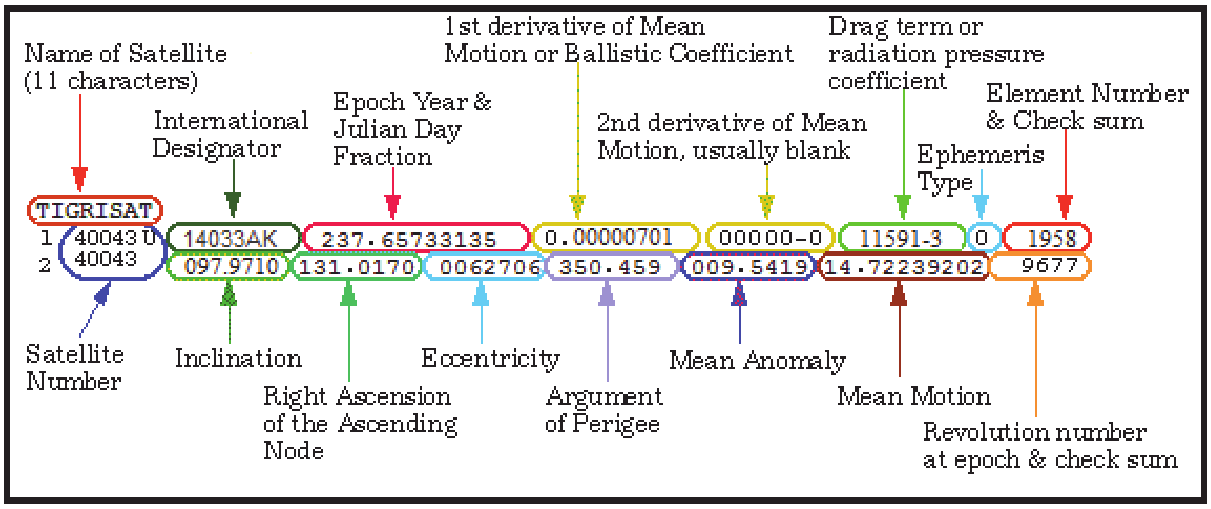

Name of Satellite (11 characters)
International Designator
Epoch Year & Julian Day Fraction
1st derivative of Mean Motion or Ballistic Coefficient
2nd derivative of Mean Motion, usually blank
Drag term or radiation pressure coefficient
Ephemeris Type
Element Number & Check sum
TIGRISAT
1 40043U 14033AK 237.65733135 0.00000701 00000-0 11591-3 0 1958
2 40043 097.9710 131.0170 0062706 350.459 009.5419 14.72239202 9677
Satellite Number
Inclination
Right Ascension of the Ascending Node
Eccentricity
Argument of Perigee
Mean Anomaly
Mean Motion
Revolution number at epoch & check sum

Figure 2.9: **TLE** example [23].

2.5. SCOOP file

TLEs encodes a list of orbital elements for an Earth-orbiting object, making them particularly helpful for orbital analysis and computation. That said, to simulate a transit over an aground station, the object states are rotated into a sensor-based reference frame and expressed with dedicated angle coordinates. For this reason the Department of Aerospace Science and Technology developed an application called **SCOOP**, (namely SpaceCraft and Objects Observation Planning). The software is able to compute the satellite passage from a ground station given its Two-Line Element Set. At the end of the process, it saves data in a txt file, convenient for further computations. SCOOP software is capable of retrieving multiple coordinates related to the observed objects, such as:

- Coordinates:

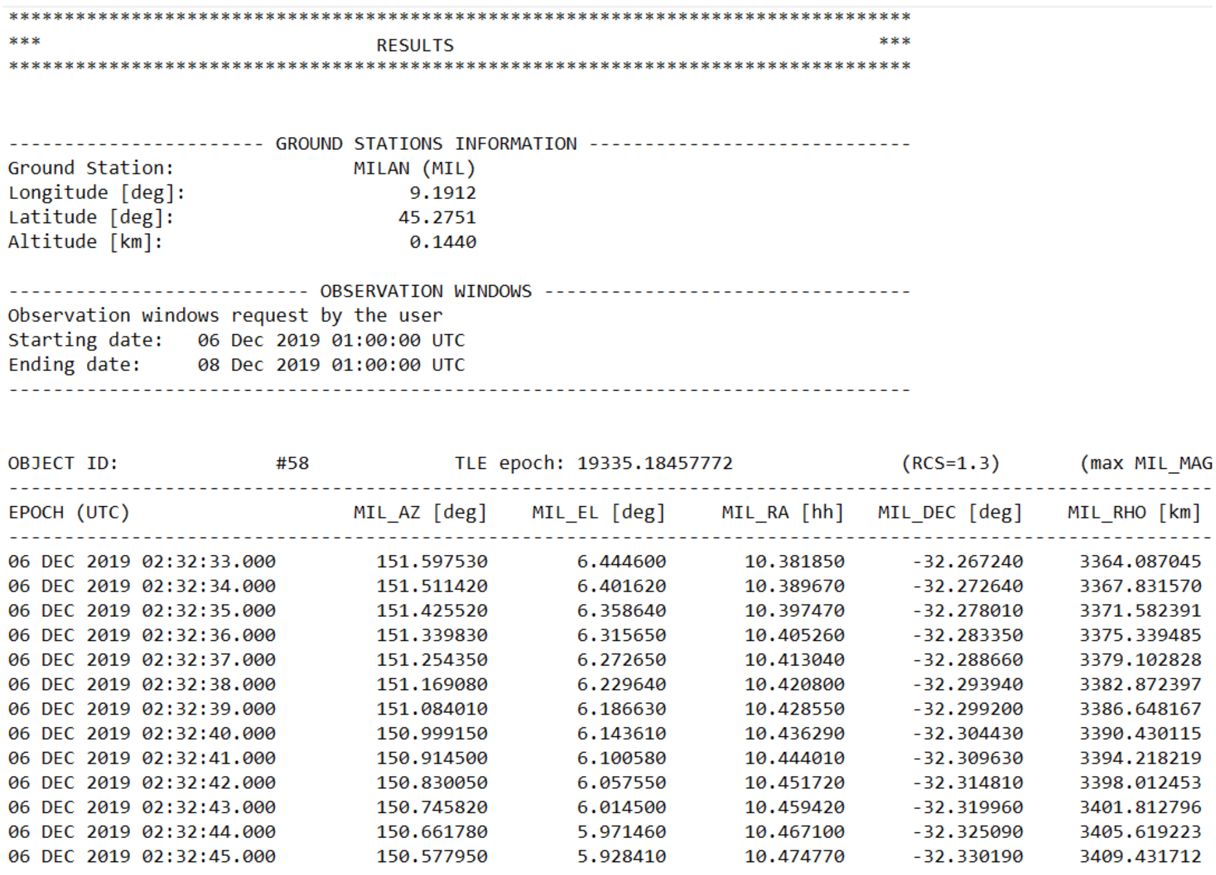
  Azimuth-Elevation

  Right Ascension-Declination

  Latitude-Longitude

- Angular distance from the Sun and the Moon;
- Illumination condition and phase angle;
- Doppler Shift, Slant range.

A sample SCOOP file is depicted in Fig. 2.10 for clarification purposes. As it is possible to notice, just the Local coordinates are shown since the other information was not used during the process. The SCOOP file fundamental role will be explained later on in Section 2.6.



```
*******************************************************************************
***                              RESULTS                                    ***
*******************************************************************************


---------------------- GROUND STATIONS INFORMATION ----------------------------
Ground Station:                MILAN (MIL)
Longitude [deg]:                    9.1912
Latitude [deg]:                    45.2751
Altitude [km]:                      0.1440

-------------------------- OBSERVATION WINDOWS --------------------------------
Observation windows request by the user
Starting date:    06 Dec 2019 01:00:00 UTC
Ending date:      08 Dec 2019 01:00:00 UTC
-------------------------------------------------------------------------------


OBJECT ID:             #58             TLE epoch: 19335.18457772             (RCS=1.3)       (max MIL_MAG
---------------------------------------------------------------------------------------------------------
EPOCH (UTC)                    MIL_AZ [deg]    MIL_EL [deg]     MIL_RA [hh]   MIL_DEC [deg]    MIL_RHO [km]
---------------------------------------------------------------------------------------------------------
06 DEC 2019 02:32:33.000         151.597530        6.444600       10.381850      -32.267240     3364.087045
06 DEC 2019 02:32:34.000         151.511420        6.401620       10.389670      -32.272640     3367.831570
06 DEC 2019 02:32:35.000         151.425520        6.358640       10.397470      -32.278010     3371.582391
06 DEC 2019 02:32:36.000         151.339830        6.315650       10.405260      -32.283350     3375.339485
06 DEC 2019 02:32:37.000         151.254350        6.272650       10.413040      -32.288660     3379.102828
06 DEC 2019 02:32:38.000         151.169080        6.229640       10.420800      -32.293940     3382.872397
06 DEC 2019 02:32:39.000         151.084010        6.186630       10.428550      -32.299200     3386.648167
06 DEC 2019 02:32:40.000         150.999150        6.143610       10.436290      -32.304430     3390.430115
06 DEC 2019 02:32:41.000         150.914500        6.100580       10.444010      -32.309630     3394.218219
06 DEC 2019 02:32:42.000         150.830050        6.057550       10.451720      -32.314810     3398.012453
06 DEC 2019 02:32:43.000         150.745820        6.014500       10.459420      -32.319960     3401.812796
06 DEC 2019 02:32:44.000         150.661780        5.971460       10.467100      -32.325090     3405.619223
06 DEC 2019 02:32:45.000         150.577950        5.928410       10.474770      -32.330190     3409.431712
```

Figure 2.10: **SCOOP** file example.

2.6. Tracklet Image Generator

Fig. 2.11 recall TIG purpose inside STRATEGIC software. As already mentioned, Kstars

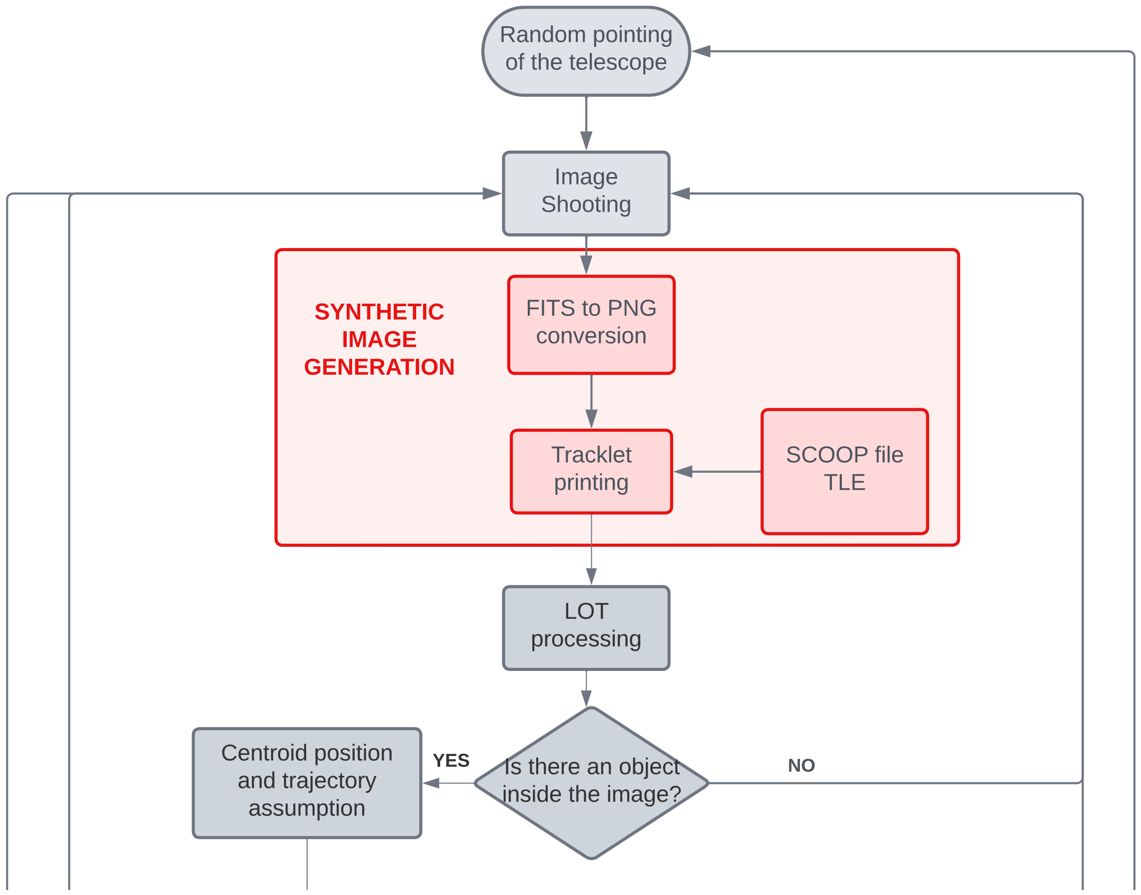

Random pointing of the telescope
Image Shooting
SYNTHETIC IMAGE GENERATION
FITS to PNG conversion
Tracklet printing
SCOOP file TLE
LOT processing
Centroid position and trajectory assumption
YES
Is there an object inside the image?
NO

Figure 2.11: Workflow section where TIG has been employed.

provides a catalogue with a great amount of celestial bodies, but no artificial object is available in Kstars dataset. For this reason, if INDI server is connected to a camera simulator, the image will never show an actual debris. Nonetheless, for STRATEGIC tracking and control implementation and validation phase, the virtual telescope tracks an already known object during its passage. If the software chases the desired object without having any prior information on the orbit, it will flawlessy apply to unknown objects. Hence, synthetic images featuring a track of an existing object are inputted to STRATEGIC for the implementation phase.

At PoliMi, a Tracklet Image Generator has been developed by Cipollone-De Vittori [2]. The code prints a synthetic streak on images resembling the tracks orbit debris leave on

[2]The whole section is based on [15]

real pictures. TIG generates a set of bespoke synthetic night sky images with random trails placement for YOLO training. During this thesis work the trail is meant to mimic a real object passage. This way, if the algorithm is able to keep the trail inside the image FOV, it can properly follow an unknown orbiting object without letting it get out of the camera frame. To achieve this purpose, TIG has been tuned to print in a “real” position the track of a space object with known coordinates. Data from real passages come from the SCOOP file, as it is summarized in Fig. 2.11 and Fig. 2.19.

2.6.1. Architecture

TIG structure is mainly divided into a set of image processing operation and tracklet printing on the image. The image processing part includes:

1. image loading
2. image conversion to a greyscale. Python treats the images as matrices where each pixel’s color is defined with a set of numbers. Since the image comes from a conversion from a fits to png and the color intensity spans from 0 to 255;
3. image resizing: the image is cropped and resized to the desired dimension, usually 1024 or, even better, 512. This resizing helps the Neural Network to improve its time performance during the detection phase. An excessive dimension of the picture would lead to unacceptable computation time for real-time detection;
4. aspect ratio adaptation consequently to the cropping and rescaling process;
5. optional noise addition.

Once the image processing is complete, the trail needs to be placed in the picture. The passages for achieving this goal are:

1. the SCOOP file is read and Azimuth angle is re-adapted if needed. In facts, a track passing by the observator’s North crosses the Azimuth 0 deg angle and this triggers a jump in the SCOOP file from 360 deg to 0 deg and vice versa, that could be problematical in a regression problem. For this reason, a conversion is actuated in the following way:
   - when the North is crossed from West to East,the coordinates are changed as

$$\mathbf{az} = [...\ 358,\ 359,\ 0,\ 1,\ 2,\ ....] \longrightarrow [...\ 358,\ 359,\ 360,\ 361,\ 362,\ ...] \quad (2.1)$$

- when the North is crossed from East to West,

$$\mathbf{az} = [...\ 2,\ 1,\ 0,\ 359,\ 358,\ ....] \longrightarrow [...\ 2,\ 1,\ 0,\ -1,\ -2,\ ....] \tag{2.2}$$

this prevents jumps when Azimuth is treated as the independent variable in a regression function where Elevation is the dependent variable.

2. An interpolation function is applied on SCOOP file coordinates, which are sampled with a time interval of 1 second, so that a value can be retrieved from the file at any time instant. The function is evaluated between the initial and final epoch, defined as the beginning and the end of the picture exposure time, and the satellite passage during the shot is retrieved.

3. Knowing the position of the satellite during the observation period, the tracklet length and slope is defined. At this point, TIG has been modified for accomplishing the requirements of this thesis work. Hence, a new function called "exact_pos()" has been created inside the class "Image_training", which was used to randomly modify noise, vignetting and tracklet position settings during YOLO network training, now updated with this new feature. This function takes as input the tracklet position deduced from the SCOOP file and the telescope position in radians. A function called "Azel_shift" computes the relative distance between the telescope pointing and the track position. Knowing that the telescope pointing coordinate is exactly the center of the image, its coordinates are:

$$T_{ad}(0.5; 0.5) \quad T_{px}\left(\frac{res_x}{2}; \frac{res_y}{2}\right) \tag{2.3}$$

where "ad" and "px" mean, respectively, in adimensional coordinates and in pixel coordinates (see Fig. 2.6 for a visual explanation), while "res" is the resolution of the image along x and y. Recalling that the relative distance computed by "azel_shift" needs to be converted in pixel, the final tracklet position has coordinates:

$$t_{x,y} = relpos * \frac{res}{FoV} + T_{px,\ x,y} \tag{2.4}$$

relpos is the relative position between the trail and the telescope pointing, multiplied by a factor that is the resolution over the FoV, and shifted from the origin of the reference frame (which is on the upper left corner) to the centre of the image, where the telescope is pointing. Thus far, the shift that applies to the tracklet drawn at point 2 and placed at its exact position. Before the end of the function, one last condition needs to be defined. If the vector containing the pixels to be colored is close to the edges, the tracklet could partially exit the FoV, exceeding the pixel range between 0 and resolution. In the following examples the two possible cases are explained, assuming image resolution as 1024 pixels.

- case 1: along x, the tracklet is exiting the FoV on the right

$$\mathbf{t_{px,x}} = [...\ 1023,\ 1024,\ 1025,\ 1026,\ 1027,\ ....] \tag{2.5}$$

  in this case no criticality are encountered. The tracklet is printed on the image until the value 1024 and all the other elements do not figure in the final image;

- case 2: along x, the tracklet is exiting the FoV on the left. In this case the pixel positions become negative. Python interprets them starting from the Right corner of the image, as in example:

$$\mathbf{t_{px,x}} = [...\ 2,\ 1,\ 0,\ -1,\ -2,\ ....] \longrightarrow [...\ 2,\ 1,\ 0,\ 1023,\ 1022,\ ....] \tag{2.6}$$

  this means that in this case the part of the tracklet exiting the FoV on the left would be printed on the right. An example is given in Fig. 2.12a.

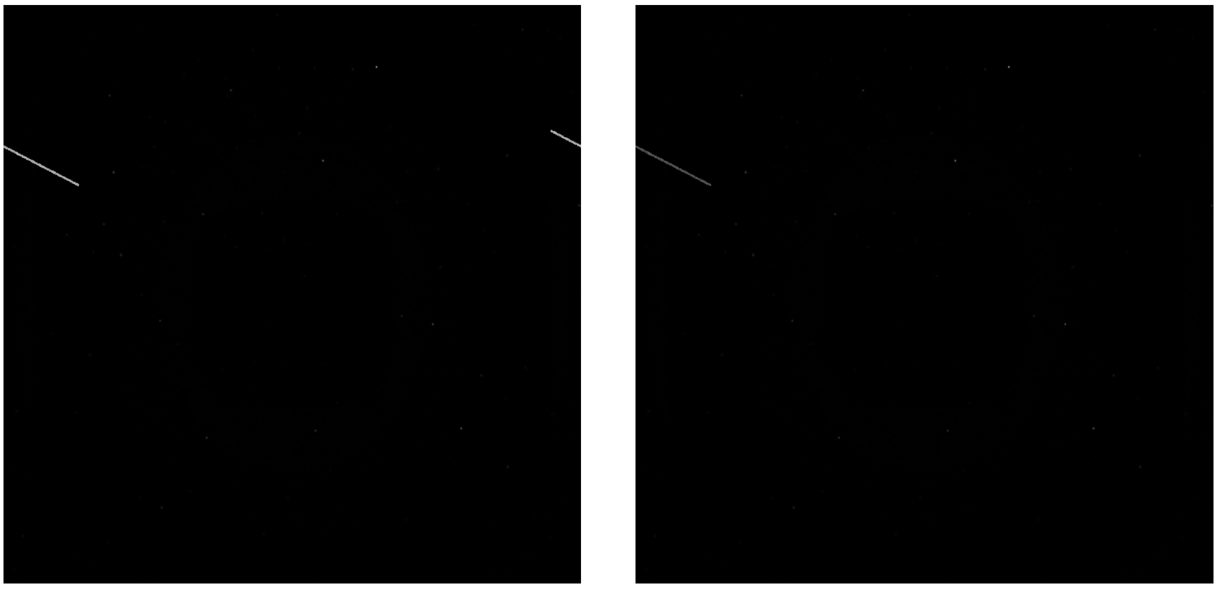

(a) Without correction. (b) With correction.

Figure 2.12: Tracklet exiting the FoV on the right.

This issue can be avoided adding an “if condition” at the end of the function, where the negative number are cut off out of the tracklet pixel vector:

$$\mathbf{t_{px,x}} = [\dots\ 2,\ 1,\ 0,\ -1,\ -2,\ \dots.] \longrightarrow [\dots\ 2,\ 1,\ 0] \tag{2.7}$$

The y-axis vector must be equalized in length to the $\mathbf{t_{px,x}}$. The result is shown in Fig. 2.12b.

Finally, a block diagram of TIG working structure is given to summarize the points mentioned in Fig. 2.13. Masks creation block is not directly used by STRATEGIC, but masks have been involved in results representation in Chapter 4 for a better visualization of the tracklets.

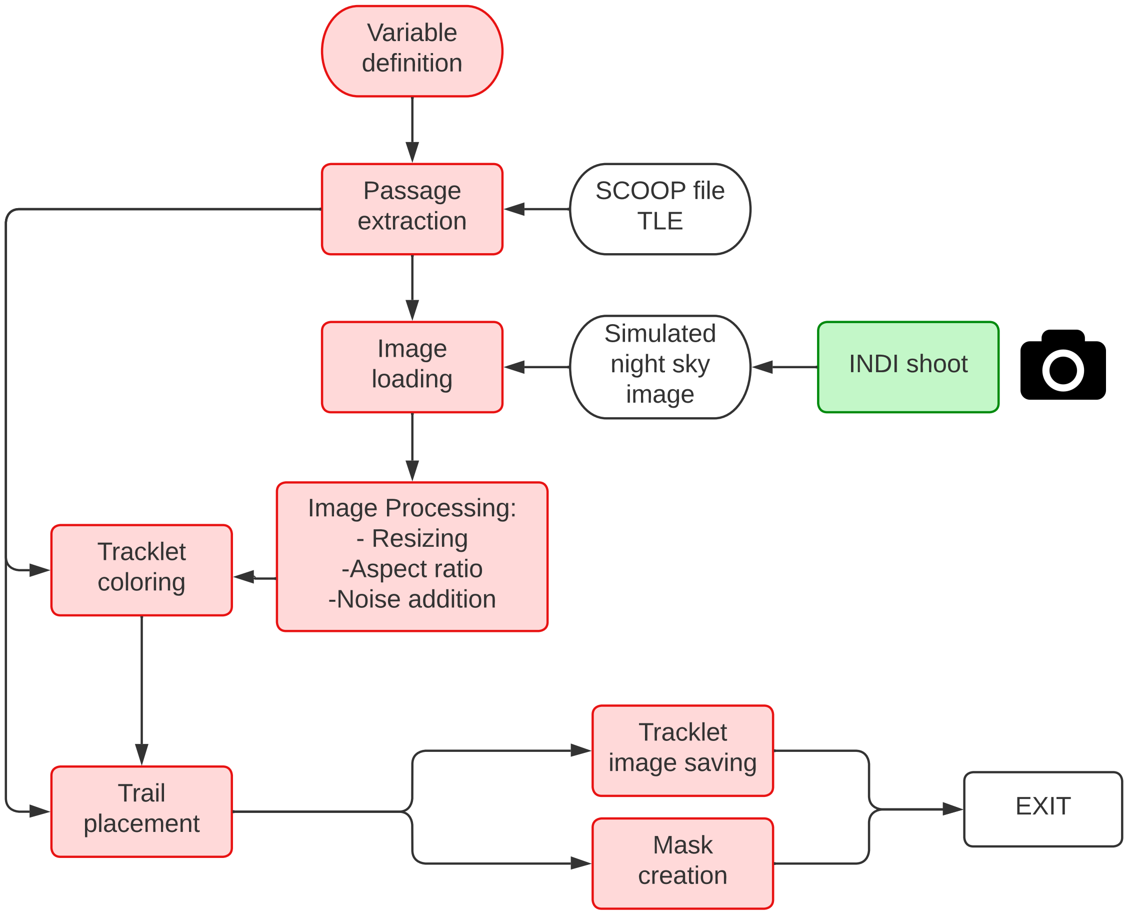


Figure 2.13: TIG block structure.

Folder organization

FITS images are shot by INDI server and placed inside the folder “Input_images_fits”, located in the upper “CONTROL” folder. After the conversion to PNG, the image is moved to the folder “Input_images_png”. At this point TIG prints the tracklet on the picture, successively moved to two different directories: initially it was kept inside the “CONTROL” folder, heritage of the previous TIG code, for code debugging and for an easier feedback on the proper functioning of the code, then it was placed instead inside the “LOT2” folder, where detection has been applied. Whenever the code is connected to a real telescope, the images are moved directly from the “Input_images_png” to the “LOT2/Input_Images”, skipping effectively the TIG routine.

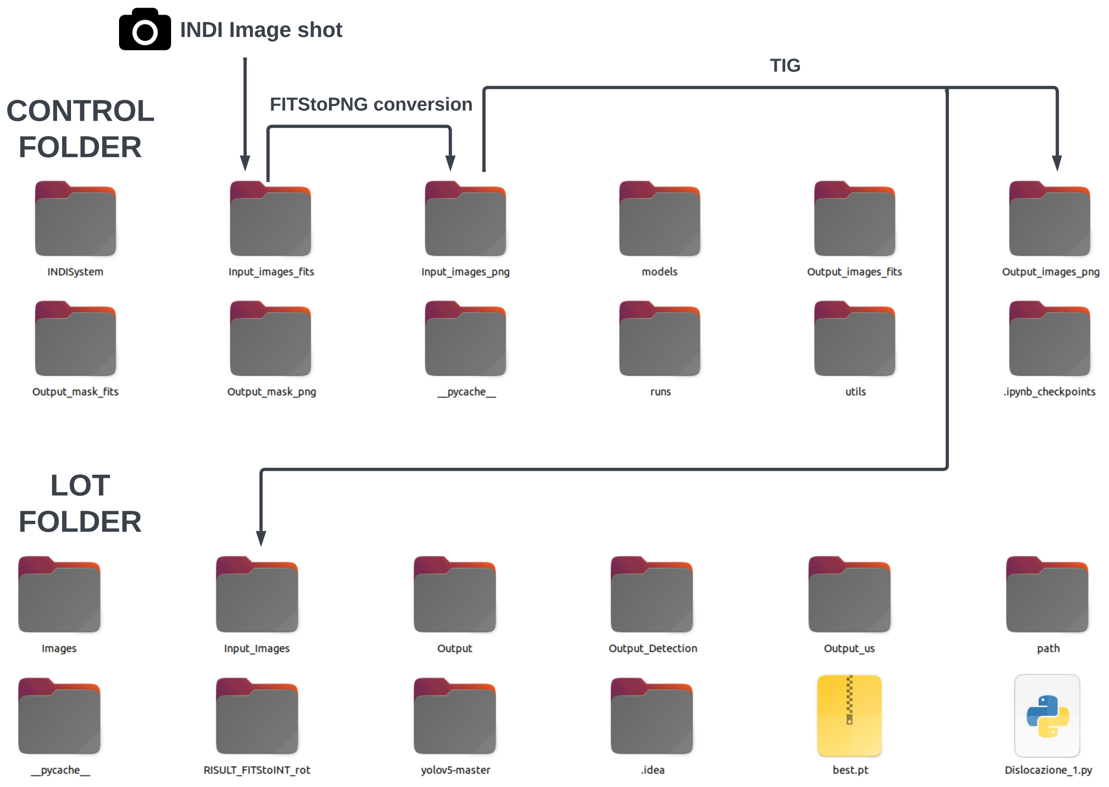


Figure 2.14: Image path during TIG processing.

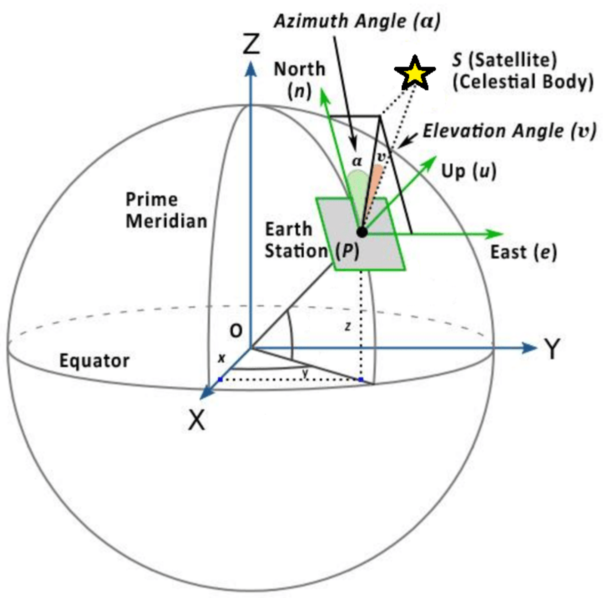
TIG coordinates

Differently from INDI server, TIG receives as input the coordinates in Azimuth-Elevation. It is a local reference frame, based on the observer location on the Earth, and it is described by:

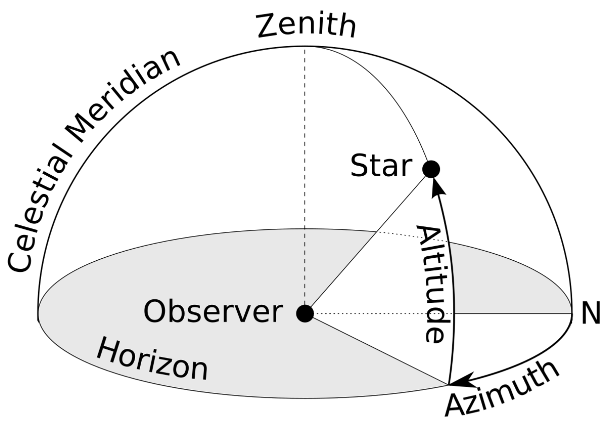
Azimuth : goes from 0° to 360° from the local North in the direction of the local East;

Elevation : goes from 0° to 90°. the 0 value is set at the horizon level, thus no object can be observed for negative angles.

Fig. 2.15 represents graphically the Azimuth-Elevation reference frame[3]. The system is centered at the observer, supposed in the center of the figure shown on the right, while on the left Azimuth-Elevation coordinates are in a Earth-fixed frame.



(a) From Earth [24].



(b) From a local observer [25].

Figure 2.15: Azimuth-Elevation.

2.7. YOLO v5

YOLO v5 is a family of object detection architectures and models pretrained on the COCO dataset, and it is an open-source AI framework [26]. As previously mentioned, YOLO stands for You Only Look Once because the detection occurs just after only one forward propagation through a Convolutional Neural Network (CNN). CNN are a multi-layer artificial neural networks for visual imagery; the convolution filters are devised to

[3]Images have been modified with respect to the original ones.

provide output maps given the input features. CNNs are specifically designed to process pixel data and are used in image recognition and processing [27]. YOLO model perfectly suits PoliMi telescope project because it learns generalized (and synthetic) representation of object, so that when it is trained on real images the algorithm outperforms other high level detection methods[10]. Despite being reliable, it requires few computational resources, a prerequisite for real-time detection on images. At the end of the processing phase each detected object is bordered within a bounding box and allocated to a class, which represents the kind of object recognized. An example is given in Fig. 2.17.

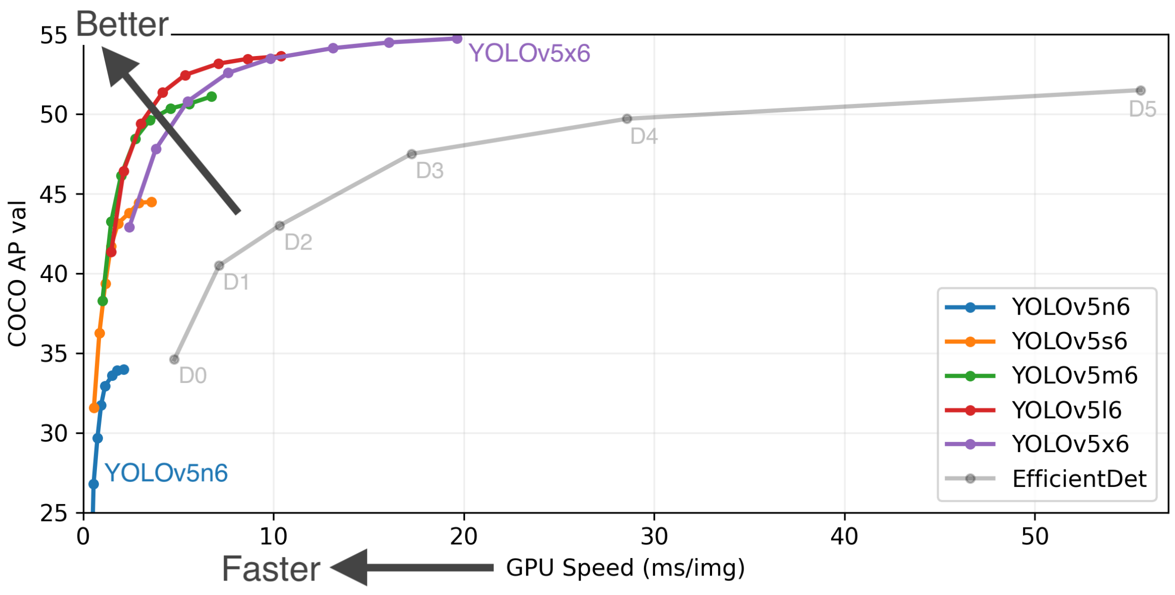
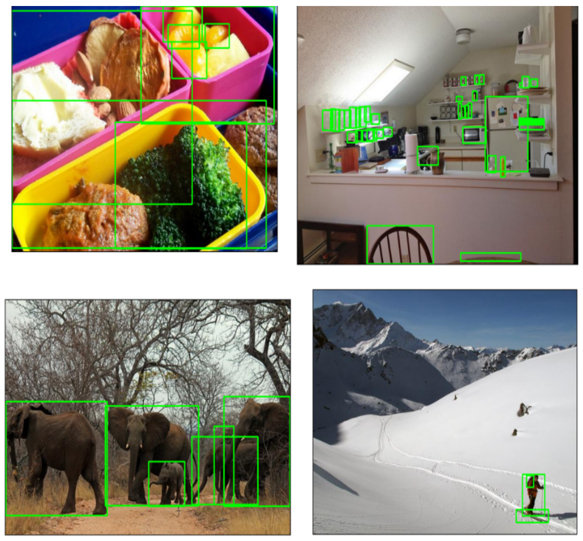
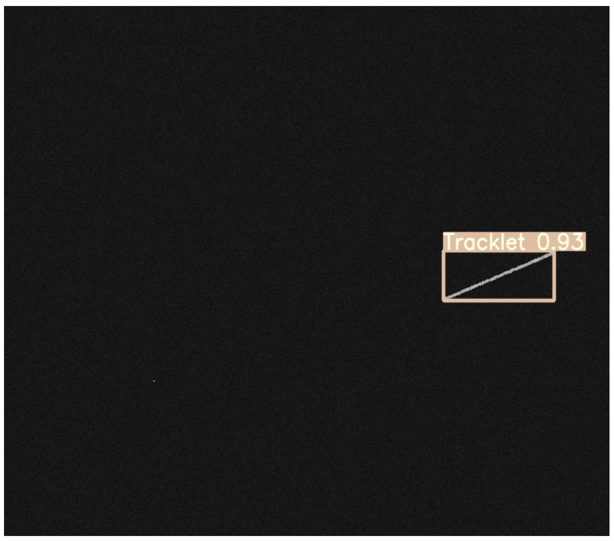


Figure 2.16: YOLO v5 performance increasing at any released version [26].

At PoliMi, YOLO Neural Network has been trained in order to detect tracklets on sky images. The result was sufficiently fast and accurate to enable unknown object detection and tracking, as it will be explained in Chapter 3. An example of YOLO detection on this kind of objects is given in Fig. 2.17b.



(a) Example of YOLO detection on images [28].



(b) Example of YOLO detection on sky images.

Figure 2.17: YOLO detection using bounding boxes.

2.7.1. Architecture

YOLO v5 is a single-stage detector based on dividing an image into a grid system, where each grid detects objects within itself. The network is structured in three main passages:

1. Backbone for pre-training;
2. Neck: added to recent versions, it consists in some layers to collect feature maps, and it is between the backbone and the Head.
3. Head for Dense prediction using bounding box and classes.

A representation of YOLO architecture is described in Fig. 2.18.

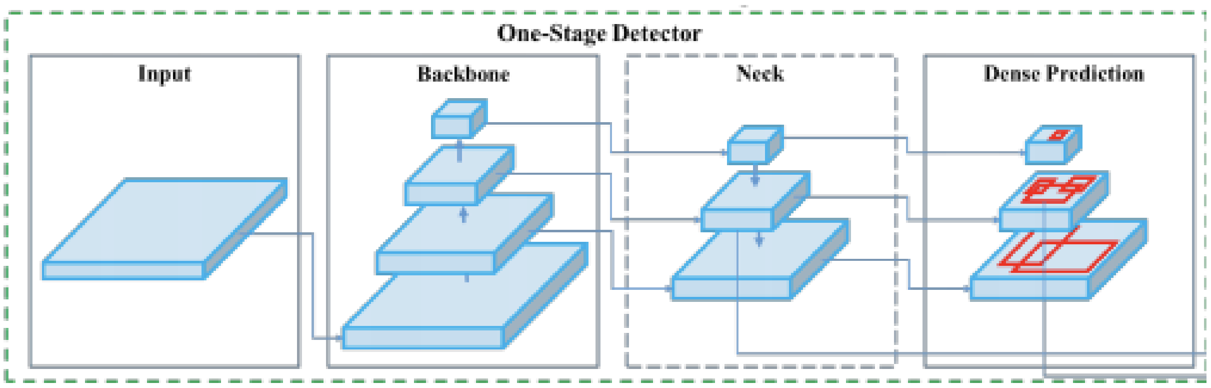


Figure 2.18: YOLO architecture [26].

YOLO is able to communicate with Python codes using a library called "PyTorch", which is responsible for the uploading, downloading and saving the trained model.

2.7.2. YOLO v5 training

The training dataset for astronomical detection included both real and synthetic images, already associated with the bounding box highlighting the desired solution. It is not straightforward to obtain real images of debris tracklets, due to permissions that must be required to the institutions dealing with space debris observation from the ground. YOLO has been previously trained and validated using synthetic images as part of the dataset. To do so, TIG generated a set of night sky images with a random tracklet thickness and placement that were used to increment the number of available images for this phase.

2.8. Fundamentals workflow

Before delving into STRATEGIC, all of the instruments and tools described in Chapter 2 have been analysed and reorganised as remarked in each section in order to connect each one to the others as in Fig. 2.19. Later on in Chapter 3 the folder organization will be explained. The color legend adopted is coherent with the one used for the main workflow:

Green blocks represent control;

Red blocks represent image processing and generation;

Orange blocks include detection;

Blue arrow indicates the tracking process.

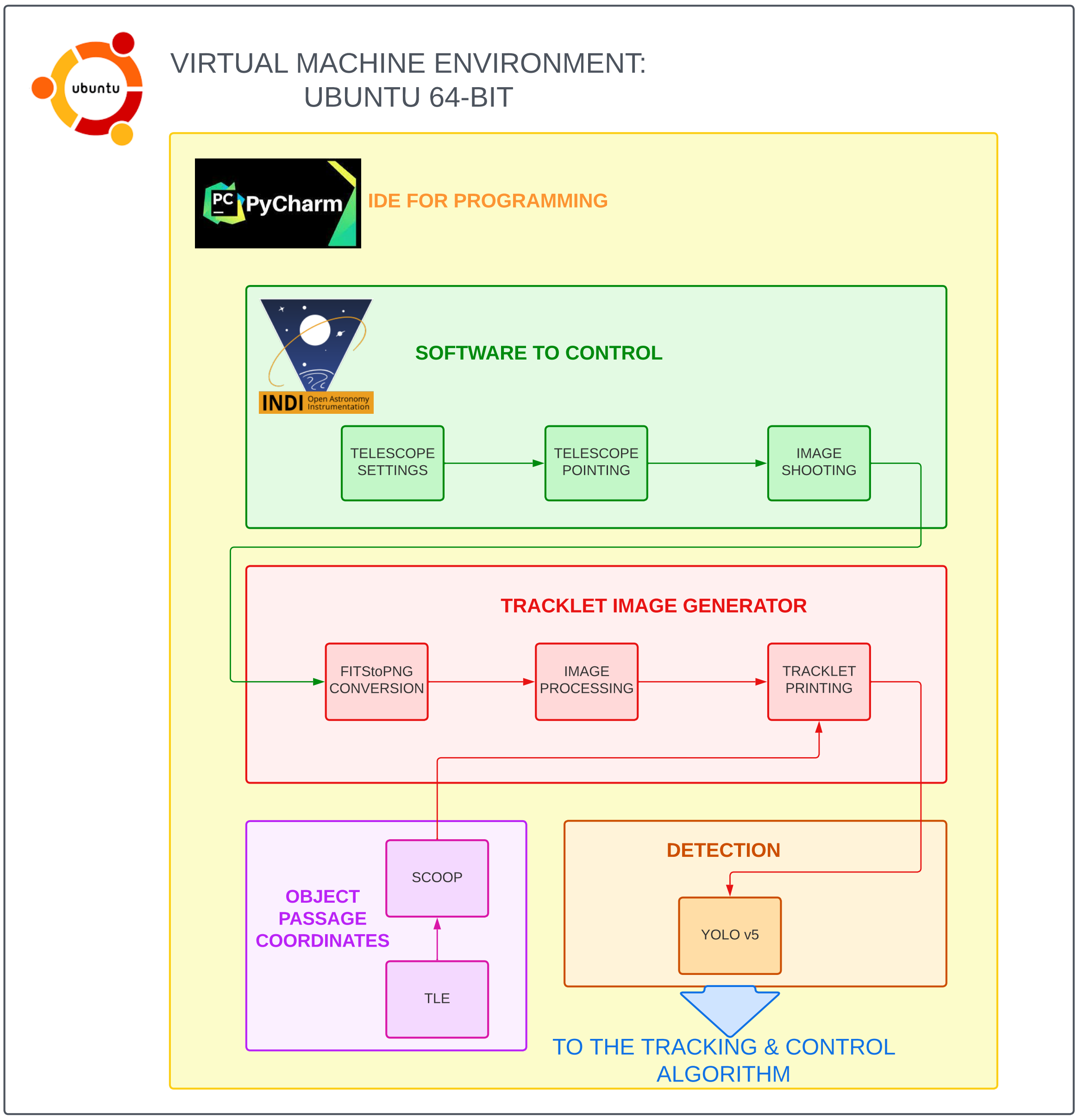


Figure 2.19: Preliminary organization workflow.

3 | STRATEGIC: Tracking and Control Software

Before this thesis work, a Neural Network has been trained to execute a detection on an astronomical image, find the presence of an object in it and recognise if the same object is present in the following image. The detection was not related to absolute ground coordinates, thus there was no way of tracking an unknown object. Nonetheless, the tools developed played a crucial role for STRATEGIC, the tracking & control algorithm that will be explained later on in this chapter. The first two sections are dedicated to LOT and RID, the image detectors which identify the tracklet cartesian position. Then, SPECTRA read LOT and RID files to deduce and collect data regarding the object location in space and time during its passages. Finally TEMPO analyzes the debris position and decides if the frame has to move to keep on following the trail.

3.1. LOT

This algorithm, namely "Linear Orbit Tracker", was made in PoliMi by Jason Calvi[10]. It allows to detect if a satellite or a debris is passing in front of the CCD camera, analysing the photo taken and searching for a tracklet in it. If many pictures are shot and a tracklet is found in two or more consecutive images, LOT estimates the possible trajectory of the streak and understands if the object in the two images is the same.

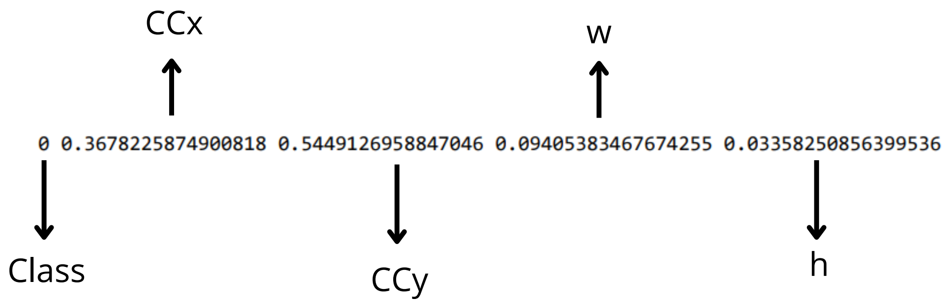
The code is stored with the name "LOT.py" inside the folder "LOT2" and it works as follows:

1. At the very beginning of the code, the model for the detection and its relative weights are uploaded using the **torch.hub** command from the PyTorch library.
    - the model uploaded is *yolov5-master*;
    - the weights are stored in the*best.pt* file optimized for synthetic images;
2. a "while True:" loop begins. It is a while loop with a True condition, thus the code

never stops running until it is manually shut down or it encounters an error.

3. The code searches inside the “input_images” folder if any image is present. It sorts the images by their name and the first one is selected.

4. The image passes in a function called “detect_torch”. This is where the proper detection happens: If a tracklet is found, a .txt file is generated. It has the same name of the image and it exists **only** if the model finds a track inside the picture. The name is in the form “1_Tracklet.txt”, and an example is shown in Fig. 3.1. The file is a single-line txt where the first number is the class of the object (0 for tracklets, 1 for clouds). Moving to the right, the tracklet’s Centroid position along x (CC_x) and y (CC_y) can be found and after them the Bounding Box width (w) and height (h).



```
0 0.3678225874900818 0.5449126958847046 0.09405383467674255 0.03358250856399536
```

Figure 3.1: Example of txt file generated from the Torch detection.

5. The image is moved from the “input_images” folder to avoid another detection on it.

6. The code divides into three parts:

   - if no files are present, no tracklets have been detected inside the image, and it is moved to a “single_image_folder” inside the folder “Output”.
   - if one file is detected, the tracklet’s slope is computed using its Centroid coordinates and Bounding Box dimension. Since only a single file is available, it is still impossible to recognise the direction of the tracklet that could follow four possible trajectories along two different lines, as indicated in Fig. 3.2. The function recalled is “prima_det” (namely, “first_determination”), and it generates a two-line txt file containing data regarding the two possible slopes (M) and intercepts (Q) the tracklet can assume with their confidence level (σ).
   - If more than one file are present, the code selects the function “seconda_det”

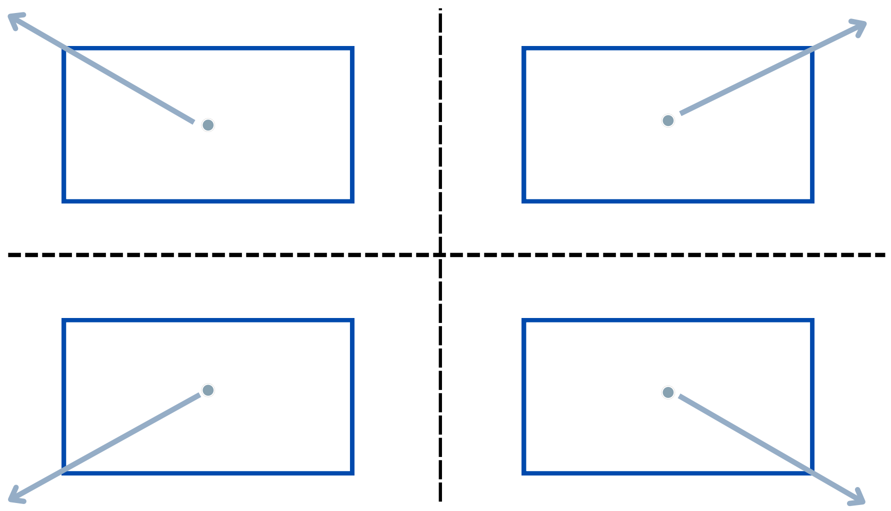
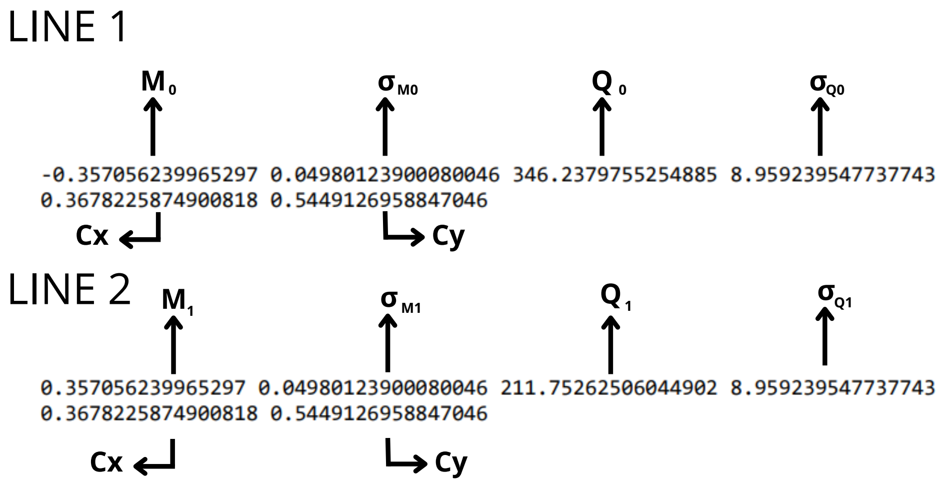


Figure 3.2: Graphical representation of the four possible trajectories.

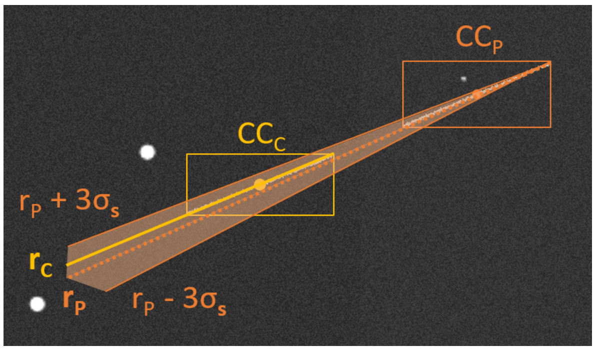

LINE 1
M_0
σ_{M0}
Q_0
σ_{Q0}
Cx
Cy
LINE 2
M_1
σ_{M1}
Q_1
σ_{Q1}
Cx
Cy

```
-0.357056239965297 0.04980123900080046 346.2379755254885 8.959239547737743
0.3678225874900818 0.5449126958847046
```

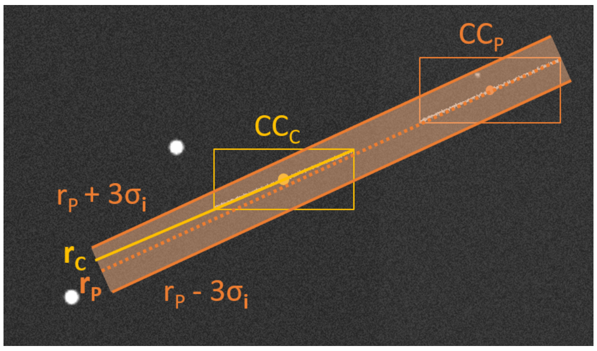
```
0.357056239965297 0.04980123900080046 211.75262506044902 8.959239547737743
0.3678225874900818 0.5449126958847046
```

Figure 3.3: MQ_Tracklet.txt example.

("second_determination"). It generates the same file as "prima_det" and then it compares the txt generated with the one related to the previous track. If the new slope matches one of the antecedent, the object is the same and the trajectory is fully determined. The pairing is determined if the slope of the new tracklet lies within the boundary set for the previous tracklet, given by $M_0 \pm 3\sigma_0$ or $M_1 \pm 3\sigma_1$. A visual representation of the empirical rule for consecutive tracklets determination is given by Fig. 3.4: "P" stands for "previous observation", while "C" is "current". "CC_x" elements are related to the centroids, "r_x" to the straight lines, and "σ_x" to standard deviation.



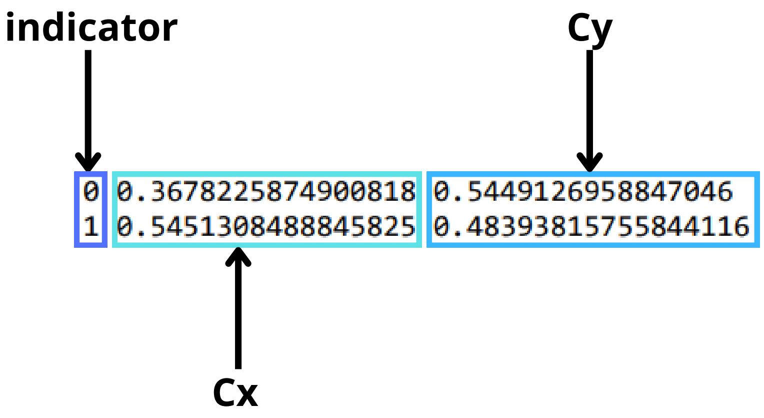
(a) Slope empirical rule.



(b) Intercept empirical rule.

Figure 3.4: Bounding boxes empirical rule. Courtesy of [29].

In this case another txt file containing the two Centroid positions is generated and saved in the “Output” folder. The file is called as “1_Tracklet_CC _2_Tracklet.txt”. An example is shown in Fig. 3.5. It is a two-line file with the Centroid coordinates along x and y related to two consecutive tracklets detected inside the last two images, in chronological order.


indicator
Cy
Cx

```
0 0.3678225874900818 0.5449126958847046
1 0.5451308488845825 0.48393815755844116
```

Figure 3.5: Centroid position file example. Saved as “1_Tracklet_CC_2_Tracklet.txt”.

7. Anytime a new object is detected, all the files related to the previous observation are moved from the folder “Output_detection” to a dedicated folder inside “Output”.

8. The code waits 0.01s before restarting the loop. This is a mandatory passage to avoid time superposition inside the “While True” loop commands, which are time-related (i.e. the choice of the most recent file inside a folder).

LOT files distribution is slightly intricate to describe, for this reason a figure is used to summarize all the passages described above. Figure 3.6 highlights the path an image follows inside the “LOT” function, denoted by continuous lines. Dashed lines instead show the text files itinerary.

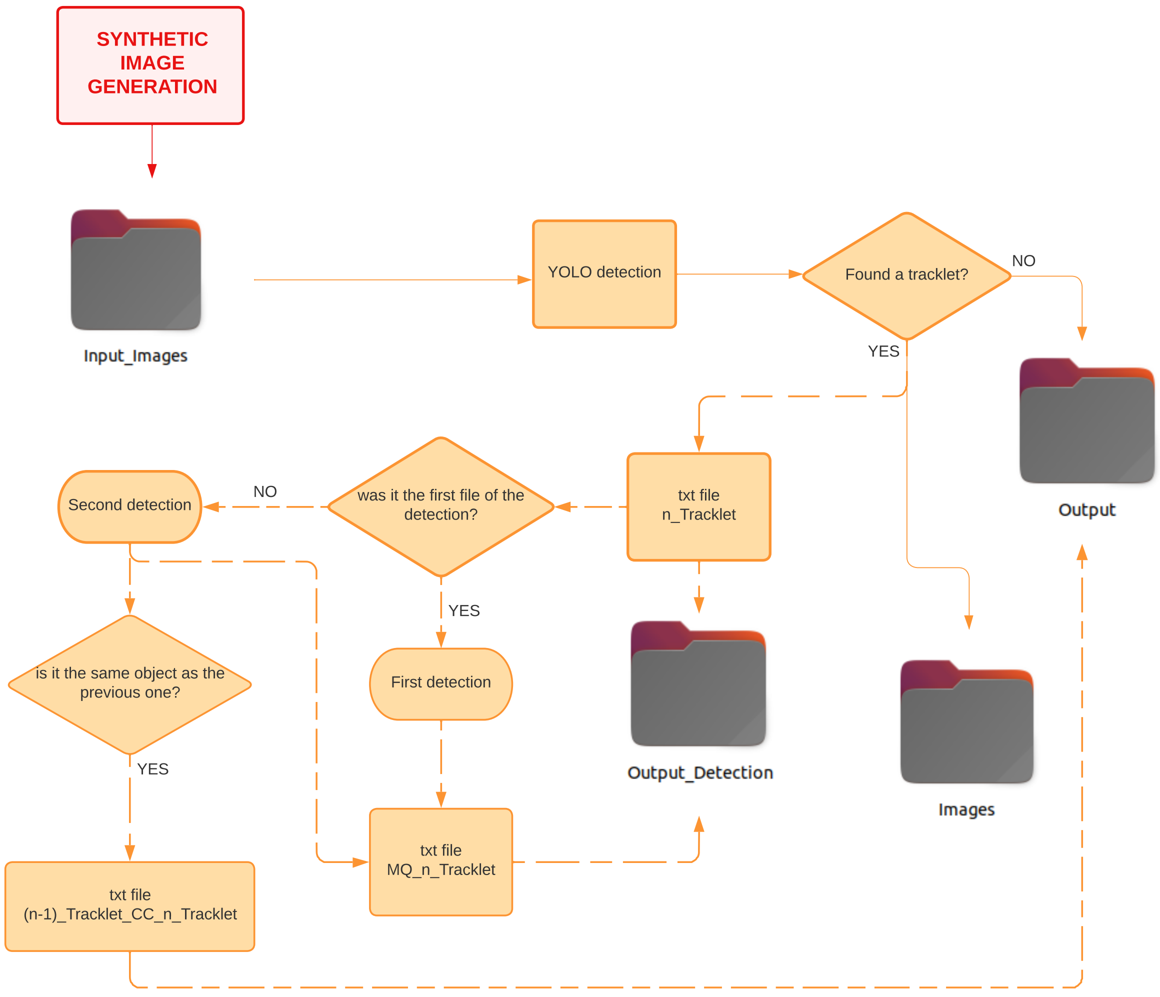


Figure 3.6: Folder arrangement and LOT workflow.

Modification

First of all, the images arriving to the “Input_Images” folder were shot by the INDI server and passed through the TIG, where the tracklet has been draw (see Fig. 2.14). A prototype of the code was made trying to run two different Python code in parallel, but the detection failed, since the main code did not wait for LOT to finish before moving to the tracking phase, where txt data generated by LOT were required. For this reason the “while True” loop became a function. The YOLO v5 model and weights are uploaded at the beginning of the whole process, so that they do not make the tracking algorithm lose time at any iteration. They are directly given as input to the new function LOT. In this way LOT is called only once per image and the other part of the code runs only when LOT ends the detection. All the process described above in Section 3.1 is taken only once per image, starting from point 3. Point 1 is moved to be part of the main code of STRATEGIC.

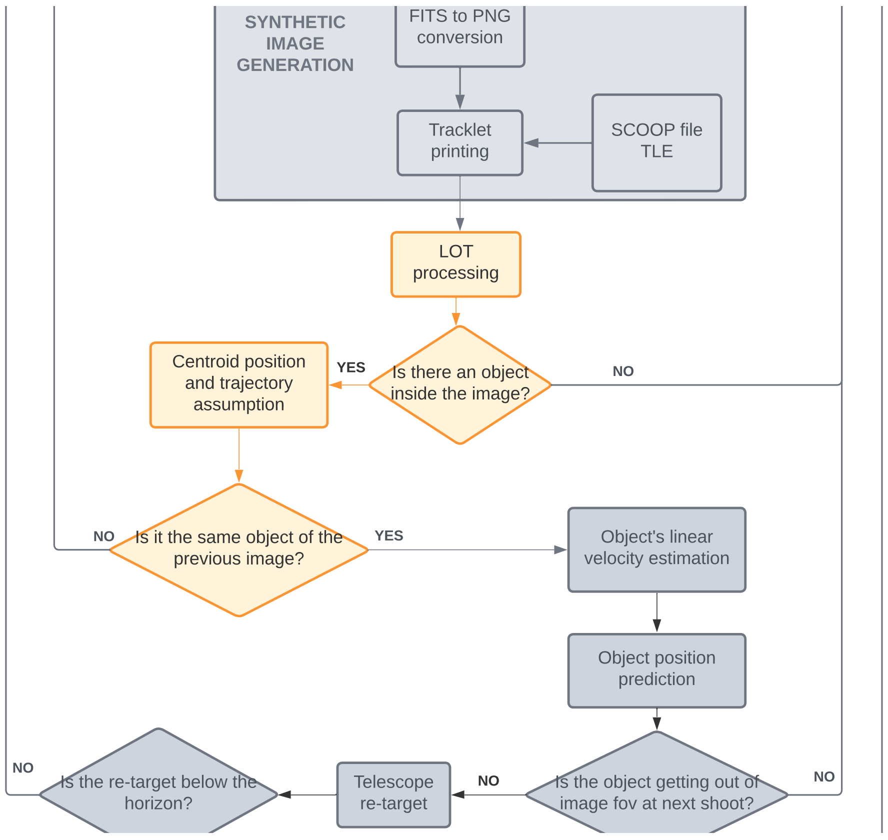

SYNTHETIC IMAGE GENERATION
FITS to PNG conversion
Tracklet printing
SCOOP file TLE
LOT processing
Centroid position and trajectory assumption
YES
Is there an object inside the image?
NO
NO
Is it the same object of the previous image?
YES
Object's linear velocity estimation
Object position prediction
NO
Is the re-target below the horizon?
Telescope re-target
NO
Is the object getting out of image fov at next shoot?
NO

Figure 3.7: Workflow section where LOT has been employed.

3.1.1. Output

All the outputs given by LOT have been already discussed and shown in Fig. 3.2, Fig. 3.3 and Fig. 3.5. The former is used in another function called “SPECTRA” that keeps track of the satellite passage and will be discussed later on. The second one is only used for determining if two different tracks belong to the same object as already defined in this section. The latter is the file txt containing the position of the last two centroids detected belonging to the same object. This kind of file is the first input of TEMPO, the movement prediction algorithm which is the main core of this thesis work.

3.1.2. Limit

It is important to remind that LOT performs its detection relying only on the last two images analyzed and using just a set of “relative” coordinates (see Fig. 2.6). Indeed, the position of the tracklet is related to the image cartesian system and **not** to absolute coordinates. This is a critical point for tracking objects, since LOT is not able to deduce whenever the telescope has moved or not. For this reason during the control software development, LOT’s tracking ability revealed two main critical cases where its interpretation was wrong:

Telescope movement: whenever a new trace is detected, usually LOT can recognize it, moving all the txt files used for the previous track from a directory to another, avoiding the computation of the file “X_Tracklet_CC_Y_Tracklet.txt”. This should happen even when the telescope is moved, since the first track that has to be followed is not linked to the previous one by the same celestial coordinates. However, due to the fact that the two trails belong to the same space target, they “fool” the LOT’s confidence level σ, that associates the last image, shot with a difference pointing of the telescope, to the previous one. The two tracklets are recognized as the same object - since actually they *are* - but in the same Field of View - which they *are not*. It returns that the two images contain the same object moving in the opposite direction (visual representation of this event is shown in the figures below). In this case, LOT still creates a file of the kind “X_Tracklet_CC_Y_Tracklet.txt”, but due to the different absolute coordinates of the telescope, it is read in the wrong way and must be ignored by the function using it later on in the main code. An example is given below. A set of images shot during tracking needs to be analized as in Fig. 3.8. They are numbered in chronological order.

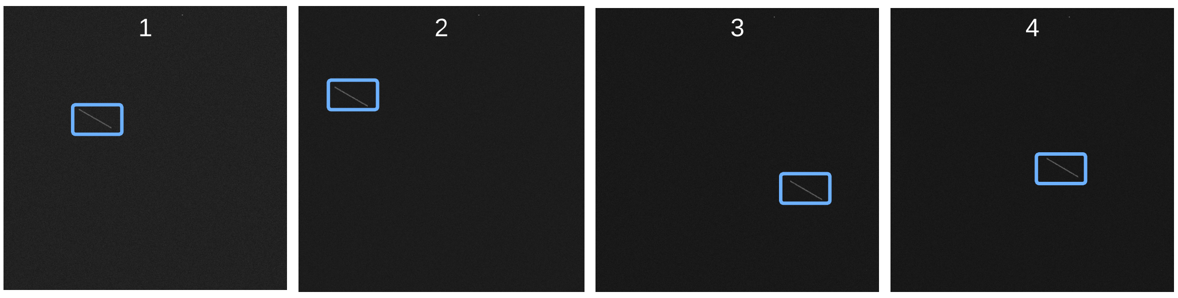


Figure 3.8: Image set. Tracklets are highlighted for a better visualization.

In the following images telescope pointing is indicated with a red dot to highlight its movement. The correct path the tracklet follows is shown in Fig. 3.9:

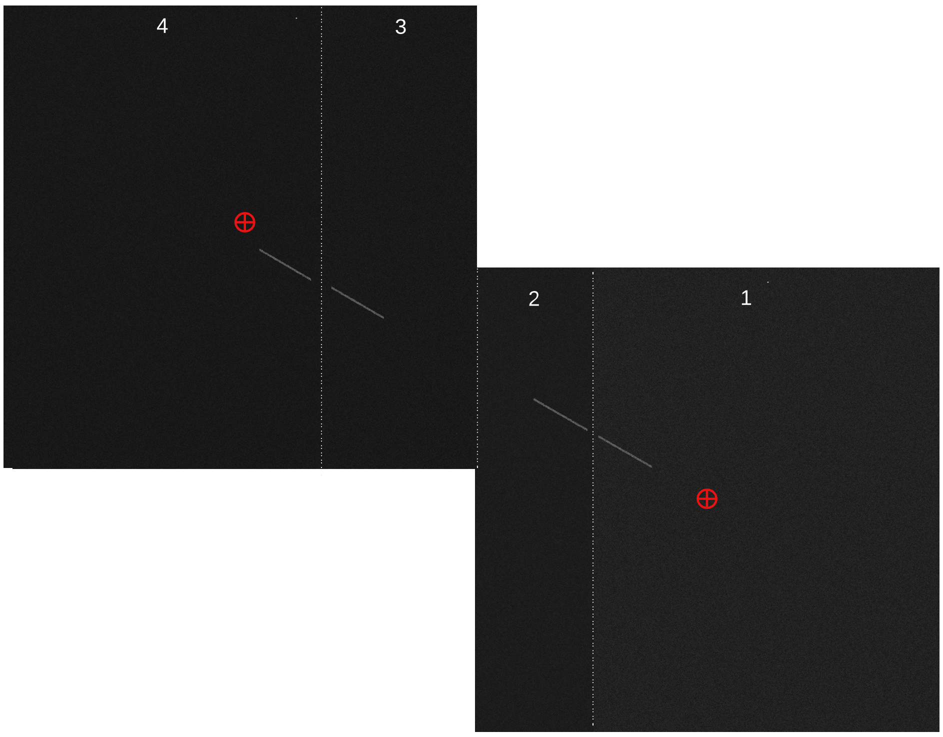


Figure 3.9: Real tracklet path in the sky.

When LOT analyzes image 2 and 3, its interpretation "believes" the tracklet is moving in the opposite direction, as shown in Fig. 3.10. This caused a critical issue for the greatest part of this thesis work, but TEMPO solved it in the way it will be discussed in Section 3.4.2.

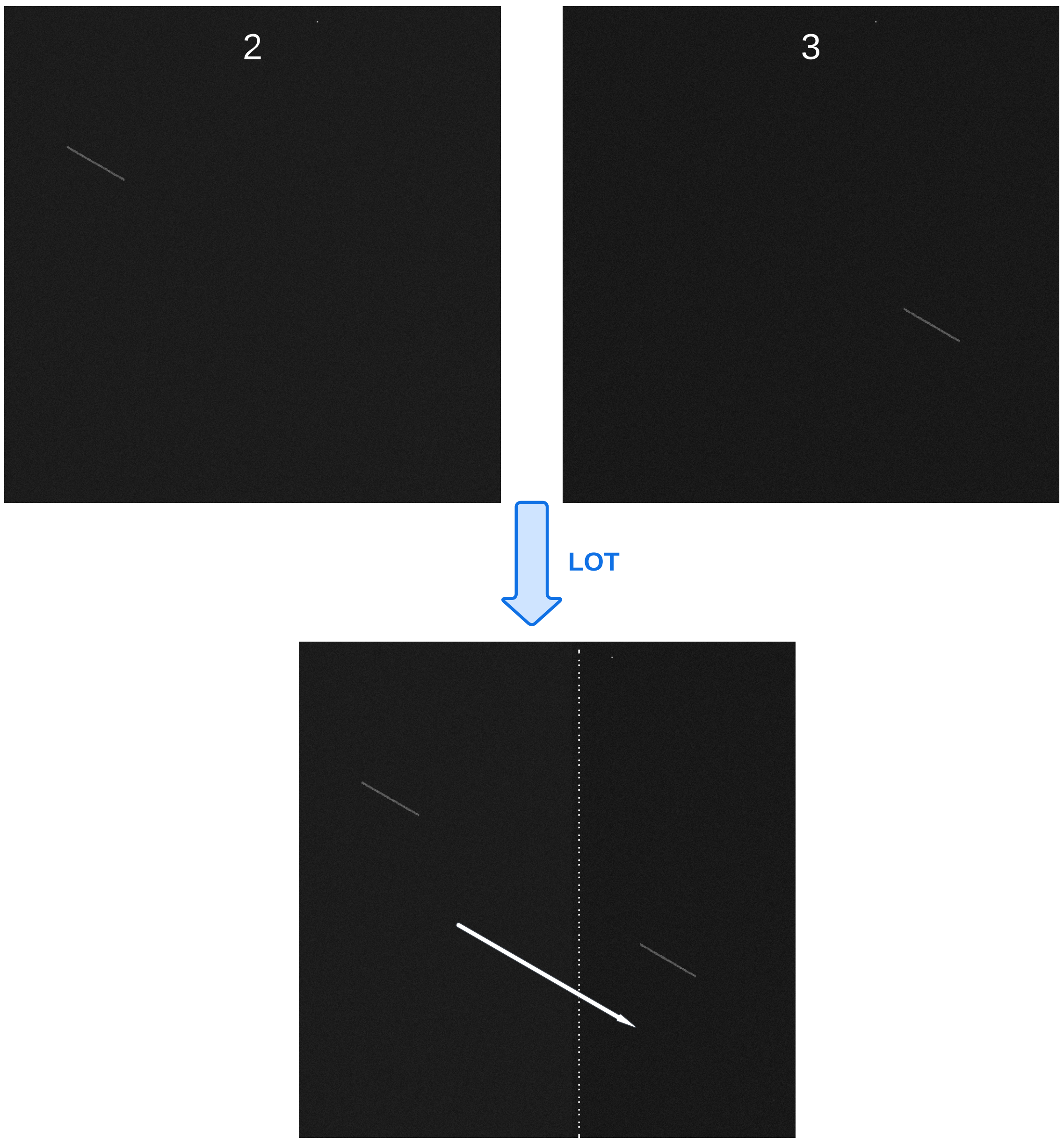


Figure 3.10: LOT interpretation of tracklet path.

Detection failure: LOT uses a Neural Network that rely on PyPlot for image detection. It is faster than SciPy, the library used for RID, but it tends to fail randomly at least once for every passage observation and it is not able to recognize the presence of two or more tracklets inside an image. This causes a set of off-nominal cases that will be explained later on, but it also limits the usage of this kind of detection to single fast-object tracking.

3.2. RID

The Real Image Detector is, precisely, an image detector developed by Jason Calvi at PoliMi. It could be considered the "earlier version" of LOT: it does not correlate consecutive images, but it applies on them a slower, more powerful detection. LOT detection demonstrated to fail recognising tracks in synthetic images and it may fail repeatedly on real images, which are affected by noise and clouds disturbances. For this reason, the hypothesis of connecting RID whenever LOT fails has been carried out.

3.2.1. Structure

Before starting the detection, a Python object "opt" is created. It contains all the options that need to be set before entering the RID function called "detect()", where "opt" is given as the only input. Regarding the model loaded, some differences from LOT need to be highlighted:

- the model is loaded every time inside the function, using the command **torch.load**;
- the uploaded weights are "*Best.pt*";
- the Yolo version is "*YOLOv5s*".
- image processing is handled with a Python library called Skimage, instead of PyTorch.

The image is processed and analyzed by the model, which is able to retrieve more information than LOT. Indeed, RID is able to return as output:

1. a cropped image centered on the tracklet, used for tracklet extraction techniques, which is not discussed in this thesis. An example is shown on the left-side of Fig. 3.11;
2. a text file with the tracklet's centroid position identical to the one LOT retrieves when the detection is succesful (see Fig. 3.1);

3. the detected image with a bounding box drawn around the tracklet as on the right side of Fig. 3.11. The number close to the Tracklet note indicates the confidence level of RID for the object to be a real tracklet. 1 means complete certainty of the target to be a tracklet, 0 means no confidence at all.

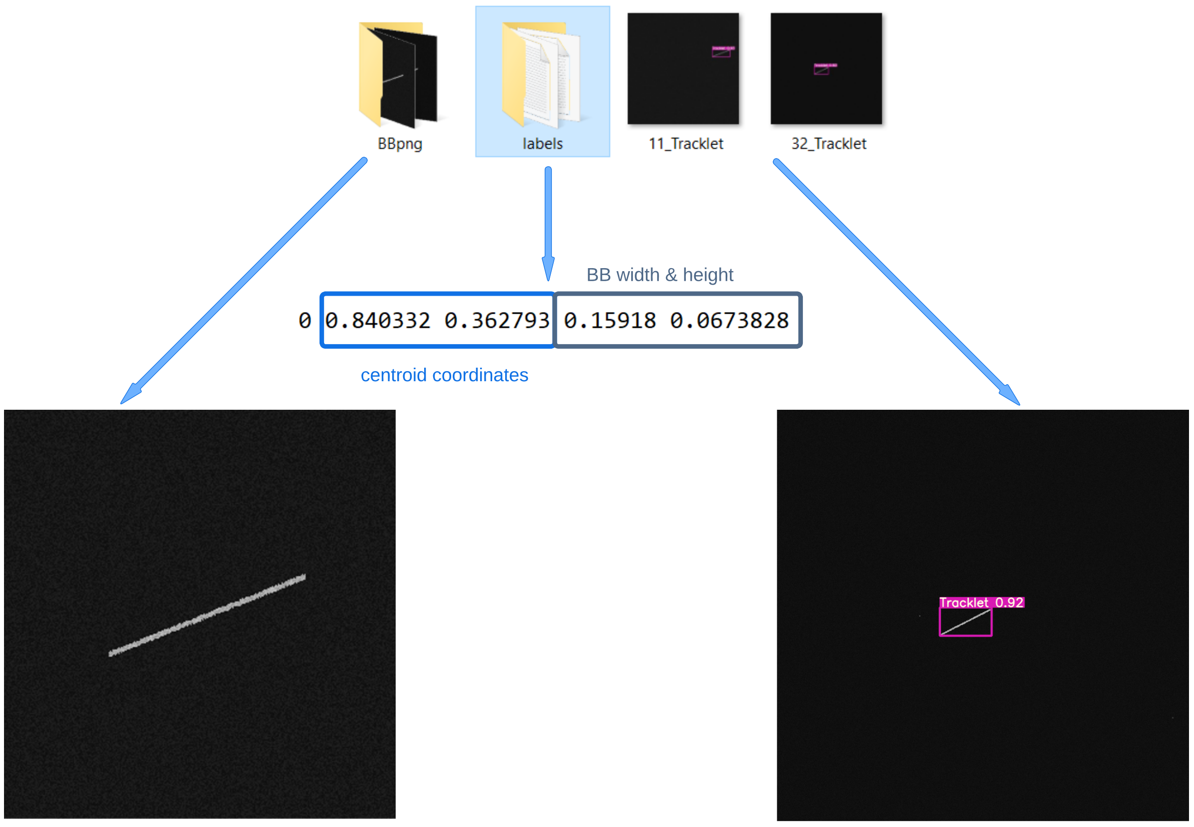
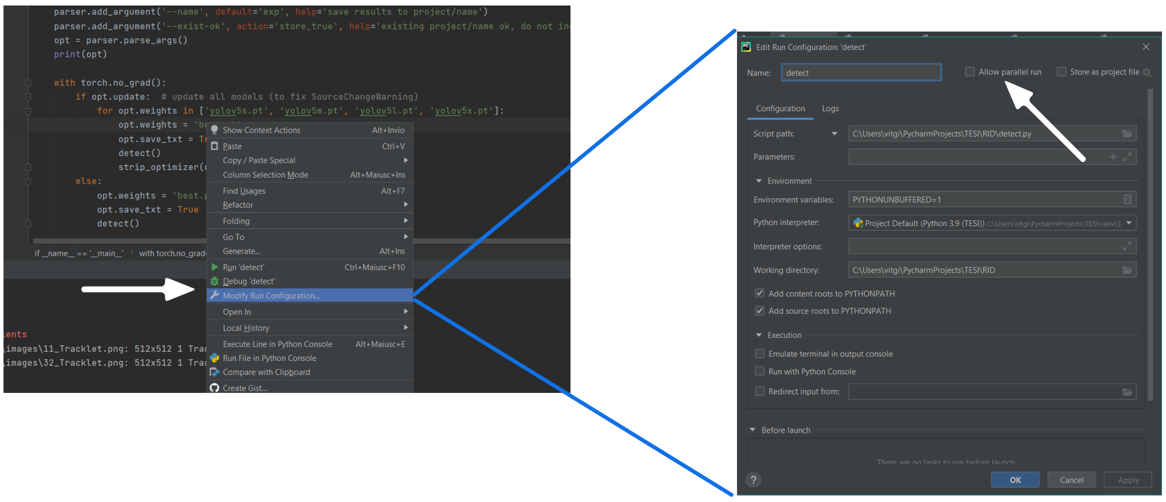


Figure 3.11: Graphical description of RID outputs.

3.2.2. RID employment

If LOT fails the detection on two consecutive images, the sky object cannot be tracked anymore. Since LOT is optimized for synthetic images, it could be possible that LOT failure frequency can increase in a real images detection and tracking case. For this reason, RID was taken into account. The idea is that if LOT fails the detection, the image is moved to RID folder and analyzed by the "detect()" function, which activates RID process. The implementation of this sequence proved to be not as straightforward as it was considered. Indeed, an issue has been encountered when connecting RID in series with LOT detection: when loaded, the model overwrites the weights of the two neural networks so that when RID is implied in the detection, its tensor dimensions do not match anymore the nominal ones it should operate with when LOT is not recalled in the same code. This way the operations carried out by the algorithm return an error. Some expedients have been tried, such as force the correct weights loading or rename all the variables involved to avoid them to be overwritten. In any case, anytime RID was recalled in the same script as LOT, the detection command line returned errors. Therefore, a RID fully dedicated code was implemented to run in parallel with LOT.

It is possible to run more than a single code simultaneously with Python. However, as for LOT's initial "while True" loop, this solution is not recommended for this kind of problem. In facts, LOT does not wait for RID to finish its detection before moving to the following passage, where RID output is necessary. In the specific case, parallel run is the only working solution to connect the two detectors. It is possible to allow this feature clicking with the right button of the mouse on the script, select "Modify run configuration" and "allow parallel run", as explained in Fig. 3.12.



```
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not in
    opt = parser.parse_args()
    print(opt)

    with torch.no_grad():
        if opt.update:  # update all models (to fix SourceChangeWarning)
            for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
                opt.weights = 'be
                opt.save_txt = Tr
                detect()
                strip_optimizer(
        else:
            opt.weights = 'best.
            opt.save_txt = True
            detect()
```

```
ents
images\11_Tracklet.png: 512x512 1 Trac
images\32_Tracklet.png: 512x512 1 Trac
```

Edit Run Configuration: 'detect'

Name: detect | ☐ Allow parallel run | ☐ Store as project file

Script path: C:\Users\vitgi\PycharmProjects\TESI\RID\detect.py

Parameters:

Environment variables: PYTHONUNBUFFERED=1

Python interpreter: Project Default (Python 3.9 (TESI)) C:\Users\vitgi\PycharmProjects\TESI\venv\S

Interpreter options:

Working directory: C:\Users\vitgi\PycharmProjects\TESI\RID

☑ Add content roots to PYTHONPATH

☑ Add source roots to PYTHONPATH

☐ Emulate terminal in output console

☐ Run with Python Console

☐ Redirect input from:

There are no tasks to run before launch

Figure 3.12: How to allow parallel run on scripts in PyCharm.

LOT and RID interact as follows:

- a variable is named after **LOT** "detect_torch". If the variable contains information, LOT proceeds as already described in Section 3.1. Instead, if the variable is "None" type, it means LOT failed the detection. Thus the image is moved to **RID**'s folder. The code pauses for 1s. The choice of 1s depends on the fact that RID's detection usually takes about 0.8s to run.
- in the meanwhile, a code called "Control_detect" is running. At the beginning of the code RID's options are loaded. Then a "While True" loop begins, where the code keeps waiting until an image arrives in RID's folder. When an image is found,

RID's detection starts and gives as output the txt file LOT failed to retrieve, placing it in LOT folder, exactly where it would have been placed from the very beginning if the detection was not missed.

- LOT ends its waiting time and move to the following section with (or without) the text file required.

This method does not ensure RID will always **always** end the detection before LOT moving to the next passages, since some images could take more than 1s to be processed. In any case, this solution would surely increase the quantity of detected images, reducing the possibility to lose the observed object while tracking it. Moreover, LOT waiting time could be enhanced, adapting it to RID requirements.

3.2.3. Considerations

Three versions of the code can be used:

1. a version including only LOT;
2. a version that involves RID **only** in case of LOT failure;
3. a version that relies only on RID.

Each one of these versions has different properties, that could be exploited for different purposes.

1. LOT takes about 0.4s to execute a detection (half the time required by RID), but fails frequently. This allows fast LEO objects tracking once the object is identified, but it could reveal low efficiency for real images (especially cloudy ones) and it is totally ineffective if more than a single track is present in the FoV.
2. This second version should be more effective in case of real images, because it mixes the two detection networks, using the slowest only in case of necessity. In any case, RID represents the bottleneck of time consumption of the whole software and the waiting time required by this detection limits the maximum speed of the traceable objects.
3. This version could be really useful at the beginning of the observation, when the object to be tracked is not defined yet and the telescope is randomly scanning the sky. This because RID can detect all the tracklets in a camera shot. A further implementation of the code could lead to an analysis on the trails detected that consists in checking their correlation with debris and satellite catalogues. In this way unidentified object could be chosen to be followed for retrieving new data.

LOT and RID Organization

LOT and RID are not directly communicating between each other. Their connection is actuated by moving the images and the text files from a folder to another. Images always arrive to LOT's folder "Input_Images". If LOT fails, they are transferred to RID's folder, inside "data/image". In there, RID's checks if an image is present every 0.01s and when it finds a file, the detection starts. The text file generated is placed at the same position in which LOT would have put it, inside the folder ""Output_Detection". Figure 3.13 shows how the two parallel codes are connected between each other. Black arrows follow the code flowing and text files generation, while the path of the images is denoted with an orange arrow.

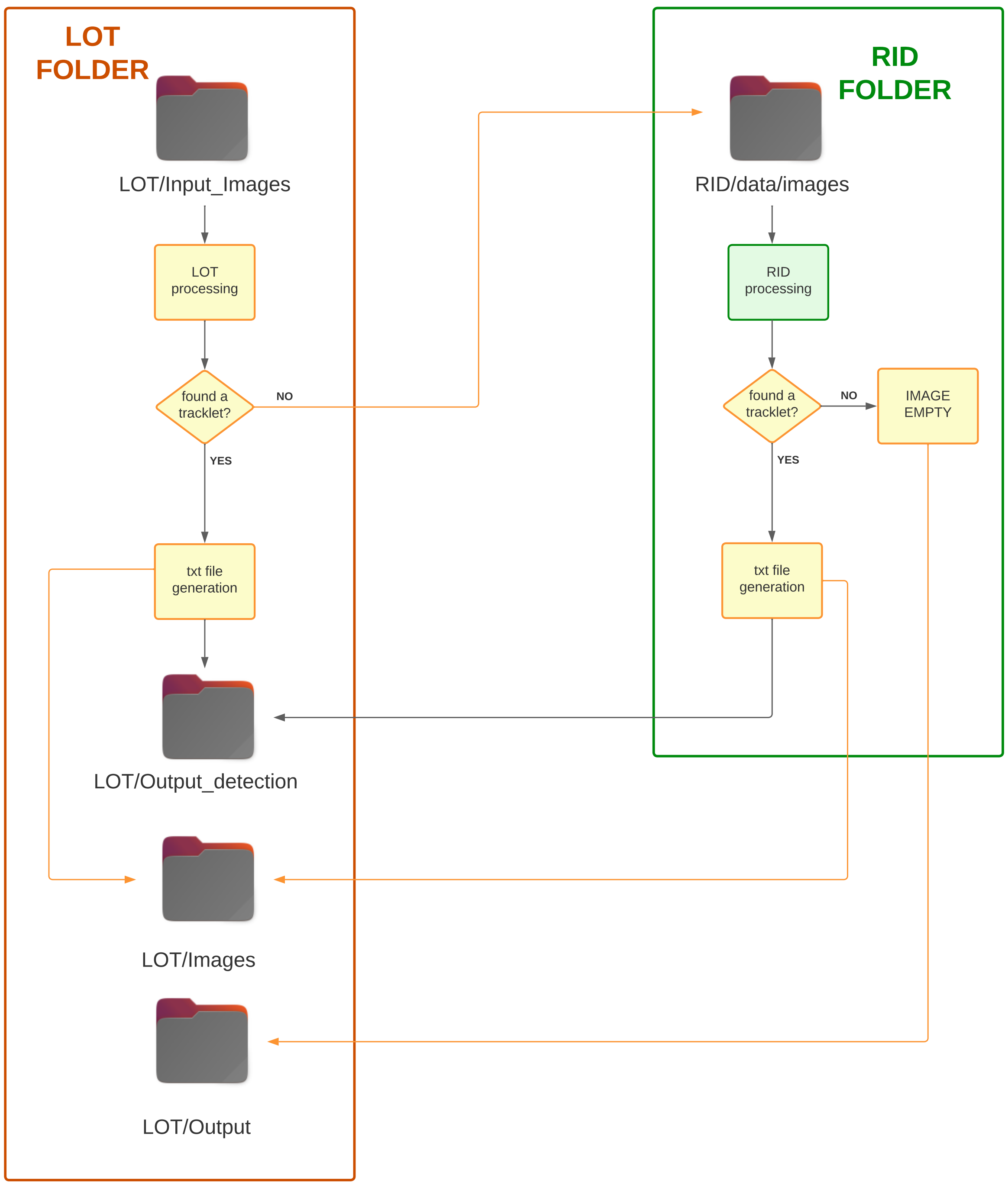


Figure 3.13: Connection between LOT and RID running in parallel.

3.3. SPECTRA

SPace dEbris TRAcker (SPECTRA) is an algorithm which is able to provide all the absolute coordinates an observed object assumed during its passage. The coordinates are retrieved in Azimuth-Elevation and Fig. 3.14 schematically explains how SPECTRA works.

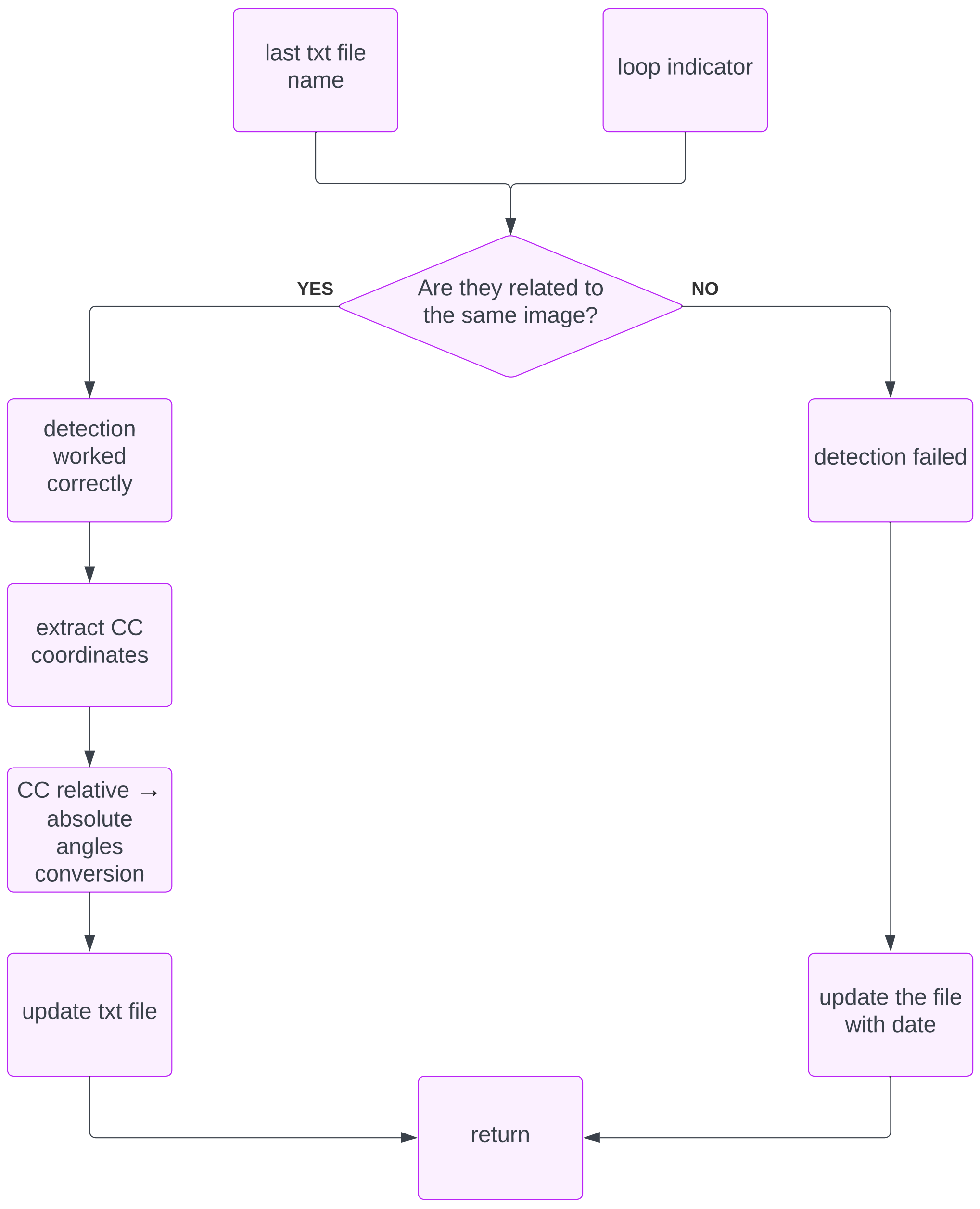

last txt file name
loop indicator
Are they related to the same image?
YES
NO
detection worked correctly
detection failed
extract CC coordinates
CC relative → absolute angles conversion
update txt file
update the file with date
return

Figure 3.14: SPECTRA flowchart.

3.3.1. Architecture

The algorithm takes as input all the information related to the shot:

- the time in J2000, computed as the mean value between the start of the shoot and the end. It is considered as the exact moment in which the object should pass by the center of the streak left on the image;
- the field of view of the telescope;
- the telescope position;
- the loop counter, used to name the images when the telescope camera shoots.

SPECTRA is recalled immediately after LOT and the first task it needs to accomplish is to check if the detection happened correctly. As anticipated in Section 3.1.1, it searches inside the folder "Output_Detection" the most recent text file containing the tracklet's centroid positions in non-dimensional coordinates. The files are called after the image they are related to (i.e. if the image is "3_Tracklet.png", its .txt file is called "3_Tracklet.txt") and the number they include inside their name is given by the counter changing at each loop iteration. The role of the counter and how it is set will be better explained in Section 3.5. At the moment in which the function is called, inside the "Output_Detection" folder two types of text file should be present:

- all the detected files "X_Tracklet.txt" with the centroid positions;
- all their "MQ" files "MQ_X_Tracklet.txt" containing information about the tracklet slope and intercept.

The function selects the most recent file that **does not** contain the letters "MQ" and splits its name to retrieve the number "X" at the beginning. If LOT detection happened correctly, the most recent file is related to the last image shot by the telescope, thus the file number and the counter should match. On the contrary, if the two numbers do not match, it means LOT failed and RID was too slow to perform its detection. This information is necessary to the second part of the whole algorithm, called TEMPO, thus it is stored in a boolean variable, "WRONG", which is saved as "True" if the detection **fails**, in the other cases it is "False". The same exact variable will be called "check" by TEMPO.

Once the detection is checked, the function reads the last file found and converts it into absolute celestial coordinates, following these passages:

1. The centroid's coordinates are read;

2. Since the coordinates are related to Python's images reference frame (see Section 2.1.2), the origin if the x and y axes is shifted to the centre of the image, where the telescope is pointing.

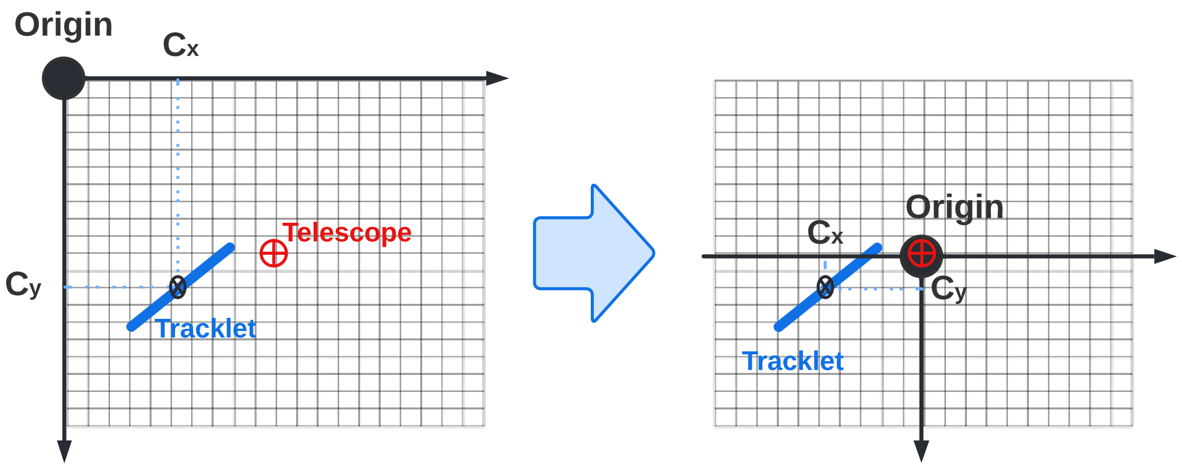
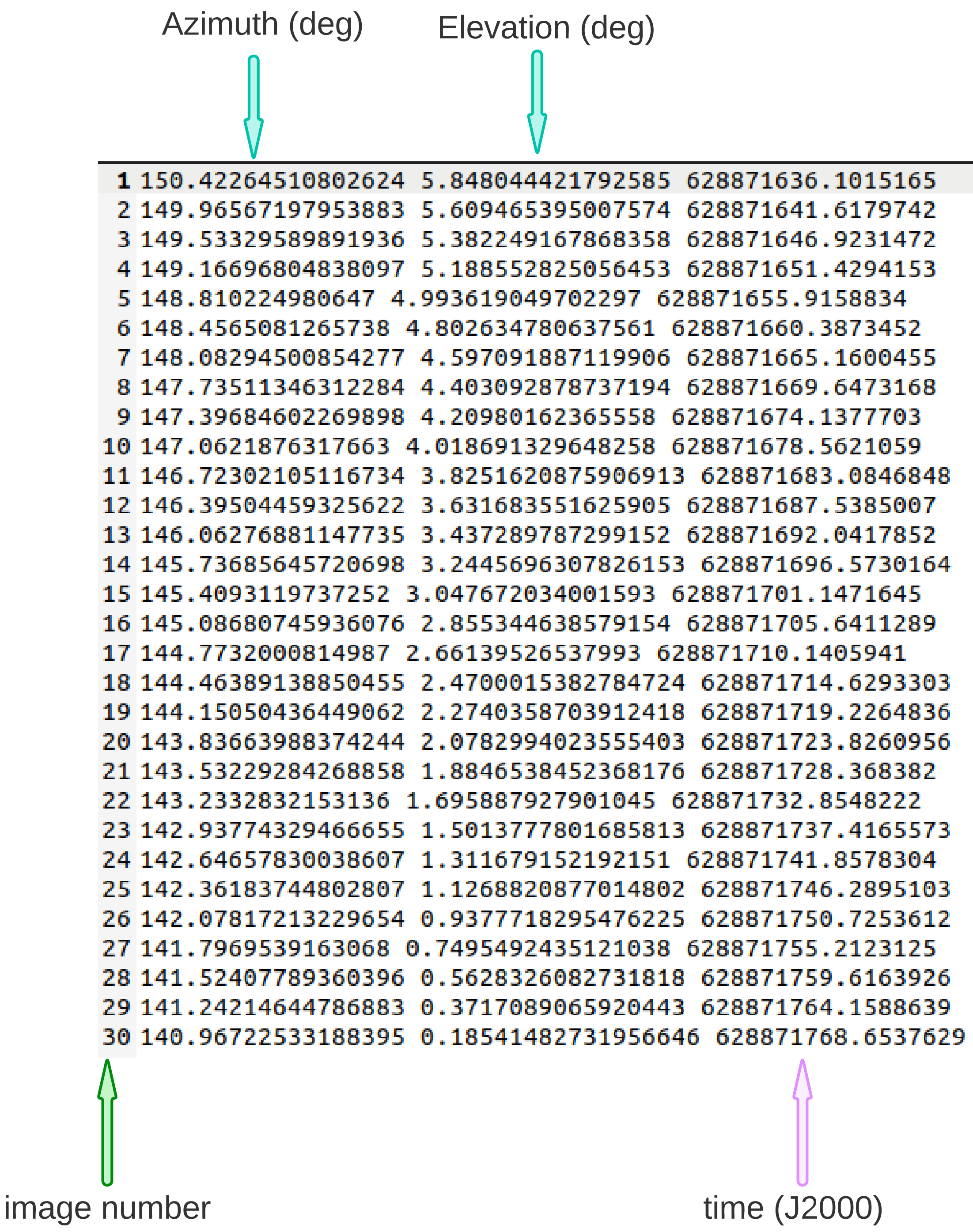


Figure 3.15: Reference frame shift.

3. The coordinates are multiplied by the FoV and converted from degrees to radians;
4. the function "Azel_Shift" takes as input the relative coordinates of the tracklet and the Azimuth-Elevation coordinates of the telescope to compute the Azimuth-Elevation coordinates of the centroid.
5. the coordinates are re-converted from radians to degree.

At this point, the function opens a text file inside the folder "Output_us" and at each iteration if the main code, it adds a row with Azimuth, Elevation and time in J2000 of the current image's centroid. At the end of the observation, a text file resembling Fig. 3.16 is available and can be used for further implementations.

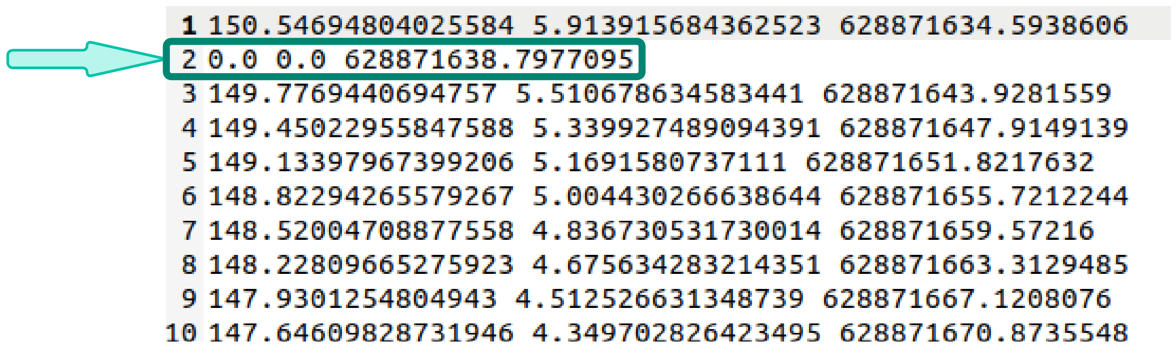


```
 1 150.42264510802624 5.848044421792585 628871636.1015165
 2 149.96567197953883 5.609465395007574 628871641.6179742
 3 149.53329589891936 5.382249167868358 628871646.9231472
 4 149.16696804838097 5.188552825056453 628871651.4294153
 5 148.810224980647 4.993619049702297 628871655.9158834
 6 148.4565081265738 4.802634780637561 628871660.3873452
 7 148.08294500854277 4.597091887119906 628871665.1600455
 8 147.73511346312284 4.403092878737194 628871669.6473168
 9 147.39684602269898 4.20980162365558 628871674.1377703
10 147.0621876317663 4.018691329648258 628871678.5621059
11 146.72302105116734 3.8251620875906913 628871683.0846848
12 146.39504459325622 3.631683551625905 628871687.5385007
13 146.06276881147735 3.437289787299152 628871692.0417852
14 145.73685645720698 3.2445696307826153 628871696.5730164
15 145.4093119737252 3.04767203400159 3 628871701.1471645
16 145.08680745936076 2.855344638579154 628871705.6411289
17 144.7732000814987 2.66139526537993 628871710.1405941
18 144.46389138850455 2.4700015382784724 628871714.6293303
19 144.15050436449062 2.2740358703912418 628871719.2264836
20 143.83663988374244 2.0782994023555403 628871723.8260956
21 143.53229284268858 1.8846538452368176 628871728.368382
22 143.2332832153136 1.695887927901045 628871732.8548222
23 142.93774329466655 1.5013777801685813 628871737.4165573
24 142.64657830038607 1.311679152192151 628871741.8578304
25 142.36183744802807 1.1268820877014802 628871746.2895103
26 142.07817213229654 0.9377718295476225 628871750.7253612
27 141.7969539163068 0.7495492435121038 628871755.2123125
28 141.52407789360396 0.5628326082731818 628871759.6163926
29 141.24214644786883 0.3717089065920443 628871764.1588639
30 140.96722533188395 0.18541482731956646 628871768.6537629
```

Figure 3.16: Example of file generated by SPECTRA.

3.3.2. Off-nominal case

If LOT fails the detection, SPECTRA ascertains that the latest file does not belong to the last image produced, as if it was the “wrong” file. The boolean variable “WRONG” is set on “True” and it is used later on as an alert for TEMPO. In this case, the algorithm prints two zeros instead of the Azimuth and Elevation values, together with the actual time in J2000. Therefore, it is immediate to recognize at which point the detection did not work as expected and correct the solution in post-processing, without losing the information regarding time.



```
150.54694804025584 5.913915684362523 628871634.5938606
0.0 0.0 628871638.7977095
149.7769440694757 5.510678634583441 628871643.9281559
149.45022955847588 5.339927489094391 628871647.9149139
149.13397967399206 5.1691580737111 628871651.8217632
148.82294265579267 5.004430266638644 628871655.7212244
148.52004708877558 4.836730531730014 628871659.57216
148.22809665275923 4.675634283214351 628871663.3129485
147.9301254804943 4.512526631348739 628871667.1208076
147.64609828731946 4.349702826423495 628871670.8735548
```

Figure 3.17: Failed detection in SPECTRA file.

3.3.3. Considerations

SPECTRA text file can be used for retrieving further information about the orbit of the satellites and debris observed and check their correlation with existing catalogues. Hence, partial orbit parameters could be given to more powerful instruments for a complete knowledge of the object. Furthermore, as it will be explained later on, for the time being missing the detection of two consecutive images means losing the track. This set of data could solve this criticality, since a linear estimation using the last two known positions at known time may be performed, even though the implementation may not be as simple as it seems: a prototype of this process has been coded during the developing of STRATEGIC but it never accomplished the objective of retrieving correct information to keep on tracking an object.

3.4. TEMPO

The algorithm is called TElescope Movement Prediction and Oversight and its main feature is to predict **when** and **where** the telescope needs to move to accomplish the objective of keeping an orbiting object inside the field of view, as schematically reported in Fig. 3.18. TEMPO is connected in the main code of STRATEGIC after SPECTRA since it is not able to recognise by itself whenever LOT fails the detection. This was the main reason of a set of criticality that caused the complete failure of the tracking. These cases are now solved and will be discussed separately in Section 3.4.2.

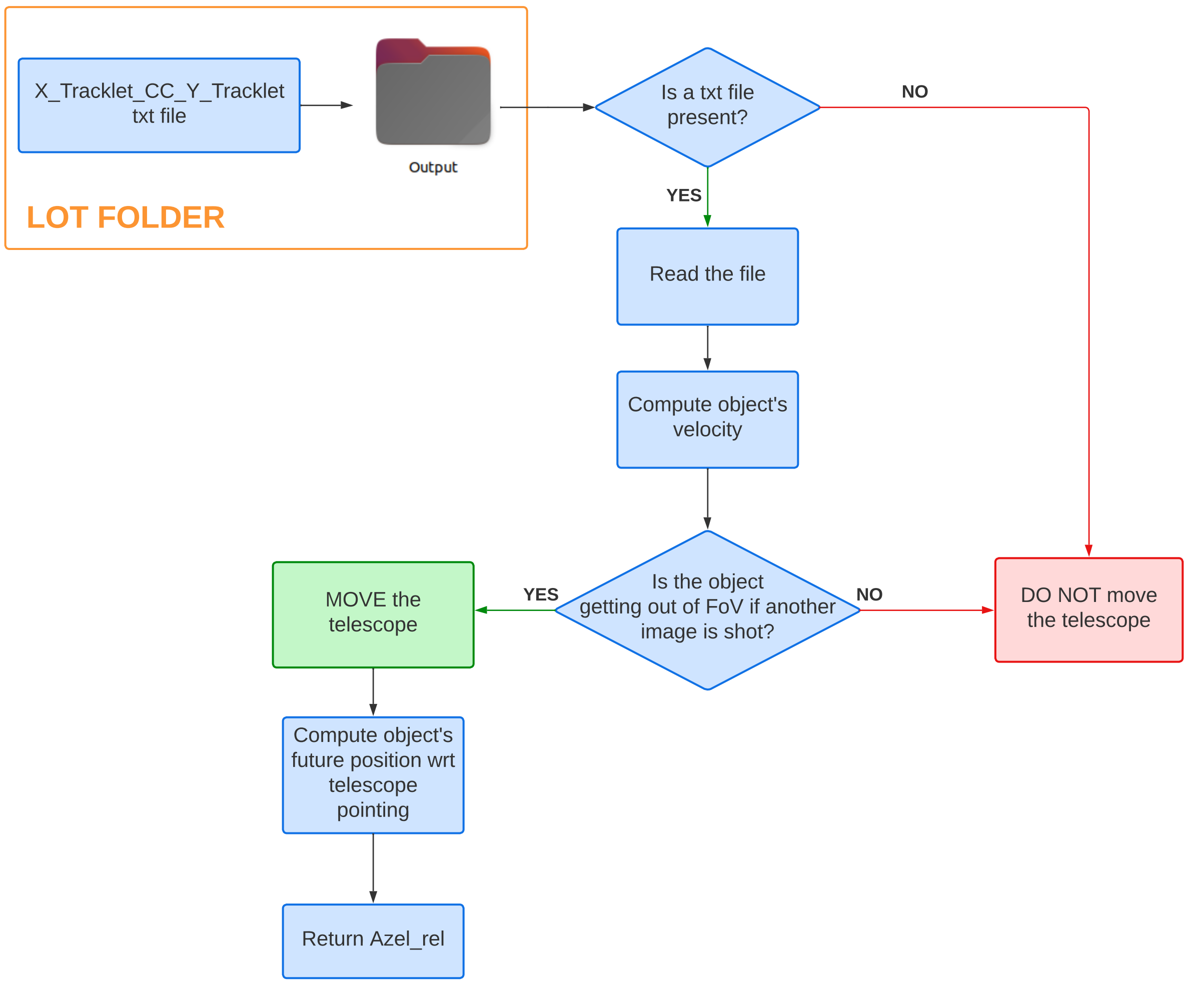

X_Tracklet_CC_Y_Tracklet txt file
Output
LOT FOLDER
Is a txt file present?
NO
YES
Read the file
Compute object's velocity
Is the object getting out of FoV if another image is shot?
YES
NO
MOVE the telescope
DO NOT move the telescope
Compute object's future position wrt telescope pointing
Return Azel_rel

Figure 3.18: TEMPO block diagram.

Input & Output

TEMPO algorithm is fully based on the text files LOT retrieves after the detection. As it was mentioned before, the "CC" files contain the information related to the centroids of the last two images detected. They are produced **only** in case LOT recognizes the two tracklets as belonging to the same image, therefore TEMPO is incapable of tracking an object if the telescope does not shoot at least two images of the same debris. The function searches inside the folder "LOT2/Output" the last file produced and analyzes it. In addition to the file, other data are needed:

1. the last two images' shooting time. The prediction requires to relate the centroid positions to an exact time instant. Hence, a mean value between the initial moment of the shooting time and its conclusion has been considered. Fig. 3.19 graphically represent the reason of this choice. The assumption is that the beginning and the end of the tracklet coincide with the position of the object when the camera starts the acquisition and when it finishes;

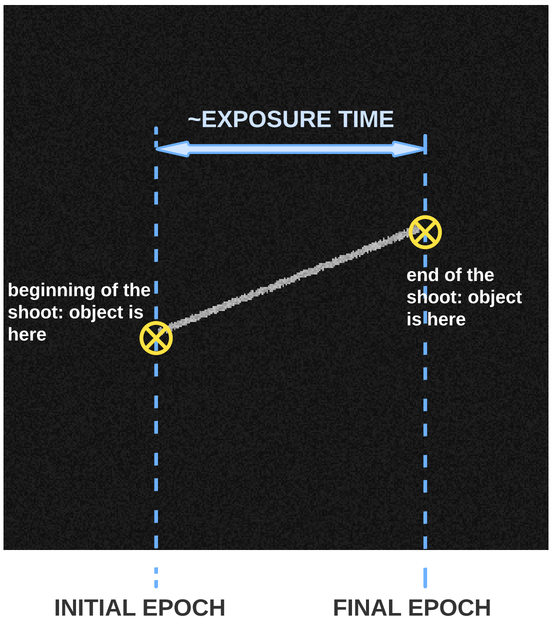

~EXPOSURE TIME
beginning of the shoot: object is here
end of the shoot: object is here
INITIAL EPOCH
FINAL EPOCH

Figure 3.19: Time assumption reference.

2. an estimation of the processing time of the algorithm;
3. the value of the field of view;
4. information regarding the last movement of the telescope and the possible failure of

LOT.

Point 4 is discussed in Section 3.4.2.

Regarding the output, Fig. 3.18 shows the two information TEMPO retrieves:

Res : the "response of the algorithm" is a boolean value which is updated as "True" if the telescope is allowed to keep on shooting and "False" if the telescope is required to move in another position;

Azel_rel : this variable stores the information of the relative position between the future pointing and the current one. It is defined as "None" type in case **Res** results as "True";

3.4.1. Architecture

The algorithm estimates the position of the tracklet's centroid inside the future image. To achieve this mission, it is essential to approximate the object velocity in the sky. Each centroid is related with a time instant, which is given as an input for TEMPO. The main passages followed to reach this objective are listed below:

1. the last text file "CC" is open from the "LOT2/Output" folder;
2. Centroids non-dimensional coordinates along x and y of the last two correlated images are extracted;
3. the coordinates reference frame is shifted to the center of the image, as well as for SPECTRA (see Fig. 3.15).
4. TEMPO implements a linear estimation of the velocity of the satellite:

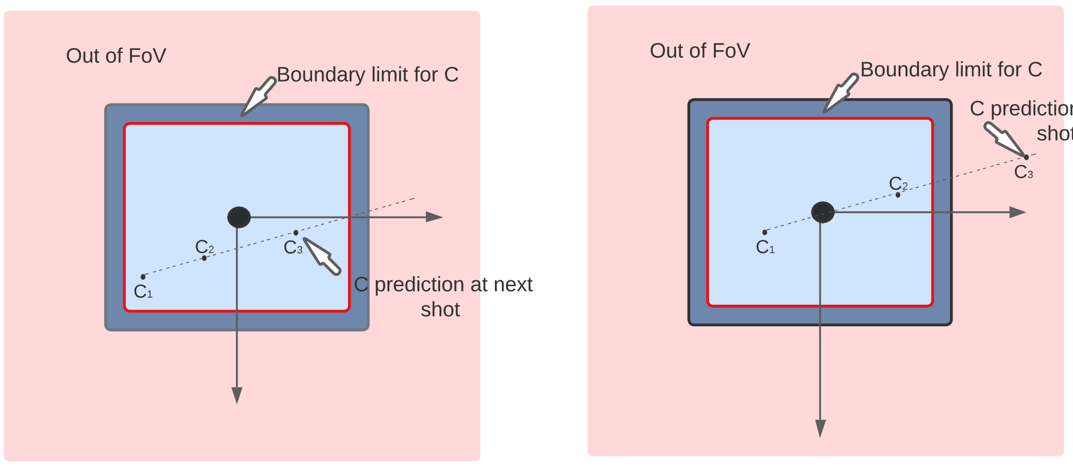
   $$V_{x,y} = \frac{C_{2\,x,y} - C_{1\,x,y}}{t_2 - t_1} \tag{3.1}$$

   where $C_{x,y}$ are the two image's centroid coordinates and $t_{1/2}$ their time instants in J2000.
5. The position of the centroid at future shot inside the image is evaluated:

   $$s_{x,y} = V_{x,y}\,\Delta t + C_{2\,x,y} \tag{3.2}$$

   Δt is estimated to be the same time interval between the last two images, so its value is given by $t_2 - t_1$.

6. Since the reference frame is shifted to the center of the image, the borders are defined at coordinates ± 0.5. If s_x or s_y get out of those edges, the next tracklet is not going to appear inside the image. It would be meaningful to point out the fact that the acceptable boundary for considering the tracklet inside the field of view was set at ± 0.4. This is due to the fact that the computed position is related to the center of the tracklet, that could partially get out of FoV otherwise, causing issues in both the detection and the tracking phases.



(a) Telescope must shoot another picture. (b) Telescope must be retarget.

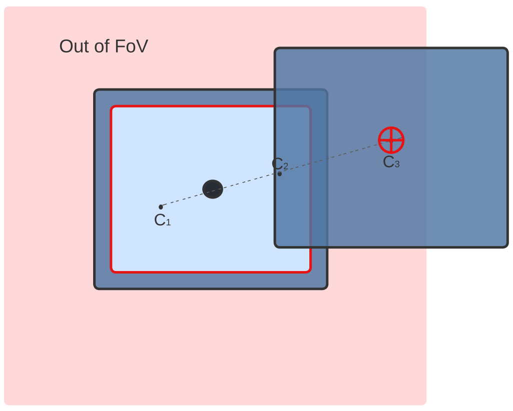
Figure 3.20: Graphical representation of future centroid's estimation.

7. If the trail is estimated to get out of FoV, a telescope repositioning is required;

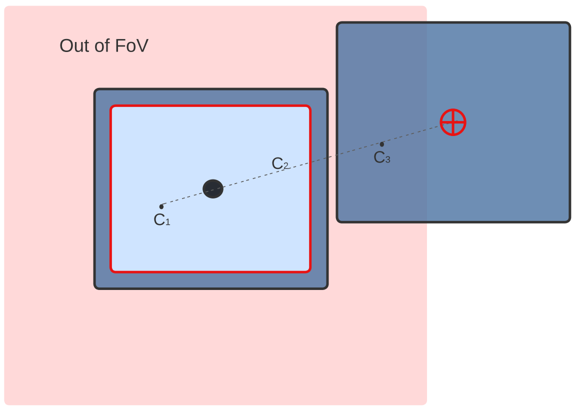
8. The position should be evaluated at a new Δt_{move} which includes:

   - the exposure time of the new picture;
   - the processing time of the algorithm;
   - the time required by the telescope to move;
   - an additional time needed to avoid the future tracklet to be exactly in the center of the image. Indeed, if an additional time is considered, the telescope "anticipates" the future tracklet, allowing a greater number of pictures shot without moving the instrument. A graphical representation of telescope re-pointing with and without additional time is shown in Fig. 3.21. It is clear that a prediction which takes into account a supplementary time when the telescope is moved, grants the shooting of a greater amount of pictures be-

fore a new movement, facilitating the tracking operation. Additional time is currently fixed by trial and error process, but a further implementation could relate it to the satellite's velocity.



(a) Telescope retarget without additional time.



(b) Telescope retarget with additional time.

Figure 3.21: Graphical representation of additional time importance.

9. The values are multiplied by the FoV to obtain the relative distance in degree between the current position of the telescope and the new one it needs to reach. The result is stored in the variable "Azel_rel"

TEMPO ends its operation returning the two output variables, which lead the telescope control part.

3.4.2. Off-nominal Cases and Solution

The detection failure leads to a series of off-nominal conditions in which TEMPO must be re-adapted. Hence, the motivation of Section 3.4 citing among all the input to TEMPO a set of boolean variables which contain the information regarding the movement of the telescope and LOT failure of the last **2** images. Indeed, a failure in LOT influences TEMPO's results even at the next iteration of the main loop. The section hereafter explains TEMPO algorithm in off-nominal cases assuming the main code just shot the picture number **N**:

- the telescope moved before image N. This case has been previously explained in Section 3.1.2 and its visual representation is shown in Fig. 3.10. It is effortlessy solved forcing the telescope to shoot another picture in the same frame.
- the detection failed on image N. SPECTRA recognizes the failure and communi-

cate a "True" boolean to TEMPO. When a detection fails, LOT does not produce the text file "N-1_$Tracklet_CC_N_Tracklet$.txt" containing the last two existing centroids of images N-2 and N-1, since the one related to image N was not produced. TEMPO opens the file, but following the nominal path it would retrieve the position of the centroid at image N, which was already shot. A graphical explanation clarifies the case in Fig. 3.22. As it is possible to notice, in this case $t_{1,2}$ are not

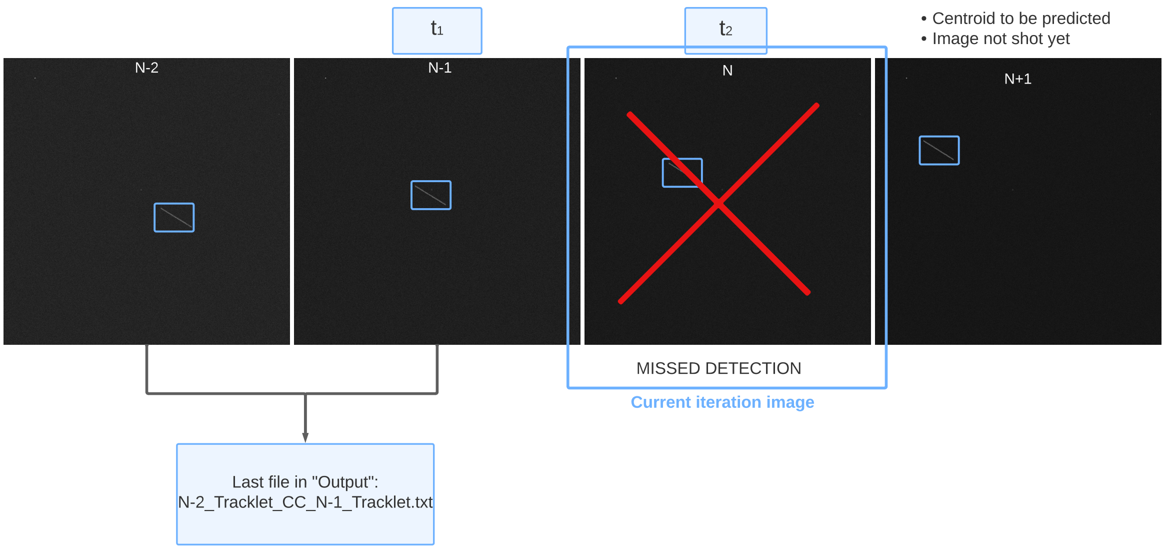


Figure 3.22: TEMPO input references if LOT missed the detection.

related to the centroid positions read inside the text file. This by itself would not cause a criticality, since Δt is approximately the same among the images. The difference with a nominal case lie in the fact that a linear estimation of the object future position with those data would lead to the prediction of the centroid inside the current image N and not the next one N+1.

The case is solved doubling the Δt when computing the future position $s_{x,y}$:

$$s_{x,y} = V_{x,y}\, 2\, \Delta t + C_{2\,x,y} \tag{3.3}$$

With a double Δt the tracking prediction skips the current image's centroid and predicts directly its coordinates in image N+1.

Figure 3.23 represents the case showing all the tracklets inside the four images taken into account in Fig. 3.22. No information are available for the centroid outlined with a red cross, while the two available coordinates are denoted as "$C_{1,2}$". For sake of clarity, the Δts used for estimating velocity and predicting the future centroid position are highligted with two different colors. The former is orange and the latter is white.

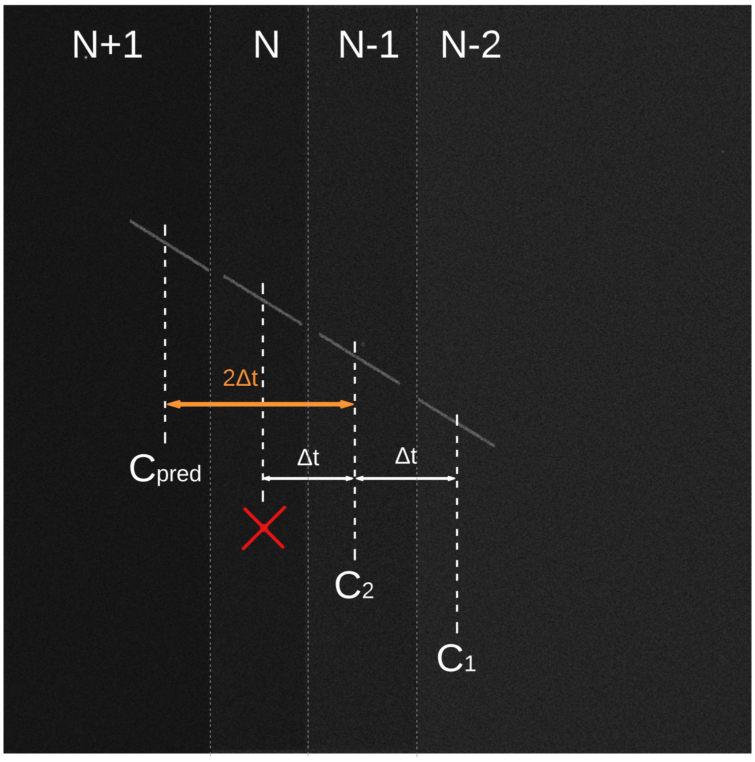


Figure 3.23: TEMPO solution if LOT fails.

If TEMPO deduces the centroid is getting out of FoV, the new Δt_{move} is computed taking into account all the components already mentioned, adding an extra time equal to an exposure time interval.

- LOT misses tracklet N and the telescope is re-pointed between N-2 and N-1. In this case TEMPO cannot retrieve any information since the telescope has not shot two images inside the same FoV yet, as it is possible to deduce from Fig. 3.24. Telescope shoot again. Regarding this case, the **Res** of the previous iteration is needed. **Res** input becomes a vector which contains all the **Res** returned at each loop repetition, so that an history of telescope motion is always available.
- LOT failed its detection at the iteration N-1. Even in this case TEMPO misinterprets the last "CC" file available. The representation in Fig. 3.25 shows how the time inputs $t_{1,2}$ of the last two images, would be erroneously assigned to the coordinates of centroids N-2 and N.

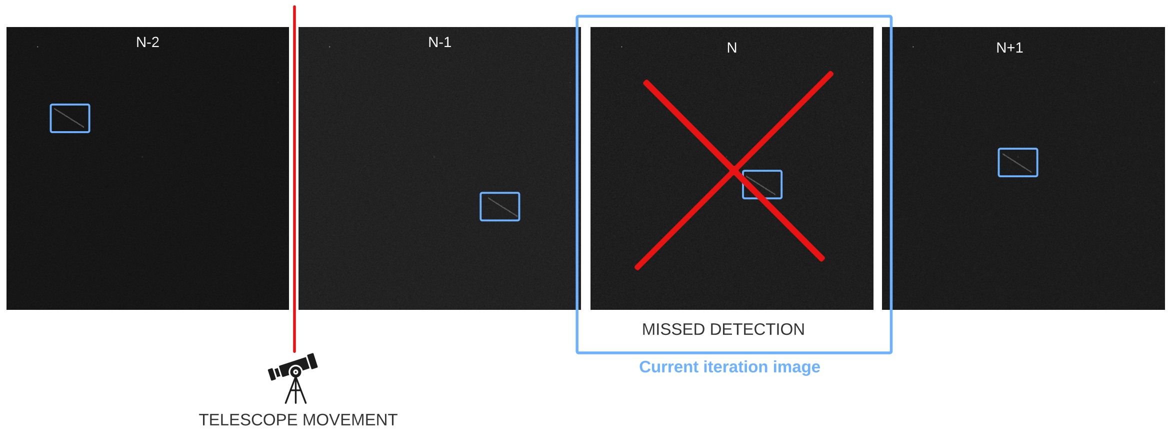


Figure 3.24: Image sequence if LOT missed the detection and telescope moved.

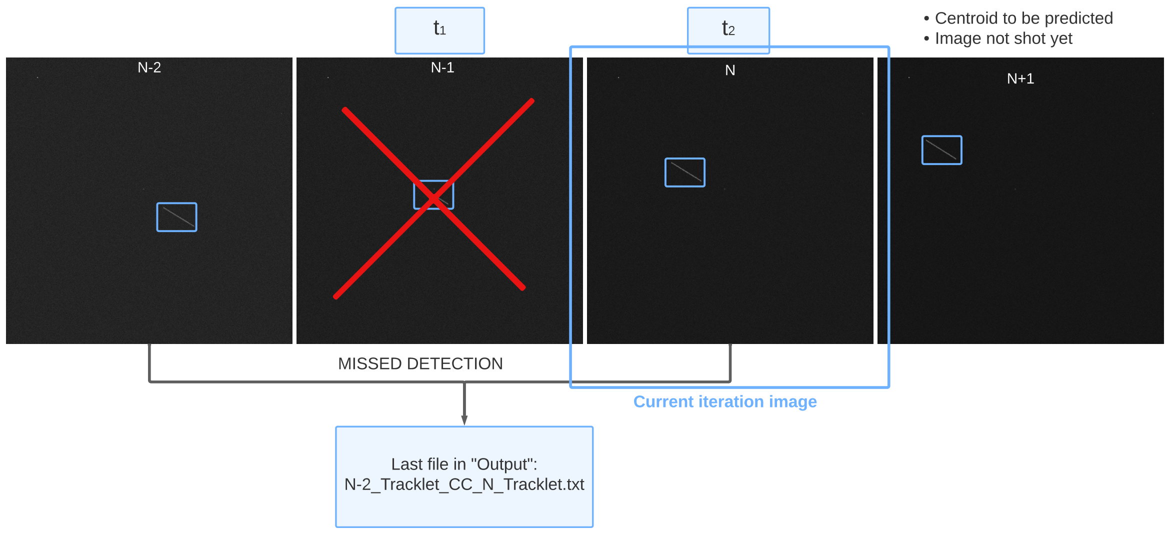


Figure 3.25: Image sequence if LOT missed the detection at previous iteration.

LOT never created a "CC" file containing the information regarding the trail at N-1, thus in a nominal case a time interval of $t_2 - t_1$ would be used for velocity computation between two non-consecutive centroids and the velocity along both x and y would result as a double value with respect to reality. As for the previous case, the problem is solved knowing that two time intervals passed between the two centroids in the file and taking it into account when estimating the linear velocity of the object.

$$V_{x,y} = \frac{C_{2\,x,y} - C_{1\,x,y}}{2(t_2 - t_1)} \tag{3.4}$$

The solution of this case is represented in Fig. 3.26, where all the trails are shown in a single image frame. As for Fig. 3.23 the color legend uses orange for highlighting the Δt implied in centroid's prediction and white for velocity estimation. $C_{1,2}$ labels are assigned to the images they belong to and their indicator can be read in the upper part of the figure. The red cross represents the missing information regarding the picture LOT failed to analyze.

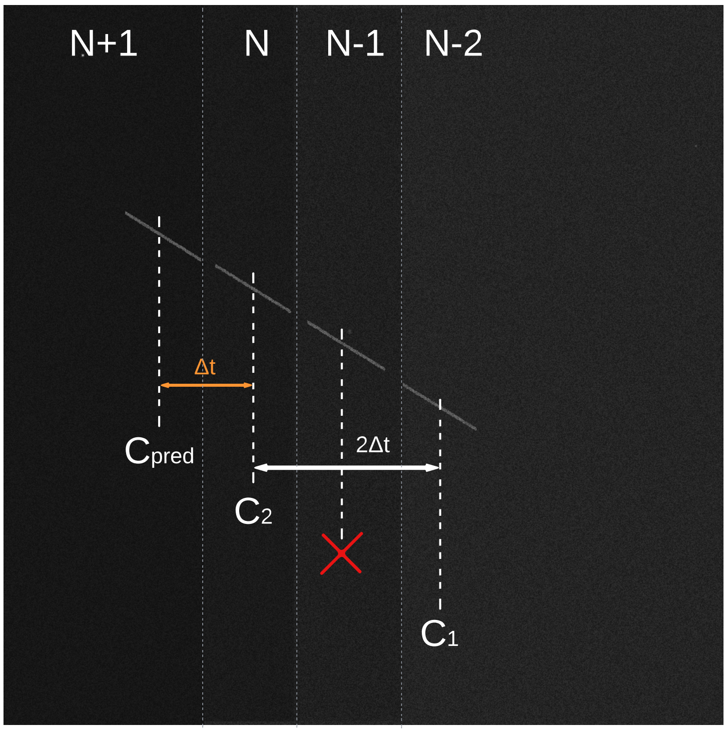


Figure 3.26: TEMPO solution if LOT fails at previous iteration.

- The previous case involves another off nominal conditions where TEMPO forces the telescope to shoot a further picture due to lack of information. It arises if in the sequence, before N-1, camera frame is moved. Even in this case, not enough pictures with the same FoV have been shot to enable TEMPO to perform its prediction.

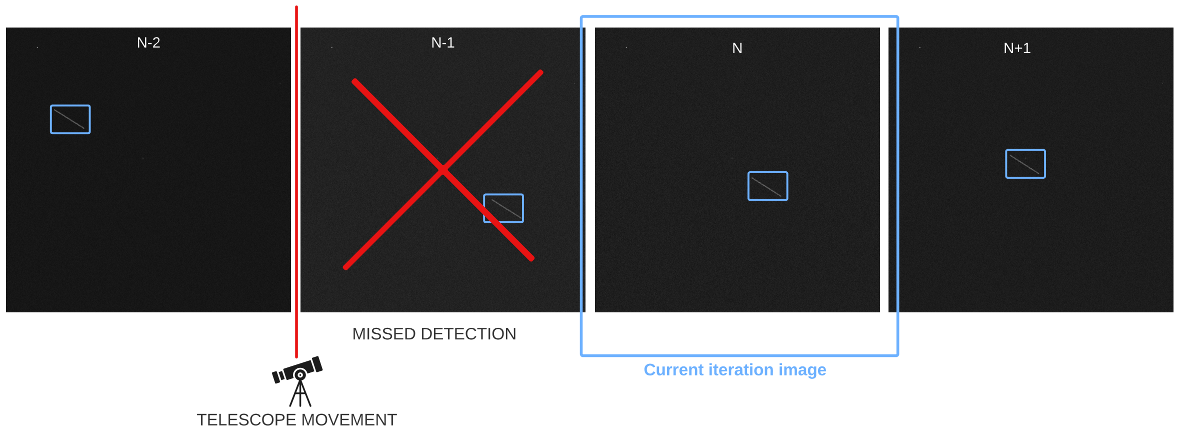


Figure 3.27: Image sequence if LOT missed the detection at previous iteration and telescope moved.

TEMPO recognizes all these cases by means of the booleans variables **Res** and **Check**. The algorithm requires the awareness of the last two values assumed by them over loop iterations. While for **Res** this means knowing TEMPO response at loop iteration N-2 and N-1, the **Check** output is required at current loop N and previous N-1. This is due to the fact that **Check** carries the information of the missing detection and it gets updated by SPECTRA **before** TEMPO algorithm, while **Res** at current iteration is available only when TEMPO ends its tracking and prediction phase. Next section will explain how these variables are stored inside the loop.

3.4.3. Criticality & further implementations

Many critical cases have been solved, but TEMPO is able to operate **only** within particular boundaries.

- if the velocity of the satellites exceeds $4°/s$ [1], the telescope is physically unable to follow the track and TEMPO returns an error message.
- TEMPO requires at least two pictures showing the object to be tracked without frame retargeting. This critical point involves that if, starting a tracking operation, LOT misses the second picture detection and afterward the object gets out of FoV, the target is lost. Further implementations where the condition on the telescope is not required could be developed exploiting the file generated by SPECTRA, although this goes beyond the aim of this thesis.
- The previous critical point further restricts the velocity range of objects that can be

tracked. As explained in Section 3.4.2, if the detection is missed TEMPO requires at least **3** images for a proper prediction. If the satellite is exceedingly fast and it is not possible to shoot three pictures of it in the same reference frame, the first missed detection would represent a criticality that could not be solved.

3.5. STRATEGIC: Telescope control

The software is called Space debris TRacking Algorithm for TElescope Guidance and Instantaneous Control (STRATEGIC) and it exploits all the algorithms mentioned before to serve the objective of tracking an unknown object for its whole visible passage. Each section heretofore properly explained all the inputs and outputs required by the designed functions, thus the section hereafter is dedicated to an overall look of the main code, namely the software STRATEGIC, to enhance the comprehension of how each block interacts with the others.

3.5.1. Architecture

STRATEGIC is divided into 6 main blocks: SETTINGS, IMAGE PROCESSING, DETECTION, TRACKING, PREDICTION and CONTROL.

SETTINGS At the very beginning of the software, the program is launched. INDI server is started and the telescope and the CCD camera are connected to it and YOLO v5 model is uploaded to avoid time wasting after the beginning of tracking phase. In case of a telescope simulation, the object TIG is about to print on the images must be chosen inside the SCOOP file and a "fake time" starts. The main properties of the telescope are set, such as the exposure time, the FoV and first telescope coordinates. The configurations applied to simulate the telescope used for STRATEGIC are:

- Telescope name: `Telescope Simulator` ;
- Camera name: `CCD Simulator` ;
- Host: `Local Host` ;
- Port: `7624` .

Before starting the loop, a counter begins. The counter is included inside the class containing the telescope settings and it a fundamental variable for the whole software. The counter gives images their name, "counter_Tracklet.png", LOT produces the corresponding text file "counter_Tracklet.txt" and SPECTRA checks its value to ensure LOT did not miss the detection.

The telescope shoots a picture of the sky. Since the satellite observed leaves a trail on the image, a linear assumption is made and the center of the trail (if any exists) is estimated to be at a mean time between the beginning and the end of the picture, which lasts at least the exposure time interval, as graphically explained in Fig. 3.19. This input is required both by SPECTRA for tracking and by TEMPO for satellite velocity estimation.

IMAGE PROCESSING YOLO requires the image format as PNG. Thus, the FITS to PNG conversion is an unavoidable passage STRATEGIC needs to actuate before moving to the detection block. TIG will be removed whenever STRATEGIC is connected to a real telescope. In any case, the amount of time TIG needs to process the image has been taken into account; indeed a real image requires more operation of filtering and background noise removal than a synthetic one before applying the detection.

DETECTION : YOLO model is recalled inside the function LOT, which searches for a new image inside its folder and applies the detection. If the tracklet is not recognised, the image passes to RID's folder. From that exact moment, RID has 1s to complete its detection before LOT moving on with the detection. RID should be launched before starting STRATEGIC, since the code just "sleeps" (`time.sleep()` is the proper line command) until an image arrives to its folder.

TRACKING SPECTRA uses the text file generated by LOT-RID to produce a dataset of debris passages in Azimuth-Elevation coordinates together with the time instants at which they have been detected. Moreover, a boolean variable is set as "True" if SPECTRA recognizes LOT did not produce any file.

PREDICTION TEMPO receives as inputs the information about the history of missed detection and precedent telescope movements, together with the last two centroid position and their corresponding time instants. It produces a response, which is "True" if the telescope can shoot another image of the object keeping the same FoV and "False" if the telescope needs to move to keep tracking the debris. The latter case also produces a vector containing the relative distance in degrees between the actual position of the telescope and the required one of tracking. It is meaningful to point out that TEMPO **never** requires any pre-knowledge of the object orbit to predict its next position.

CONTROL STRATEGIC converts the coordinates received by TEMPO in absolute Azimuth and Elevation degrees and updates the telescope settings with the new values. The telescope is moved to the required position and the response of TEMPO is stored in a vector to keep track of the telescope progression.

At this point, the counter is updated for next shot and the loop begins again shooting another picture. The whole process is schematized in Fig. 3.28

Stop of STRATEGIC

The "while loop" is not designed to arrive to an end point, so far it needs to be stopped manually. During the developing of this thesis the code used to stop whenever TIG could not print a tracklet inside the picture, thus when the tracklet was lost or the SCOOP file ended the information regarding the object's passages. A further implementation of the code could add a "mode" in which the telescope randomly scans the sky, searching for a new track to follow. If TIG is not connected to STRATEGIC, an "empty" image would not produce any error messages, the "Output" folder would not contain any file and TEMPO would just communicate to the telescope to keep on shooting.

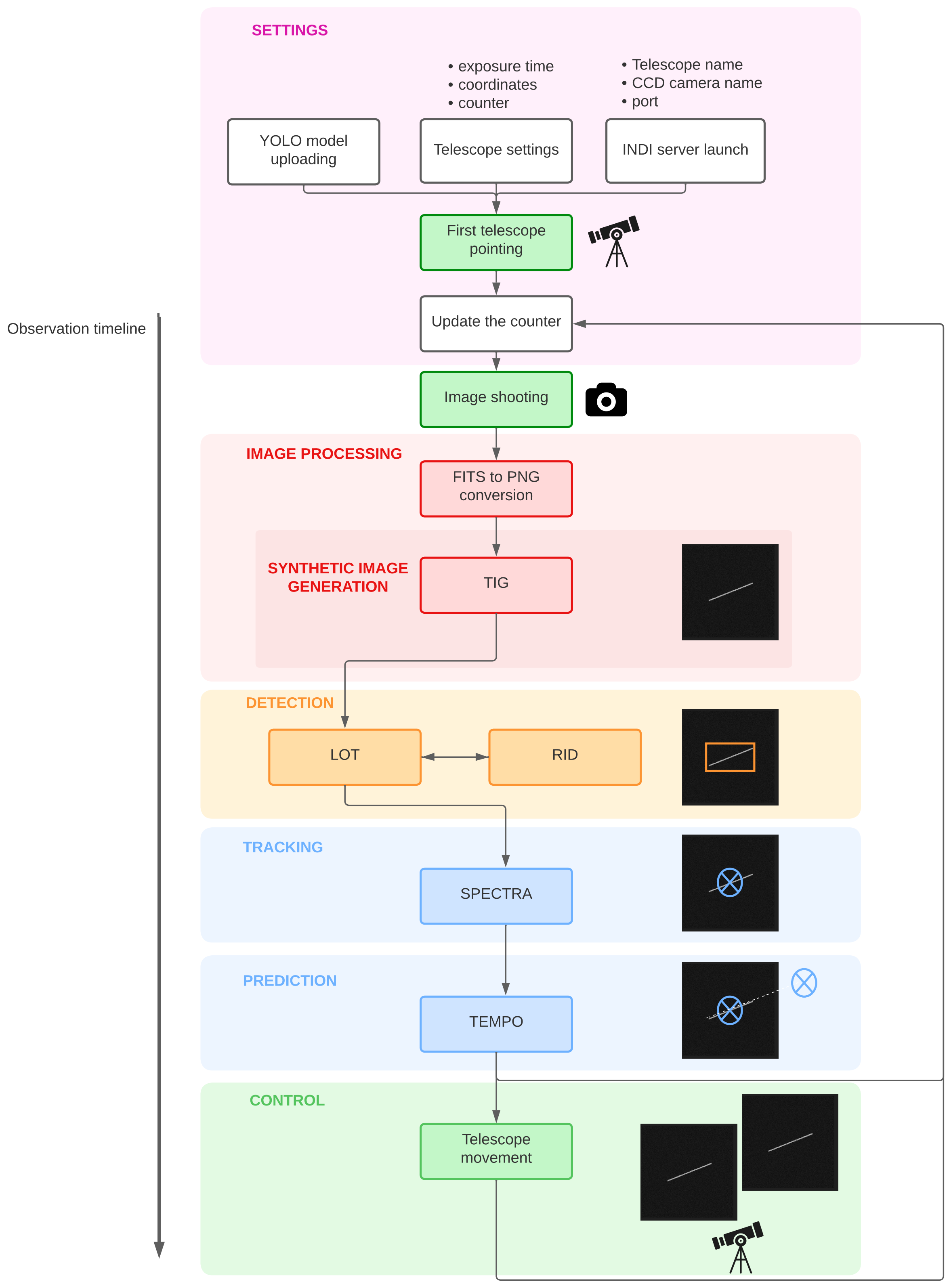


Figure 3.28: STRATEGIC block diagram.

3.5.2. Considerations

For a matter of time, celestial coordinates have been considered as if they were taken in the same reference frame. Actually, INDI server accepts the coordinates in Right Ascension-Declination frame, while TIG accepts **only** Azimuth-Elevation coordinates. If the correct conversion had to be produced at the time being, Ra-Dec coordinates would be converted into Azimuth-Elevation after the image shot and reconverted before the telescope Ra-Dec updating. This would increase the computational time in a real situation, where a function similar to `azel_shift`, eventually called `radec_shift`, could be implemented just for converting the relative position between the two telescope frames given by TEMPO directly in Ra-Dec coordinates.

STRATEGIC is a real-time dependent software. Therefore, debugging involved many complications due to time flowing even when it stopped the running code. For this reason, another version of STRATEGIC was implemented, where time was a variable updated with fictitious values representing the time intervals required by the functions to operate. This second code has been extremely worthwhile during the individuation of all the off-nominal cases involved in TEMPO prediction and could still be useful in case STRATEGIC needs to be enhanced.

STRATEGIC was built trying to follow the track of a known object, with available TLE and SCOOP file. Even if it has been tested for many objects with different trajectories, many parameters could still be improved, especially when velocities are different from the ones considered during STRATEGIC's development. This considerations come from a visual analysis of the results given by STRATEGIC, explained in Chapter 4. Nevertheless, the aim of STRATEGIC is to keep the desired object inside the field of view of the CCD camera and this objective was perfectly accomplished.

4 | Results

Before experiencing STRATEGIC in real life, a guideline was necessary during the operations of software construction. The software has been created in a simpler environment, where time-dependencies were manually adjusted, the telescope shot simulated night sky pictures and a single object with completely known passage needed to be tracked. In this way all the criticalities were identified and corrected and finally, a real time-dependent software is now available and capable of tracking unknown objects. STRATEGIC has been tested with a set of satellite passages, each one with different trajectories, to prove its consistency. The results were obtained in the following way:

1. An object is chosen within the SCOOP file dataset;
2. initial observation moment and initial coordinates are set to simulate the first picture in which a random object is found;
3. after the first shot STRATEGIC automatically follows the object.

Each time STRATEGIC is launched, results may vary. This is due to different factors:

- TIG prints on images tracklets with a random thickness.
- this causes a random variation on LOT detection time and random LOT failures;
- LOT failures cause readjustments on the code which modify the nominal times used for tracking;
- computer processing may require different time intervals for the same algorithm. A faster PC can retrieve distinct results than a slower one and all results may vary at each iteration of the software.

Some of the tested objects are listed and their results are represented hereafter.

4.1. First Object

The first tracked object is characterized by the following observation data:

Exposure time	3s
FoV	2°
Object number	0
Initial observation time	06 DEC 2019 02:32:28.000
Available final time instant	06 DEC 2019 02:35:03.000
First Right Ascension pointing	150.494330°
First Declination pointing	5.885350°

Table 4.1: First object data

Using these information, STRATEGIC pointed the telescope on the first available data of the object and then shot 33 images during its passage. The last image matches SCOOP final data at the time indicated in Table 4.1, thus the software interrupted since TIG did not have enough information to print the tracklet on the image. Next figures show the pictures STRATEGIC shot with the simulated telescope of the satellite passage. Figure 4.1 shows the whole set of images retrieved. Zooming in, in each image a tracklet is visible, which means STRATEGIC accomplished its objective.

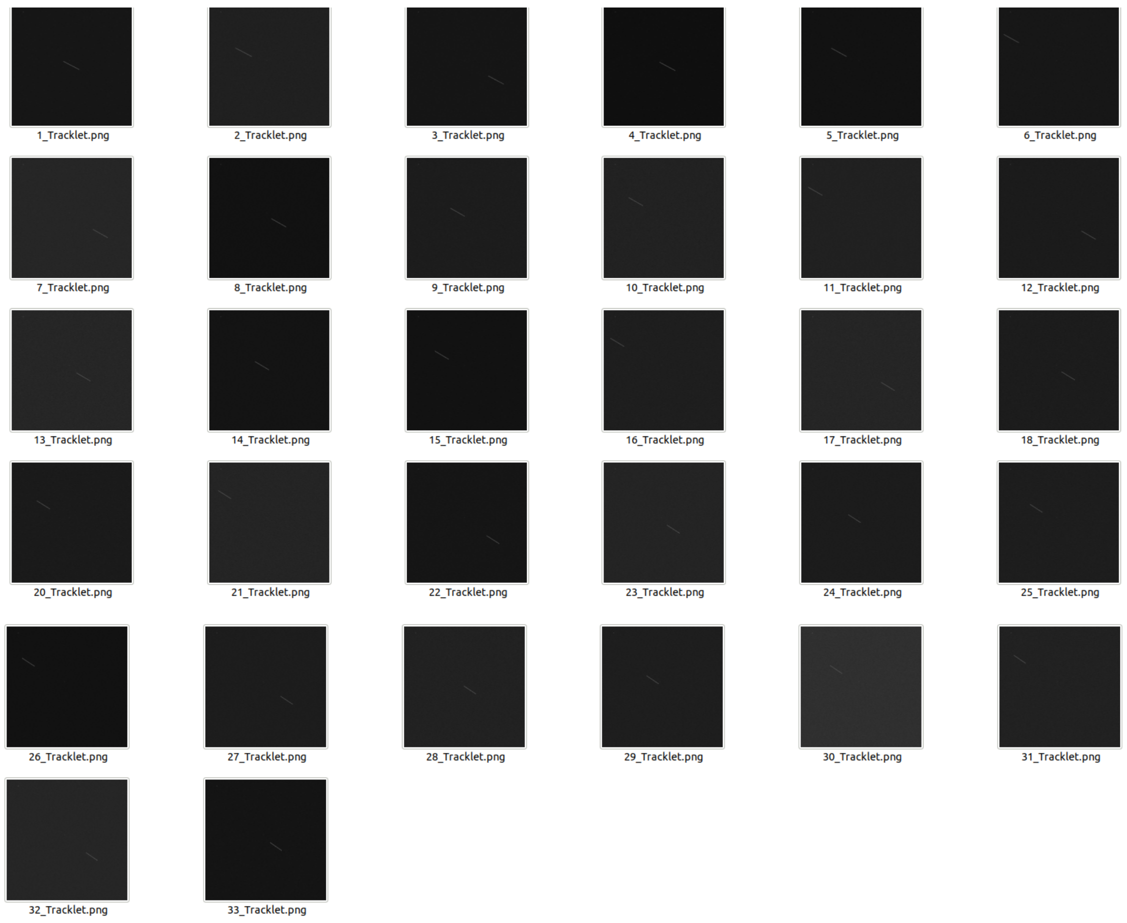


Figure 4.1: Complete set of images.

A reduced set of images is shown below to better highlight the tracklet positions.

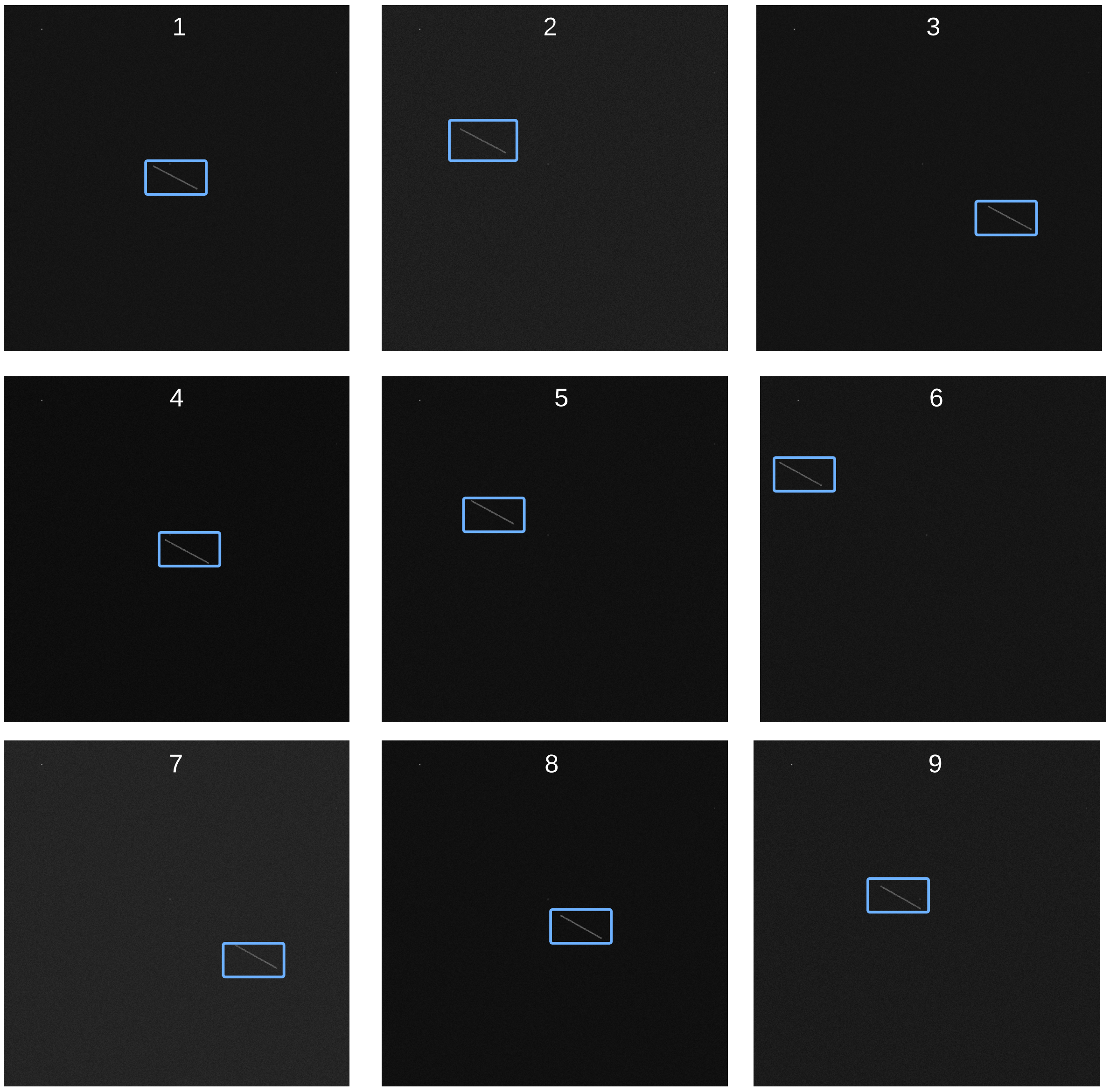


Figure 4.2: Zoom on the first 9 images.

Nonetheless, even in this case a simple set of images is not capable of properly representing the satellite passage in the sky. Its trajectory has been reconstructed using the information retrieved by TEMPO and related to the telescope movement. The representation superposes on the same frame all the tracklets captured whithout moving the telescope. To better highlight the tracks, the masks produced by TIG have been employed instead of the real images. Masks pictures are black/white representation of the images, where the whole background is black and **only** the tracklet has white pixels. This kind of image has never been involved in the process of tracking and control, but it emphasizes the visualization of the tracklets inside the images.

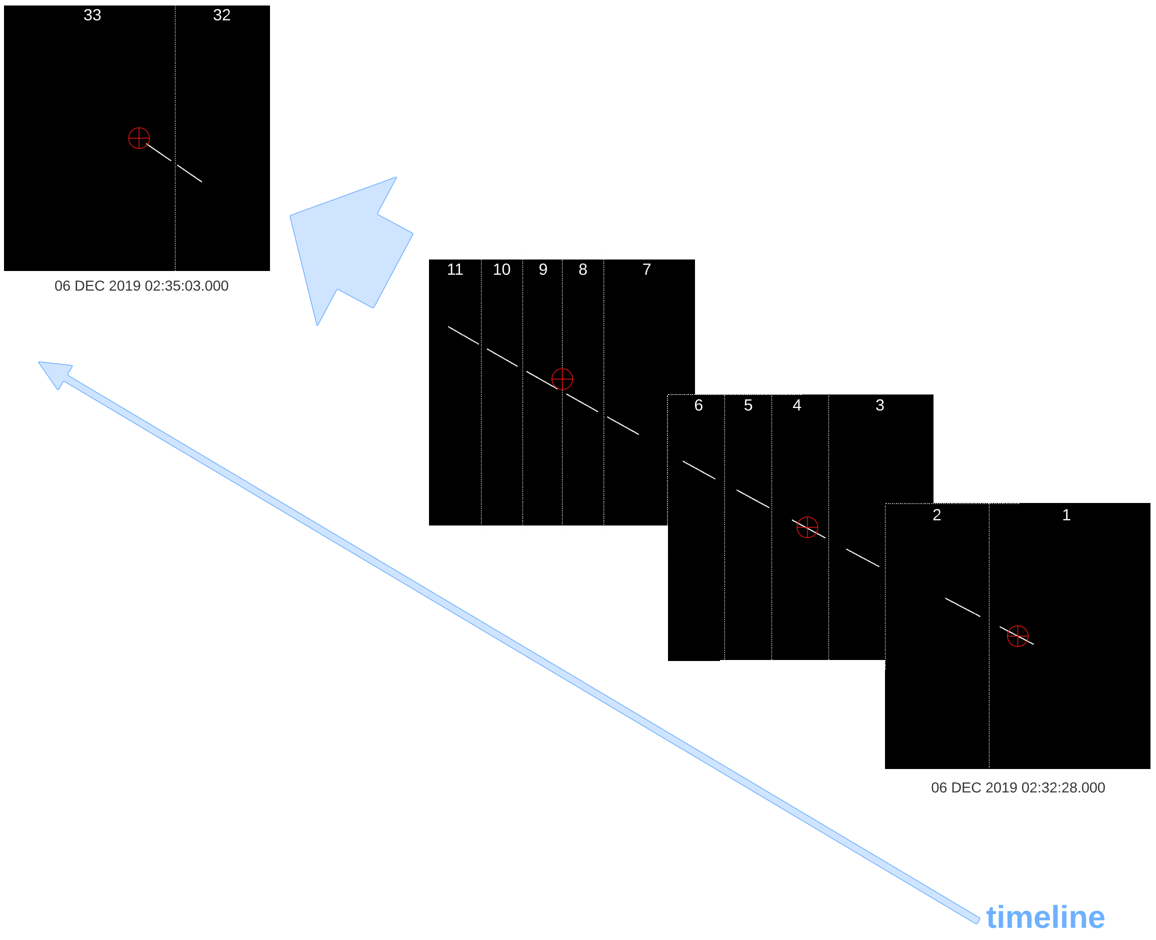


Figure 4.3: Pictures sequence.

As it is possible to notice, the additional time discussed in Section 3.4 is still perfectible, even though the objective is achieved and the last image represented is the one in which TIG reaches interpolation limit of data furnished by the SCOOP file and cannot print any more tracklets. About once every ten times STRATEGIC is launched, LOT does not recognize track 1 or 2. This still represents a criticality if RID does not end its substitutive detection in time, since the tracklet gets lost: indeed, the third satellite position is already out of the initial camera frame and if the detection misses one of the first two images TEMPO does not have enough information to re-point the telescope to keep on tracking the debris.

4.2. Second object

STRATEGIC software shot pictures of many other objects available in the SCOOP file, even modifying part of the settings. In this case, exposure time was reduced to 1.5s.

Exposure time	1.5s
FoV	2°
Object number	1
Initial observation time	06 DEC 2019 02:49:50.000
Available final time instant	06 DEC 2019 02:52:41.000
First Right Ascension pointing	16.430380°
First Declination pointing	6.629000°

Table 4.2: Second object data.

This time, STRATEGIC shot 69 images with a streak in them before TIG ended the available data on SCOOP. Since all the results have been previously shown for Object 1, a different representation creatively shows how the tracks should be seen from an observer using the "CCD simulator" to look at the object passage. Figure 4.4 could not contain the pictures of **all** the positions assumed during the observation, thus the first and the last two frames are shown. Here the additional time computed by TEMPO (and visible between the first two image frames) better fits the telescope movement optimization, causing a reduced superposition between image frames with respect to the first object tracked. Furthermore, different computational times of the machine are noticeable by the different distances between consecutive trails inside the same telescope frame.

Moreover, this object better shows the curvature of its trajectory between the initial and final time instants.

RID dedicated code printed the alert that inside the 69 images shot, the number 23 and 44 were not recognised by LOT and thus they have been analyzed. RID took exactly 0.703s for image 23 and 0.667 for image 44. Since it was less than 1s, namely the waiting time planned by LOT, their detection was completed, the retrieved information were placed in the correct folders and STRATEGIC kept on with the algorithm treating them as nominal cases.

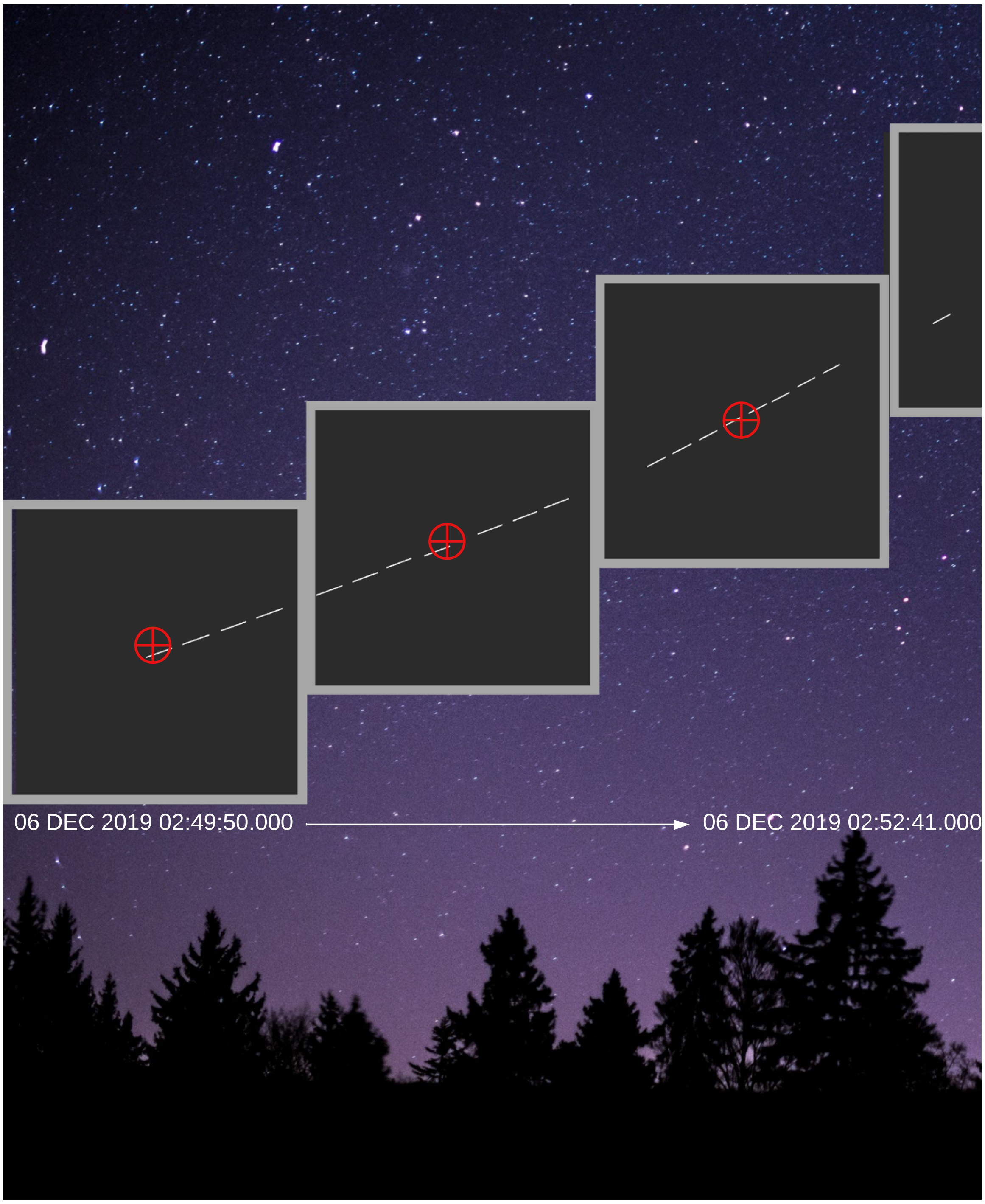


Figure 4.4: Object 2 passage representation in the sky.

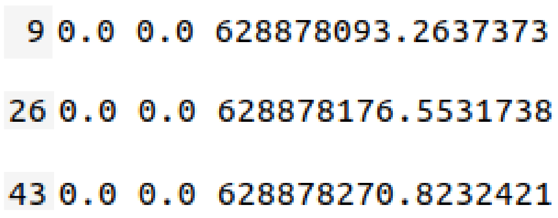
4.3. Third object

The characteristic of this particular object is a trajectory with extremely low slope and avaliable SCOOP time of more than 5 minutes, double than the other two objects taken in consideration. Hence, 62 images were obtained over the observation time. This way, STRATEGIC demonstrated its ability in following tracklets even for longer time intervals. Exposure time has been increased to 4s, thus the streak on images appear longer than the ones previously analyzed.

Exposure time	4s
FoV	2°
Object number	2
Initial observation time	06 DEC 2019 04:19:27.000
Available final time instant	06 DEC 2019 04:25:15.000
First Right Ascension pointing	207.309070°
First Declination pointing	6.007530°

Table 4.3: Third object data

LOT could not detect tracklets 9, 26, 28 and 43. In this case, RID required respectively 1.417s, 1.122s, 0.788s and 2.290s to complete its detection. Therefore, according to LOT waiting time, Tracklet 28 was added to SPECTRA file data, while the others followed the path of their correspondent off-nominal cases. Tracklet 9 is indicated in Fig. 4.6 with a white arrow. TEMPO spent an higher amount of time than the other iterations to predict the next position, thus the successive trail is more distant than the others. None of them caused a criticality in the observation, which arrived at the end of TIG limit either way. Next picture shows SPECTRA file leaving empty row 9, 26 and 43, namely the images with RID detection slower than 1s. As expected, row 28 is filled with the correct information.



```
 9 0.0 0.0 628878093.2637373

26 0.0 0.0 628878176.5531738

43 0.0 0.0 628878270.8232421
```

Figure 4.5: Object 3 SPECTRA off-nominal results.

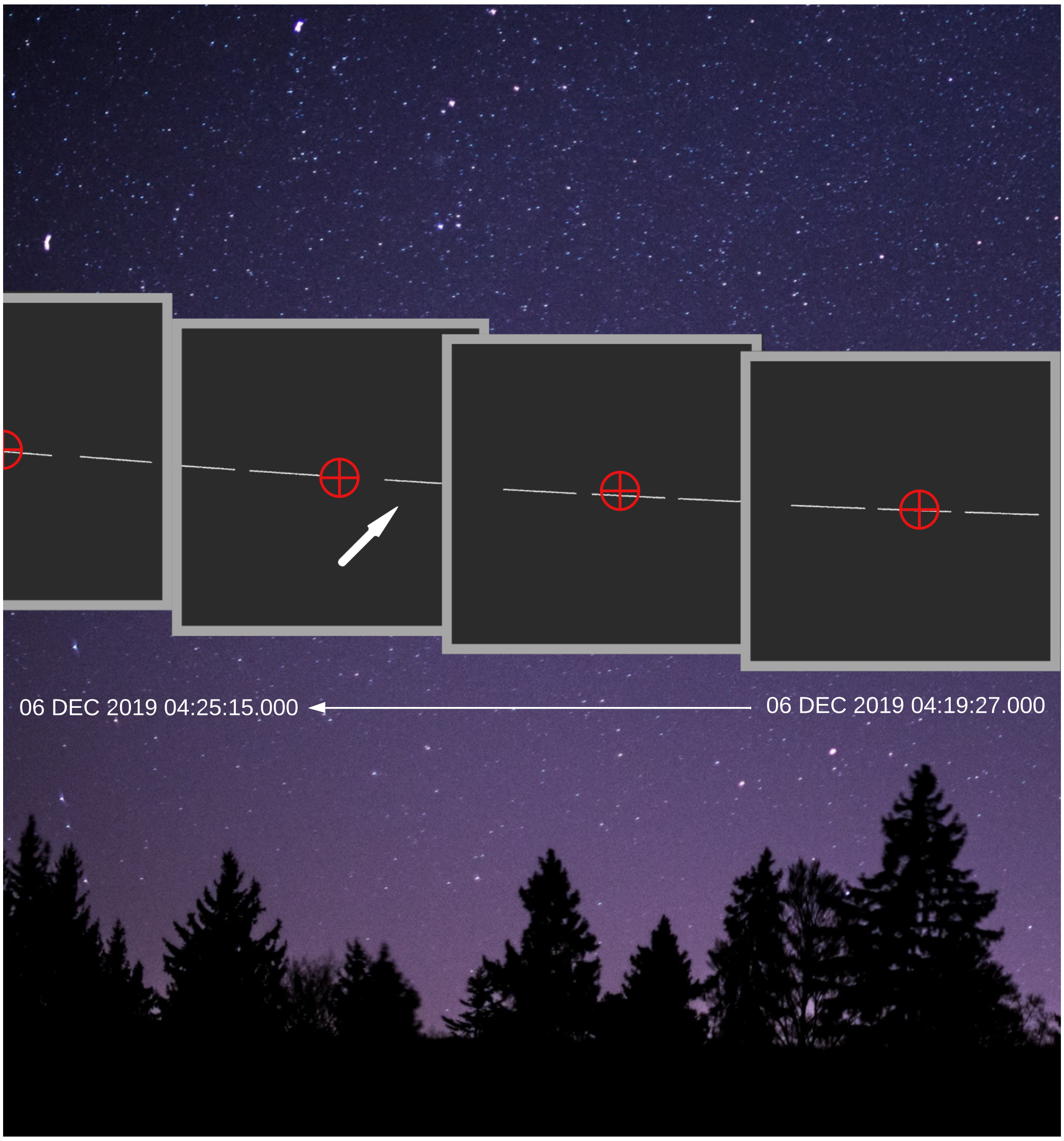


Figure 4.6: Object 3 passage representation in the sky.

5 | Conclusions and future developments

Space debris observation and surveillance is rising its importance over time, due to the increment of launched satellites per year and the consequent amount of collisions due to overcrowding of the main orbital regimes, GEO and LEO. Now, the awareness of this problem is spreading and many solutions are actuated directly in the design phase of a satellite, such as a safe disposal at the end of its operational life. This precautions are not enough to completely solve the problem. Satellites still need to dodge out of their way an arising number of debris and each one of them is able to cause collateral damages. To avoid missions failure, it is fundamental to keep track of the debris population in a fast and efficient way. STRATEGIC is a fast, innovative, modular software able to provide data on a completely unknown object and predict its movement with linear assumptions. Its innovation comes from the employment of Artificial Intelligence for real-time images detection, a new field which is exponentially growing and enhancing its performances. This is one of the main reasons STRATEGIC's potential is unlimited: the possibility of substitution of the different blocks constituting the whole system allows to update the software with new tools and still be helpful for unknown object tracking whenever a faster detection or a faster mount will be available for a telescope. As a matter of fact, STRATEGIC already resulted to be incredibly effective for any kind of trajectory and for objects moving at less than 1°/s. In facts, it demonstrated how, after proper corrections applied to the eventual off-nominal cases caused by detection failures, the software was able to fully track object passages with no difficulties, perfectly accomplishing its mission. SPECTRA files collect all the position the tracked debris assumed during its whole passage and in future these data could be correlated with existing catalogues, updating known object orbits or adding a new debris to the existing dataset, actively contributing to Space debris surveillance. Regarding object detection, RID is able to find **any** track recorded by the CCD camera. STRATEGIC could be improved by selecting a criterion for choosing which tracklet to follow, such as the correlation with catalogues for chasing the unidentified object or the confidence level returned by RID. The latter choice would increase the possibility of selecting the tracking of an effective trail and not a cloud or a random background noise.

Further implementations could optimize also the timing performance of the software, still at its prototype phase. Furthermore, STRATEGIC results still need to be tested on real images, with a real telescope. This phase will surely demonstrate the limits of STRATEGIC. It will require some corrections, as any tool examinated for the first time in a real environment, but it can surely bring an innovative contribution to space debris surveillance from ground-based telescopes.

Bibliography

[1] Matteo Caruso and Tommaso Tognozzi. Space debris tracking control of the polimi space surveillance telescope for orbit determination. Master's thesis, Politecnico di Milano, 2018.

[2] NASA. Space debris definition, 2021.

[3] ESA. Esa space environment report, 2022.

[4] ESA. How many space debris objects are currently in orbit?, 2022.

[5] ESA. Hubble's impactful life alongside space debris, 2022.

[6] Marric Stephens. Space debris threat to geosynchronous satellites has been drastically underestimated. *Electronic document available at https://physicsworld.com/a/space-debris-threat-to-geosynchronous-satellites-has-been-drastically-underestimated/*, 2017.

[7] Holger Krag Tim Flohrer. Space surveillance and tracking in ESA's SSA programme. Technical Report 18, ESA Space Debris Office Ed. T. Flohrer & F. Schmitz, roc. 7th European Conference on Space Debris, Darmstadt, Germany, 4 2017.

[8] A.E. Potter. Ground-based optical observations of orbital debris: A review. *Advances in Space Research*, 16(11):35–45, 1995. Space Debris.

[9] GAUSS srl. Astronomical observation from castelgrande observatory, 2018.

[10] Jason Calvi, Alessandro Panico, Riccardo Cipollone, Andrea De Vittori, and Pierluigi Di Lizia. Machine learning techniques for detection and tracking of space objects in optical telescope images. 11 2021.

[11] Matt Williams. What is the difference between a ccd and cmos camera sensor?, 2021.

[12] Mauro Massari. *Optical instruments ppt presentation.* Politecnico di Milano school of industrial and information engineering department of aerospace science and technology, 2021.

[13] D. Mehrholz, L. Leushacke, W. Flury, R. Jehn, H. Klinkrad, and M. Landgraf. Detecting, tracking and imaging space debris. *ESA bulletin 109*, 2002. Space Debris.

[14] ESA. Tira space observation radar, 2017.

[15] Riccardo Cipollone and Andrea De Vittori. Machine learning techniques for optical and multibeam radar track reconstruction of leo objects. Master's thesis, Politecnico di Milano, 4 2020.

[16] Jasem Mutlaq and Akarsh Simha. Kstars, 2022.

[17] INDI Library. Indilib.org, 2022.

[18] Heiko Wilkens. What is a ser file?, 2009.

[19] NoirLab. Fitsliberator website, 2022-06.

[20] Jon Fincher. Python ides and code editors (guide), 2020.

[21] Hyperphysics. Orbits and the ecliptic plane, 2020.

[22] Unknown. Right ascension-declination from a local observer, 2020.

[23] Mohammed Chessab Mahdi. Tigrisat orbital motionsimulation and analysis. *Journal of Control Engineering and Technology*, 5:1–8, 01 2015.

[24] Md Akhtaruzzaman, Sadakatul Bari, Syed Hossain, and Md. Mahbubur Rahman. Link budget analysis in designing a web-application tool for military x-band satellite communication. 8:17–33, 07 2020.

[25] Iharka Szücs-Csillik and Dora Poputa. The astro-biblio-students program. *Didactica Mathematica*, 33:101–111, 12 2015.

[26] GitHub. Yolo v5, 2022.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[28] GitHub. Yolo v5-bounding boxes, 2022.

[29] Jason Calvi. Machine learning techniques for detection and tracking of space objects in optical telescope images. Master's thesis, Politecnico di Milano, 2021.

List of Figures

List of Tables

List of Symbols

Name	Description	Unit
$\boldsymbol{GEO}$	GEOstationary Orbit	-
$\boldsymbol{LEO}$	Low Earth Orbit	-
$\boldsymbol{TIG}$	Tracklet Image Generatior	-
$\boldsymbol{YOLO}$	You Only Look Once	-
$\boldsymbol{STRATEGIC}$	Space debris TRacking Algorithm for TElescope Guidance and Instantaneous Control	-
$\boldsymbol{LOT}$	Linear Orbit Tracker	-
$\boldsymbol{GEO}$	GEOstationary Orbit	-
$\boldsymbol{RID}$	Real Image Detector	-
$\boldsymbol{SPECTRA}$	SPace debris TRacking	-
$\boldsymbol{TEMPO}$	Telescope Movement Prediction and Oversight	-
$\boldsymbol{res}$	resolution	pixel
$\boldsymbol{FoV}$	field of view	°
$\boldsymbol{relpos}$	relative position between centroid and telescope ponting	[-]
$\boldsymbol{Res}$	response for moving the telescope	Boolean
$\boldsymbol{WRONG, Check}$	Information on LOT missing detection	Boolean
$\boldsymbol{Azel\_rel}$	relative distance between current and future position of the telescope	°
$\boldsymbol{C_{x,y}}$	Tracklet centroids	[-]
$\boldsymbol{V_{x,y}}$	Object velocity estimation	1/s
$\boldsymbol{s_{x,y}}$	Object estimated distance from the telescope pointing	[-]

Acknowledgements

I would like to personally thank my advisor Prof. Di Lizia, who knew how excited I was regarding the idea of working with a telescope and proposed me this topic.

One thousand thanks to my co-advisors Riccardo Cipollone and Andrea De Vittori, who followed the development of this thesis from the very beginning and spent a lot of their time supporting me when I wasn't sure about how to proceed. I would also like to thank Jason Calvi, since his work was fundamental to build mine.

Thanks to all my university colleagues during these tough years. Eli, Vero, Giacomo, Thomas: you are the best team mates ever. Oscar, Marco, Carlo, Riccardo and Stefano: I have missed you so much during my Master's classes. Thanks Ale for being my sister for a whole year and still standing me every time. Thank you Lorcan for reminding me I could be worthy even when I thought I could not live up to the world's expectations.

Juanki, if it wasn't for you I would probably still be at my first exam of this Master's degree. Thanks for always supporting me, I have never told you how important it has been.

Thanks to my study mates Carlo, Federico, Alessandro, Andrea, Margot, Sara and especially Ilaria for all the endless days at Tiraboschi, and to my friends Elisa, Francesca, Giorgio, Laura and Giuseppe for always being there. Thanks Matteo just for being the crazy person he is and Antonella and Vincenzo for our study-free dancing nights out.

Thanks to Emilio, Redi, Giulio, Ines and all the people I met at Liceo Secco Suardo: my students of classes 3Y, 4Y and 5Y, to whom I wish the best for their future, and all my colleagues, they have all been inspirational to me and I will remember this experience whatever my future job will be.

Finally, thanks to my father, who listened to the review of all my exam programs; my mother, who always believed in me even when I did not; Domenico and Luna, for standing me the whole time; my relatives that sent me all their love from Roma and Cristina and Adriana, who always supported me as only a family member could do.

Diego, I will never thank you enough for being my reference point. I love you.