POLITECNICO DI MILANO

School of Industrial and Information Engineering

Master of Science in Automation and Control Engineering

REALIZATION OF INTELLIGENT SYSTEMS FOR THE OBSERVATION, IDENTIFICATION AND CLASSIFICATION OF GEOSTATIONARY AND LOW ORBIT SATELLITES

Advisor:
Prof. Marco Lovera

Co-Advisors:
Prof. Roberto Furfaro
Prof. Pierluigi Di Lizia
Prof. Vishnu Reddy

Master Thesis by:
Simone Bellesia
Student ID 839322

Academic Year 2015-2016

"Here's to the crazy ones. The misfits. The rebels. The troublemakers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do".

To my family

Statement of Autorship

I declare on oath that I completed this work on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this, nor a similar work, has been published or presented to an examination committee.

Tucson, March 18, 2017

Acknowledgements

Gaining an experience in the United States is an opportunity which has always inspired me, since I was a child. These five months have been very enriching for me both from the academic and the social point of view. This experience helped me a lot to grow as a person. That is the reason why, first of all, I would like to thank Prof. Roberto Furfaro, who accepted me at the University of Arizona even without knowing me. He has been very kind and welcoming during all my stay, other than the fact that he taught me a lot of things and he guided me along all this path. According to this I would like to give thanks to Prof. Francesco Topputo, who has supported me from the beginning.

I would like to thank my advisor Prof. Marco Lovera for having made this thesis possible and for the interest in my research, and my co-advisor Prof. Pierluigi Di Lizia for his brilliant advices and for all the support. They have been very helpful even if far away. Thank you!

Moreover, I am also very thankful to my second co-advisor in Arizona, Prof. Vishnu Reddy, incredible and very affable person, to Dr. Michael Mommert, for the great hospitality during my stay in Flagstaff and for the availability, to Prof. Jeffrey Larsen and to Prof. Richard Linares for their useful suggestions.

Furthermore, I would like to thank the friends I met here in Tucson: Remi, Linda, Vishnu, Omar, Faisal, Sahand, Ethan and Saheed. They are very friendly guys and they have gladdened my stay in a wonderful way. Many thanks also to Federico, Matteo, Andrea and Barbara for these amazing two years together at Politecnico di Milano.

I would like to give special thanks to my mates here in Arizona: Giulia and Roberto. They have been the best mates I could ask for and they made me feel like at home. Love you guys!

Finally, being abroad and far away could be difficult, but it is not if you know that someone is waiting for you in the other side of the world. That is why, last but not least, I would like to thank very much my family, who has always supported me in all my choices and whom I love, and all my friends, the family I have chosen.

Ringraziamenti

Fare un'esperienza negli Stati Uniti è un'opportunità che mi ha sempre ispirato, fin da quando ero piccolo. Questi cinque mesi mi hanno arricchito molto sia sotto l'aspetto accademico che sotto quello sociale. Questa esperienza mi ha aiutato a crescere come persona. Questa è la ragione per cui, prima di tutto, voglio ringraziare il Prof. Roberto Furfaro, che mi ha accettato alla University of Arizona anche senza conoscermi. E' sempre stato molto gentile e accogliente durante tutto il soggiorno, oltre al fatto che mi ha insegnato molte cose e mi ha guidato lungo tutto il percorso. A proposito di ciò , vorrei ringraziare il Prof. Francesco Topputo, che mi ha supportato fin dall'inizio.

Voglio ringraziare il mio relatore Prof. Marco Lovera per aver reso possibile questa tesi e per l'interesse dimostrato nella mia ricerca, e il mio correlatore Prof. Pierluigi Di Lizia per i suoi brillanti consigli e per tutto il supporto. Entrambi sono stati molto utili anche se lontani. Grazie mille!

Inoltre, sono molto riconoscente al mio secondo correlatore in Arizona, Prof. Vishnu Reddy, persona incredibile e molto affabile, al Dr. Michael Mommert, per la sua grande ospitalità durante il mio soggiorno a Flagstaff e per la disponibilità , al Prof. Jeffrey Larsen e al Prof. Richard Linares per i loro utilissimi consigli.

Inoltre, vorrei ringraziare gli amici che ho incontrato qua a Tucson: Remi, Linda, Vishnu, Omar, Faisal, Sahand, Ethan e Saheed. Sono ragazzi molto simpatici e hanno allietato questi mesi in modo fantastico. Un sentito ringraziamento anche a Federico, Matteo, Andrea e Barbara per questi due anni eccezionali trascorsi insieme al Politecnico di Milano.

Vorrei riservare un grazie speciale ai miei compagni qua in Arizona: Giulia e Roberto. Sono stati i migliori compagni d'avventura che avrei potuto desiderare e mi hanno fatto sentire a casa. Vi voglio bene ragazzi!

Infine, essere all'estero e lontani da casa può essere difficile, ma non lo è se sai che c'è qualcuno che ti aspetta dall'altra parte del mondo. Per questo motivo, ultimo ma non ultimo, vorrei ringraziare la mia famiglia, che mi ha sempre supportato nelle mie scelte e che amo, e tutti i miei amici, la famiglia che mi sono scelto.

Abstract

Satellites in orbit around Earth are used in many areas and disciplines, including space science, navigation, meteorology, climate research, telecommunication and many others. They offer a unique resource for collecting scientific data, which lead to unrivalled possibilities for research and exploitation, both scientific and commercial. However, in the past decades, with the increasing of space activities, the number of space objects orbiting around Earth has dramatically increased. The reasons are that more and more countries have achieved a presence in space and that there has been a growth in the number of inactive and debris objects. In February 2009 a collision occurred between an active Iridium communication satellite and a defunct Russian satellite. It happened at an altitude that will ensure a long life for the resultant debris cloud. It was a total unexpected accidental collision that catapulted the concept of Space Situational Awareness (SSA) to the fore. SSA refers to the ability to view, understand and predict the physical location of natural and manmade objects in orbit around the Earth, with the objective of increasing knowledge and avoiding collisions. This thesis fits in the field of SSA. In particular, the main goal is to study and develop techniques in order to realize intelligent systems to observe, identify and classify geostationary and low orbit satellites. Since the subject is too wide, only three problems are examined.

The first task is the realization of an astrometry and photometry pipeline to automatically extract information related to the position and the brightness of targets from telescope images. Two solutions are presented: a simple solution built "from scratch" and a more complex one which makes use of already existing programs available online. The second part investigates how to classify the shape/control of an object starting from its light curve through machine learning algorithms (especially through a recent method called Extreme Learning Machines). Finally, the third problem is associated with sensor management via Q-learning, where the goal is to maintain knowledge over a given number of geostationary satellites using a limited number of sensing platforms.

The results are particularly satisfactory. The complex pipeline is shown to perform very good with real images depicting satellites. Then, the accuracy of the classification procedure turns out to be around 95% and the Extreme Learning Machines present a very fast learning with respect to traditional algortihms. Finally, the Q-learning applied to the sensor tasking problem with 3 satellites works properly.

Sommario

I satelliti in orbita intorno alla Terra sono utilizzati in svariate aree e discipline, tra cui la scienza spaziale, la navigazione, la meteorologia, le telecomunicazioni e tante altre. Essi offrono una risorsa unica per la raccolta di dati scientifici, che porta a opportunità impareggiabili per quanto riguarda la ricerca e possibili utilizzi sia scientifici che commerciali. Tuttavia, negli ultimi decenni, con l'aumento delle attività spaziali, il numero di oggetti in orbita intorno alla Terra è drammaticamente aumentato. Le ragioni sono che sempre più stati hanno raggiunto una presenza nello spazio e che si è verificata una crescita degli oggetti inattivi e dei detriti. Nel Febbraio 2009 è avvenuta una collisione tra il satellite attivo per la comunicazione Iridium e un satellite Russo inattivo, ad un'altitudine che garantirà una lunga sopravvivenza alla nuvola di detriti creata. Si è trattato di una collisione accidentale totalmente inaspettata che ha portato alla ribalta il tema della Space Situational Awareness (SSA). Il termine SSA si riferisce all'abilità di osservare, capire e predire la posizione fisica di oggetti naturali o creati dall'uomo in orbita intorno alla Terra, con l'obiettivo di aumentare la conoscenza ed evitare collisioni. La tesi rientra nell'ambito della SSA. In particolare, l'obiettivo principale è lo studio e lo sviluppo di tecniche per la realizzazione di sistemi intelligenti per osservare, identificare e classificare satelliti geostazionari e in orbita bassa. Dato che il tema è molto vasto, solamente tre problemi sono esaminati.

Il primo compito riguarda la realizzazione di una procedura per l'astrometria e la fotometria in modo da estrarre automaticamente informazioni riguardanti la posizione e la luminosità di alcuni oggetti da immagini prese da un telescopio. Vengono presentate due soluzioni: una semplice costruita da zero e una più complessa che utilizza programmi esistenti disponibili in rete. La seconda parte studia come classificare la forma/controllo di un oggetto spaziale a partire dalla sua curva di luce attraverso algoritmi di machine learning (specialmente attraverso un recente metodo chiamato Extreme Learning Machines). Infine, il terzo problema è associato alla gestione di sensori ottici per mezzo del Q-learning, in cui l'obiettivo è osservare un certo numero di satelliti geostazionari usando un limitato numero di sensori.

I risultati sono particolarmente soddisfacenti. La procedura complessa delineata nella prima parte si comporta molto bene con immagini reali che includono satelliti. Successivamente, l'accuratezza del procedimento di classificazione risulta essere intorno al 95% e il metodo Extreme Learning Machines presenta un apprendimento molto veloce rispetto agli algoritmi tradizionali. Infine, il Q-learning applicato al problema sensoristico con 3 satelliti funziona in maniera appropriata.

Contents

List of Figures

List of Tables

Introduction

"The danger of asteroid or comet impact is one of the best reasons for getting into space. I'm very fond of quoting my friend Larry Niven: 'The dinosaurs became extinct because they didn't have a space program. And if we become extinct because we don't have a space program, it'll serve us right!'"
(Arthur C. Clarke, Writer, Author and Inventor)

The U.S. Air Force has maintained a catalogue of Space Objects (SOs) since the dawn of the space age and the network of sensors that provides the data for this catalogue is called the Space Surveillance Network (SSN) [1]. Over the past decades, the number of SOs has dramatically increased for two main reasons: the first one is that more and more countries have achieved a presence in space, the second one is that there has been a growth in the number of inactive and debris objects.

In addition to this trend, space has become a crucial region during the years. Space-based systems are indispensable to many services critical to economy and government functions, including those related to security. This dependency will only increase in the future. Any shutdown or loss of services from these systems would seriously affect an enormous range of commercial and civil activities, including commercial land, air and sea travel, maritime navigation, telecommunications, information technology and networks, broadcasting, climate monitoring, and so on [2]. It is possible to think about a catastrophic but possible situation: if space debris or unknown objects, for some reasons, hit some GPS satellites[1], significant problems would be caused for the security and the normal operation of the land transports system. This is only one of the many examples which can be listed.

What has been said so far is a great motive for the need to support and invest in the space surveillance and this is why a large number of space agencies (especially National Aeronautics

[1] A GPS satellite is a satellite used by the NAVSTAR Global Positioning System (GPS). The first satellite in the system, Navstar 1, was launched February 22, 1978. The GPS satellite constellation is operated by the 50th Space Wing of the U.S. Air Force. The GPS satellites circle the Earth at an altitude of about 20,000 km and complete two full orbits every day [3].

and Space Administration (NASA) and European Space Agency (ESA)) is devoting a lot of time and effort to it. Furthermore, the growth in the number of SOs is stressing the capability of the SSN creating a need for new approaches for improving Space Situational Awareness (SSA). For example, SSA has become a key mission area for the U.S. Air Force which is tasked with collecting tracking data on over 22000 SOs, 1100 of which are active, currently being tracked [4].

The SSA procedure includes:

- CATALOG MAINTENANCE. Obviously, there exists a catalogue that contains all the information about the known objects. This list tracks the position and some orbital knowledge of each object, but it must be updated. Indeed, the models used to propagate the state of a body are becoming more and more precise, but they cannot take into account everything. There will be, then, some unmodeled dynamics that will make the uncertainty on the state grow with time and for this reason long term predictions are usually not useful. The principal task of this part is, so, keeping the catalogue updated.
- CLASSIFICATION. Due to the large number of SOs and the limited number of sensors available to track them, it is difficult to maintain persistent surveillance, and, therefore, there is inherent uncertainty and latency in the knowledge of the SO population. Although the amount of light collected from these objects is small, information can still be extracted from photometric data which can be used to determine shapes and other properties. Light curve data are the time-varying sensor wavelength–dependent apparent magnitude of energy (e.g., photons) scattered (reflected) off of an object along the line-of-sight to an observer. For example, based on the photometric data got from the object, it is possible to understand its size, attitude, shape and spin control in order to improve knowledge over catalogued bodies or to begin to collect knowledge over unknown ones.

These are, clearly, good reasons to invest money in the field and to study how to improve the actual situation with reasonable and pioneering solutions.

To support the cause and to bring a contribution, the University of Arizona is, as this introduction is written, currently building, under the leadership of Prof. Vishnu Reddy and with the help of the students, a telescope which will be operative in the coming years. The idea on which this work is based is to study and develop techniques in order to realize intelligent systems to observe, identify and classify space objects. In particular, this thesis is focused only on geostationary and low orbit satellites.

Technically, objects in Low-Earth-Orbit (LEO) are at an altitude of between 160 to 2000 km (99 to 1200 mi) above the Earth's surface. Any object below this altitude will suffer from orbital decay and will rapidly descend into the atmosphere, either burning up or crashing on the surface. Objects at this altitude also have an orbital period (i.e., the time it will take them to orbit the Earth once) of between 88 and 127 minutes. The vast majority of missions to space over the years, be they crewed or uncrewed, have been to LEO. It is here, indeed, that the Earth's vast array of communications, navigation and military satellites reside. And it is here, for example, that the International Space Station (ISS) conducts its operations, which is also where the majority of crewed missions today go [5].

A geostationary satellite, instead, is an Earth-orbiting satellite placed at an altitude of approximately 35800 km (22300 mi) directly over the equator, that revolves in the same direction the Earth rotates (west to east). An object in such an orbit has an orbital period equal to the Earth's rotational period (one sidereal day) and thus appears motionless, at a fixed position in the sky, to ground observers. Communications satellites and weather satellites are often placed in geostationary orbits, so that the satellite antennas (located on Earth) that communicate with them do not have to rotate to track them, but can be pointed permanently at the position in the sky where the satellites are located.

Everything that is presented in this work is designed for the telescope under development at the University of Arizona, but obviously the discussion is totally versatile and can be applied to every kind of instrument able to observe the sky. But, as the subject is so extended, the amount of work and possible implementations is considerable. That is the reason why this work considers only three contents: the realization of an astrometry and photometry pipeline, the classification procedure and the sensor tasking problem.

The project is particularly important since intelligent systems for this purpose have never been made. So far, indeed, telescopes have only been used to take images of the sky possibly with the aid of some programs which, however, are not able to identify eventual targets or to recognize unknown moving objects. The astrometry and photometry solution of each image is nowadays made manually using some software (e.g., Astrometrica) which cannot be automatized. Hence the need to build some intelligent systems which can solve this job automatically. The application of machine learning algorithms, finally, allows to extract information directly from the data without any user intervention. Moreover, they are intelligent techniques since, after the training, they make the system able to guarantee a satisfactory operation in every possible situation.

The thesis is organized as follows.

The introduction includes the motivation and the organization of the thesis.

Chapter 1 is intended to be a kind of manual for the realization of an astrometry and photometry pipeline. In the first part, a personal simple solution "from scratch" is presented and explained. Then, a more complex solution developed by Dr. Mommert, post-doctoral researcher at the Department of Physics and Astronomy at Northern Arizona University in Flagstaff, is shown. Since it is designed for asteroids, a personal integration is realized in order to work also with satellites.

The second subject of the thesis, the classification procedure, is elucidated in Chapter 2. The problem, that is trying to classify objects from their light curve, is faced with machine learning algorithms and, especially, with a new and interesting one, the Extreme Learning Machines (ELM).

Chapter 3 is about the explanation of a sensor tasking and management issue for optical space objects tracking. Q-learning, that is an example of reinforcement learning technique, is used for the purpose.

The last Chapter, 4, contains the conclusions of the thesis and provides an insight into future works.

Chapter 1
Pipeline

"The contemplation of celestial things will make a man both speak and think more sublimely and magnificently when he descends to human affairs."
(Marcus Tullius Cicero, Philosopher, Politician and Orator)

This chapter is focused on the realization of an astrometry and photometry pipeline for the observation of geostationary and low orbit satellites through a telescope. The sections explains in an exhaustive way the main concepts in the interest of creating a kind of manual on the argument: at the beginning some basics of astronomy in order to allow the reader to understand as much as possible, then a simple personal solution "from scratch" and at the end a working and well designed solution which makes use of existing free online programs.

1.1 Problem setting and fundamentals

In this section the problem to be faced is set and all the fundamentals that are necessary to understand its definition and the development of the solution are described. The theory explained in this chapter should be sufficient to comprehend also the arguments of the other chapters.

1.1.1 Definition of the problem

As the title of this chapter, the main goal is to develop an astrometry and photometry pipeline that can be used for the images taken by a telescope. It is then necessary to explain better the meaning of these three terms and why they are really important for the realization of an intelligent system to observe, identify and classify geostationary and low orbit satellites:

- ASTROMETRY. Astrometry is the branch of astronomy that involves precise measurements of the positions and movements of stars and other celestial bodies.

- PHOTOMETRY. Photometry is the science of measurement of light, in terms of its perceived brightness to the human eye.
- PIPELINE. The term pipeline generally refers to the process of data reduction. More specifically, it usually refers to a well-established, largely (or entirely) automated series of data reduction processes associated with a particular instrument or type of data.

Therefore, in this chapter a solution for this kind of issue is explained: given an image, or a set of images, taken by a telescope or by an instrument that is able to observe and take a picture of the sky, that includes one or more space objects, obtain their angular coordinates and collect information about their brightness and light curve[2]. In particular, as the title of the thesis, the whole discussion concerns geostationary and low orbit satellites.

1.1.2 Fundamentals

Once the problem is set, the theoretical fundamentals which this work is based on are illustrated.

1.1.2.1 Keplerian elements

Seven parameters are necessary to define an orbit of a body (so also of a satellite) about a spherically symmetric body [6]. One is the gravitational parameter, generally denoted by the symbol μ. There are many ways of representing the other six elements. The two most popular are position and velocity (r and v) vectors and Keplerian elements. The first ones, also called Cartesian elements, are well known, the second ones are represented in Figure 1.1 and are defined as follows: Ω is the longitude or right ascension of the ascending node[3], or the angle from the reference direction to the line where the orbit plane intersects the xy-plane; ω is the argument of periapsis and is the angle in the orbit plane between the ascending node line and periapsis (where the orbit is closest to the center of the central body); ν is the true anomaly and is the angle between periapsis and the object; i is the inclination and is the angle between the xy-plane and the orbit plane; the size of the orbit is determined by the semi major axis a, which is the

[2] A light curve is a graph of light intensity of a celestial object as a function of time (generally taken in a particular frequency or spectral bandwidth).

[3] The ascending node is the point in an orbit where a body traveling from south to north crosses a reference plane, such as the plane of the ecliptic (in the case of a Solar System object) or the celestial equator. The opposite point in the orbit, where the body moves from north to south across the reference plane, is the descending node [7].

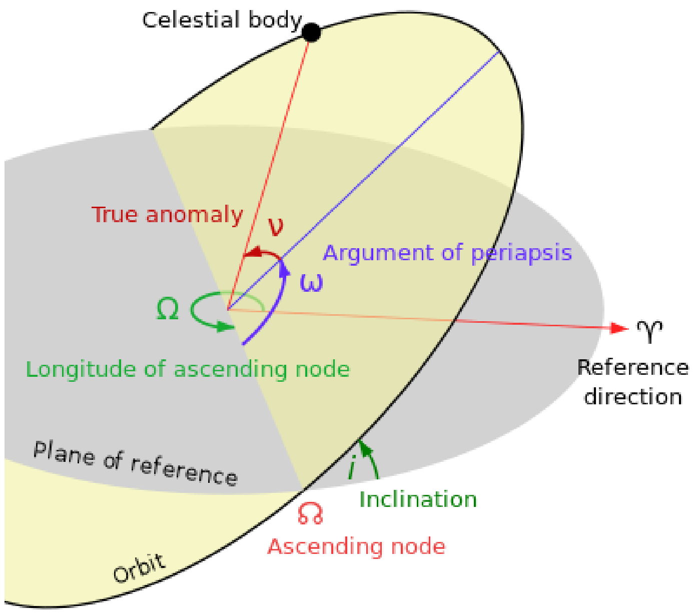


Figure 1.1: Illustration of the Keplerian elements

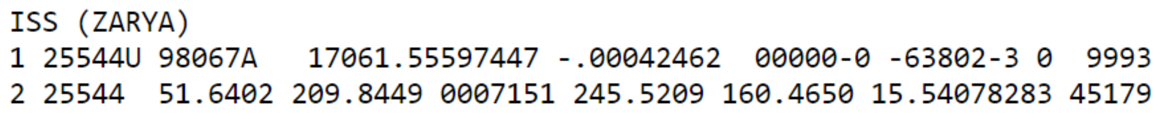
average of the periapsis radius and apoapsis radius; and the final parameter is e, the eccentricity, which determines the shape of the orbit. The mean anomaly M may be used in the element set instead of ν. To summarize, the Cartesian and the Keplerian elements are:

$$x = \begin{bmatrix} \mu \\ r_x \\ r_y \\ r_z \\ v_x \\ v_y \\ v_z \end{bmatrix} \qquad \text{CARTESIAN ELEMENTS}$$

$$x = \begin{bmatrix} \mu \\ a \\ i \\ \Omega \\ \omega \\ e \\ M \end{bmatrix} \qquad \text{KEPLERIAN ELEMENTS}$$

1.1.2.2 TLE

As it was said in the previous paragraph, the Keplerian elements are a set of numbers which allows satellite tracking programs to calculate a orbital body's position in space. The Keplerian elements of this kind of objects for a given point in time, the epoch, come mainly in two formats, either the NASA two-line elements (TLE, also called NORAD TLE) or the AMSAT verbose format elements. In this work, only TLE are used. Once these elements are known for a specific time, the satellite's position in space can be predicted using complex mathematical calculations.



```
ISS (ZARYA)
1 25544U 98067A   17061.55597447 -.00042462  00000-0 -63802-3 0  9993
2 25544  51.6402 209.8449 0007151 245.5209 160.4650 15.54078283 45179
```

Figure 1.2: Example of TLE for the International Space Station, 02/03/17

Naturally, tracking programs need accurate and recent data in order to generate accurate predictions. Space-Track.org is the primary distributor of U.S. Department of Defense (DoD) two-line orbital elements and related data, replacing NASA Goddard Space Flight Center's Orbital Information Group (OIG) which permanently ceased operations on 2005 March 31.

For this project, the TLE are taken from the open website celestrak.org, which relies on the Space-Track.org website and in which, depending on the objects, the TLE are updated once or twice a day. Indeed, TLE have one problem: they are given for a specific time and the accuracy of the position prediction degrades as time goes by. The suggestion is not to use TLE older than two weeks for low orbiting satellites and than four-five weeks for high orbiting ones.

As shown in Figure 1.2, the TLE can include a title line preceding the element data. However, the title is not required, as each data line includes a unique object identifier code.

In the following, an exhaustive explanation of the TLE format is given [8].

```
1 AAAAAU YYLLLPPP  BBBBB.BBBBBBBB  .CCCCCCCC  DDDDD-D  EEEEE-E F GGGGZ
2 AAAAA  HHH.HHHH III.IIII JJJJJJJ KKK.KKKK MMM.MMMM NN.NNNNNNNNRRRRRZ
```

- [1] - Line #1 label
- [2] - Line #2 label
- [AAAAA] - Catalogue number assigned sequentially (5-digit integer from 1 to 99999)
- [U] - Security classification (U = Unclassified)
- [YYLLLPPP] - International designator (YY = 2-digit launch year; LLL = 3-digit sequential launch of the year; PPP = up to 3 letter sequential piece ID for that launch)
- [BBBBB.BBBBBBBB] - Epoch time -- 2-digit year, followed by 3-digit sequential day of the year, followed by the time represented as the fractional portion of one day
- [.CCCCCCCC] - $\ddot{n}/2$ drag parameter (rev/day^2) -- one half the first time derivative of the mean motion. This drag term is used by the SGP orbit propagator.

- [DDDDD-D] - $\ddot{n}/6$ drag parameter (rev/day^3) -- one sixth the second time derivative of the mean motion. The ”-D” is the tens exponent (10-D). This drag term is used by the SGP orbit propagator.
- [EEEEE-E] - Bstar drag parameter (1/Earth radii) -- Pseudo Ballistic Coefficient. The ”-E” is the tens exponent (10-E). This drag term is used by the SGP4 orbit propagator.
- [F] - Ephemeris type -- 1-digit integer (zero value uses SGP or SGP4 as provided in the project Spacetrack report).
- [GGGG] - Element set number assigned sequentially (up to a 4-digit integer from 1 to 9999). This number recycles back to ”1” on the update following element set number ”9999”.
- [HHH.HHHH] - Orbital inclination (from 0 to 180 degrees).
- [III.IIII] - Right ascension of the ascending node (from 0 to 360 degrees).
- [JJJJJJJ] - Orbital eccentricity -- there is an implied leading decimal point (between 0.0 and 1.0).
- [KKK.KKKK] - Argument of periapsis (from 0 to 360 degrees).
- [MMM.MMMM] - Mean anomaly (from 0 to 360 degrees).
- [NN.NNNNNNNN] - Mean motion (revolutions per day).
- [RRRRR] - Revolution number (up to a 5-digit integer from 1 to 99999). This number recycles following revolution number 99999.
- [Z] - Check sum (1-digit integer). Both lines have a check sum that is computed from the sum of all integer characters on that line plus a ”1” for each negative (-) sign on that line. The check sum is the modulo-10 (or ones digit) of the sum of the digits and negative signs.

1.1.2.3 Reference systems

A reference system is a sytem with respect to which a certain body or phenomenon is measured or some measurements are specified. Obviously, there are plenty of them, each one with its rigorous definition. The reference systems used in this paper are explained in the following.

- EARTH-CENTERED, EARTH-FIXED (ECEF) COORDINATE SYSTEM
  ECEF is a geographic coordinate system and Cartesian coordinate system, and is sometimes known as a “conventional terrestrial” system. It represents positions as an X, Y and Z coordinate with the origin at the Earth’s center of mass. The x-axis extends through the intersection of the prime meridian (0 $^\circ$ longitude) and the equator (0 $^\circ$ latitude), the z-axis extends through the north pole (i.e. coincident with the Earth spin axis), the y-axis completes the right-handed coordinate system, passing through the equator and 90 $^\circ$ longitude, as shown in Figure 1.3 [9].

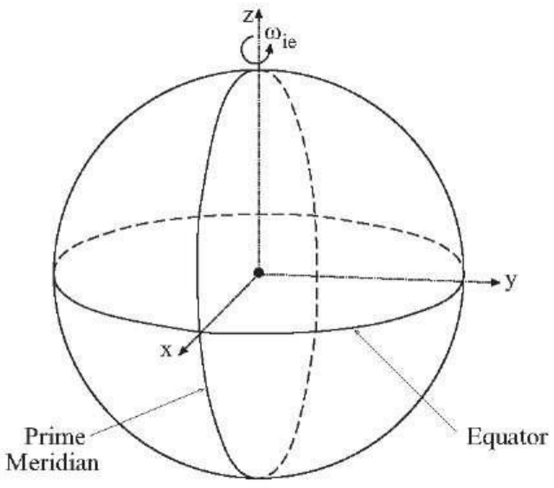


Figure 1.3: Illustration of the ECEF coordinate system

- EARTH-CENTERED INERTIAL (ECI) COORDINATE SYSTEM
  ECI coordinate frames have their origins at the center of mass of the Earth. ECI frames are called inertial in contrast to ECEF frames which rotate in inertial space in order to remain fixed with respect to the surface of the Earth. The most known are:

  J2000. One commonly used ECI frame is defined with the Earth's Mean Equator and Equinox[4] at 12:00 Terrestrial Time on January 1, 2000. The x-axis is aligned with the mean equinox. The z-axis is aligned with the Earth's spin axis or celestial North Pole. The y-axis is rotated by 90 ° East about the celestial equator.

  TEME. The ECI frame used for TLE is sometimes called true equator, mean equinox (TEME) although it does not use the conventional mean equinox.

[4] The fundamental plane of an astronomical reference system has conventionally been the extension of the Earth's equatorial plane, at some date, to infinity. The equinox is the point at which the Sun, in its yearly circuit of the celestial sphere, crosses the equatorial plane moving from south to north. The Sun's apparent yearly motion lies in the ecliptic, the plane of Earth's orbit. The equinox, therefore, is a direction in space along the nodal line defined by the intersection of the ecliptic and equatorial planes; equivalently, on the celestial sphere, the equinox is at one of the two intersections of the great circles representing these planes. Because both of these planes are moving, the coordinate systems that they define must have a date associated with them; such a reference system must be therefore specified as "the equator and equinox of [some date]" [10].

- SPHERICAL EQUATORIAL COORDINATE SYSTEM
  The equatorial coordinate system is a celestial coordinate system widely used to specify the positions of celestial objects. The spherical coordinates are a different definition of the ECI coordinates. Indeed, the reference system is the same, but the ECI coordinates are rectangular (defined as X, Y and Z), instead the ones described here are spherical. Body's spherical coordinates are often expressed as a pair, right ascension and declination. The declination (dec) of an object is its angular distance north or south of the equatorial plane. The right ascension (RA), instead, is its angular distance measured eastward along the equator from the equinox (Figure 1.4).
  For stars these coordinates do not change with the observer position, but this happens for satellites or other relatively close bodies. This can be misleading and not intuitive, but the explanation is that RA and dec indicate the positions on the background sky: stars can be described by fixed spherical coordinates because they are infinitely far and do not move relatively to one another when the position of the observer changes. Satellites are much closer, so if the observer moves they will move compared to the star background.

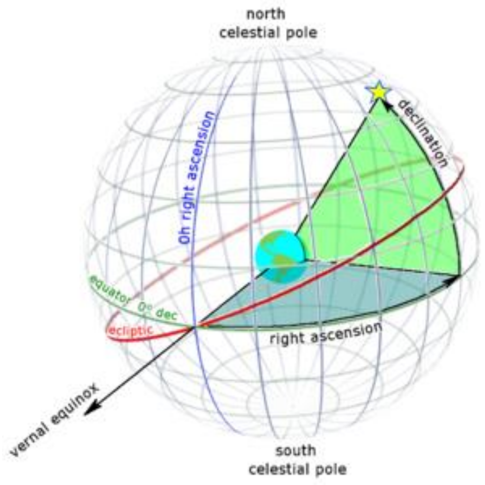


Figure 1.4: Illustration of right ascension and declination

1.1.2.4 Telescope images

Flexible Image Transport System (FITS) is an open standard (based on a specification, the "FITS Reference", publicly available) defining a digital file format useful for storage, transmission and processing of scientific and other images. FITS format is currently the most common used format in astronomy and there are many software applications for creating and viewing files in FITS format and for their conversion from and to other image formats [11]. The success of this kind of format is due to the fact that it is designed specifically for scientific data: each FITS file consists of a header that carries keyword/value pairs and of an image data block. Furthermore, the information relating to the pixels of the images are stored in a very simple way, without any compression; due to its simplicity it is possible to read a file in FITS format using basic software tools.

All the images made by a telescope used in this paper are in FITS format and the header reports important features like:

- the number of bits used to represent each pixel of the image;
- the time and Julian date[5] of the observation;
- the exposure time in seconds, which is the length of time when the film or digital sensor inside the camera of the telescope is exposed to light;
- the height and width in pixels of the image;
- the name, latitude and longitude of the observer;
- the name of the target (if present);
- the right ascension and declination of the center of the image;
- the focal length of the telescope, aperture diameter, pixel scale (defined as arcsec/pixel) and field of view (defined in degrees).

1.2 Astrometry and photometry pipeline

In this section the outline to realize an astrometry and photometry pipeline is drafted and all its passages are deeply described, with also examples regarding real images. A final solution, realized for geostationary and low orbit satellites, is presented.

1.2.1 Outline

The steps to be solved in order to reach the goal are:

1. OPEN IMAGE
2. READ THE HEADER
3. FETCH STAR CATALOGUE(S) (online or local)
4. PLATE SOLVE THE IMAGE (plate solve, in astronomic language, means to match the stars in the image to a given star catalogue)
5. CHECK TLE PREDICTIONS FOR OBJECTS IN THE FIELD OF VIEW
6. PLOT PREDICTED POSITIONS ON IMAGE
7. IDENTIFY MOVING TARGETS

[5] The Julian date is the number of elapsed days (and fractions of day) since the beginning of a cycle of 7,980 years, invented by Joseph Scaliger in 1583. The purpose of the system is to make it easy to compute difference between one calendar date and another calendar date. The starting point for the first Julian cycle began on January 1, 4713 B.C. and will end on January 22, 3268. The following day will begin the first day of the second Julian date period.

8. ASTROMETRY AND PHOTOMETRY (obtain real angular coordinates and magnitude of the objects)

1.2.2 Getting started

Reading the outline defined at the previous paragraph, it is clear that the first two points are really straightforward and are well explained in paragraph 1.1.2.4. So, from now on, the successive problems are faced. First of all, how to manage and access star catalogues.

1.2.2.1 Star catalogues

A star catalogue is an astronomical catalogue that lists stars. There is an incredible large number of them which have been produced for different purposes over the years. Star catalogues were compiled by many different ancient people, including the Babylonians, Greeks, Chinese, Persians and Arabs. Most modern catalogues are available in electronic format and can be freely downloaded from space agencies data center.

There are two main ways to access star catalogues: using programs that query online servers which contain a large number of catalogues or using a local copy of the catalogue and reading it.

The first option would obviously lead to more accurate results and it would be easier to use since there is the possibility to make the program search inside some very equipped catalogues. There are some open source solutions: “scat” is a utility in the WCSTools package (made by the Smithsonian Astrophysical Observatory) for finding from a specified catalogue objects which are located in a specified region of the sky [12] and “VizieR” is a tool created by the Observatoire astronomique de Strasbourg which provides access to the most complete library of published astronomical catalogues and data tables available online [13]. But, contrary to what it seems, handling these tools in an automated way, as the goal, is not easy at all and secondly this procedure is not instructive since it is completely done with the user in the dark. For this reason, for the sake of simplicity and clarity, at the beginning the second option is followed.

The disadvantages of the second option is that it is necessary a local copy of the catalogues (which usually are very large files) and managing them is not very easy. Three different catalogues are taken into consideration:

- HIPPARCOS CATALOGUE. The Hipparcos astrometric catalogue contains a lot of information (including positions and magnitude) for 118218 stars. It is one of the final products of ESA’s Hipparcos satellite mission.
- TYCHO-2 CATALOGUE. The Tycho-2 astrometric and photometric reference catalogue contains information for 2539913 of the brightest stars in the sky. This list is a more complete and precise version of Tycho catalogue, which is one of the primary products of the ESA’s astrometric satellite, Hipparcos, which collected data for fours years, from November 1989 to March 1993.

- USNO-B1.0 CATALOGUE. The USNO-B1.0 is a catalogue of 1042618261 objects sorted by right ascension in zone catalogues of one tenth of a degree of declination each. The data were obtained from various sky surveys during the last 50 years by the Precision Measuring Machine (PMM) at the US Naval Observatory in Flagstaff, Arizona.

The first two catalogues can be found as .mat files (the second file is a courtesy of Eran Ofek, Weizmann Institute of Science, Israel), so files accessible from MATLAB, the software used for this work. The Tycho-2 catalogue, however, is much more complete than the Hipparcos one. The third catalogue can be downloaded online but it's a very large file (almost 70 GB) and in this case it is possible to have it by courtesy of Prof. Larsen, United States Naval Academy, Annapolis, Maryland. However, it is a binary file and this makes trying to access it from MATLAB difficult. These are the reasons why for the purpose of this work the Tycho-2 catalogue is used.

Star catalogue	Number of objects	MATLAB format
HIPPARCOS	118218	✓
TYCHO-2	2539913	✓
USNO-B1.0	1042618261	✗

Table 1.1: Summary of the star catalogues considered

1.2.2.2 Plate solving

Plate solving is a procedure of image analysis that detects the stars and tries to identify them using catalogues of known stars. If the analysis is successful it is possible to calculate with extremely high precision the relation between the position of the body in the sky and its position in pixels in the image.

The first step is then to recognize the stars displayed in the image and the only way to do it is to find all the bright points in the picture. Certainly, not all the bright points are stars (they could be space objects or other things) but at least it is possible to guarantee that all the stars are detected. Explained in this way the problem seems easy, but questions arise: what does bright point mean? How to recognize at the same time small stars and big ones? How to distinguish between two small near stars and a single large star? All these questions are important and are faced in this paragraph.

Starting from the first trouble, what does bright point mean? Initially, it is necessary to define the term with respect to which the brightness is evaluated. The easiest and maybe most intelligent approach to do it is to consider the brightest point of the image. As it was stated in paragraph 1.1.2.4, each pixel of a FITS picture is associated with a number (using a certain defined number of bits) that points out its brilliance. So it is possible to fix a percentage (or threshold) and consider as luminous pixels the ones whose brightness is higher than the set percentage of the most brilliant pixel. Just to give an example, if the image uses 8 bits to define brightness and the most brilliant pixel has a value of 255, with a threshold of 0.9 the pixels with an associated value greater than $0.9 \cdot 255 = 229.5$ will be recognized as luminous.

The definition of the threshold is a crucial point: indeed, trotting out the other problems listed at the beginning, if a small threshold is chosen then it is probable that a single star will be split in several smaller identified stars and if, conversely, a large threshold is chosen then it is probable that some stars will be missed. The examples reported in Figures 1.6-1.7 (with respectively a threshold of 0.5 and 0.95) applied on the image shown in Figure 1.5 should clarify potential doubts.

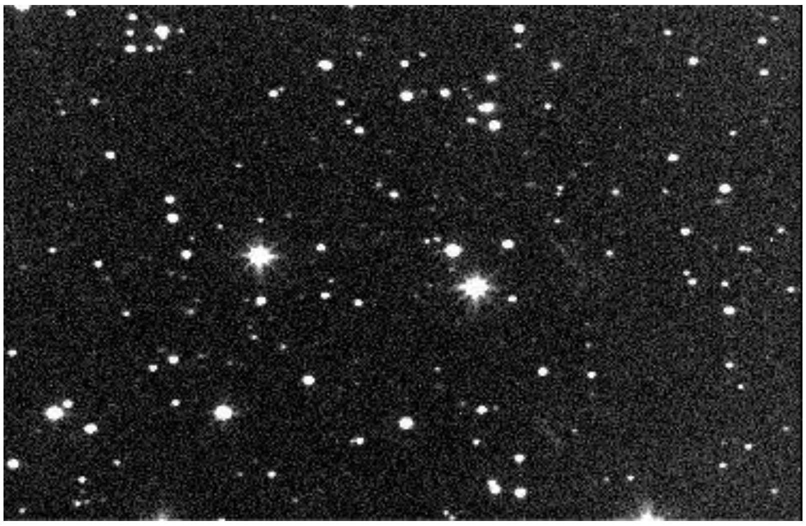


Figure 1.5: Example of sky image taken using a telescope

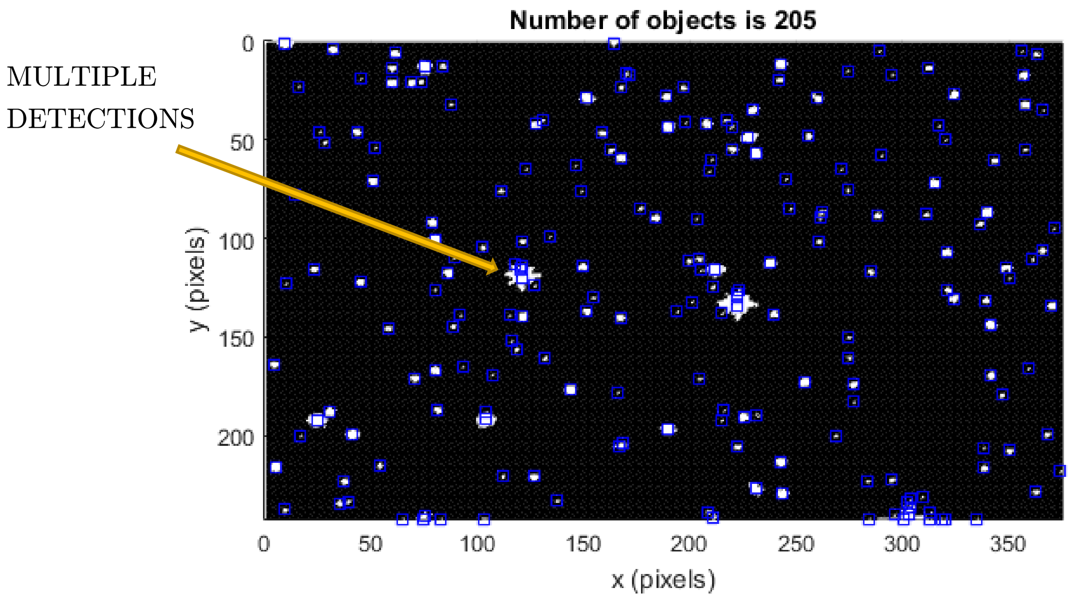


Figure 1.6: Bright points identified in Figure 1.5 with threshold equal to 0.5

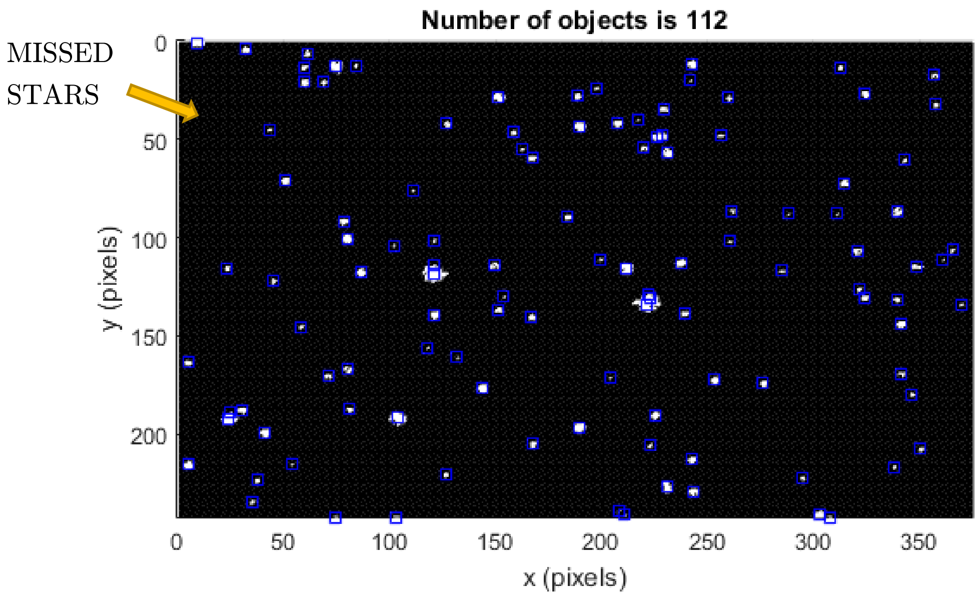


Figure 1.7: Bright points identified in Figure 1.5 with threshold equal to 0.95

In the examples shown, it happens what was expected.

The employed algorithm is the following: first of all, the pixels whose brightness is smaller that the threshold value (equal to the threshold times the brightness of the most brilliant pixel) are considered as full dark (with a value of zero). Then, regions are computed: starting from the upper left angle of the image and moving toward right, each point with a value different from zero is analyzed. Following the strategy described in Figure 1.8, the program controls if there is an existing region nearby. If so, the point belongs to that region, which is propagated to the near pixels (not all, only to the one at the same row on the right and the three bottom ones). Once the algorithm finishes, the image is divided in regions and the center of each region is a bright point to be displayed.

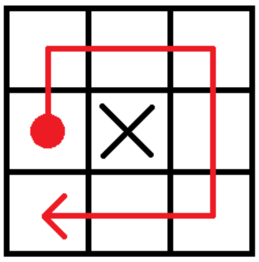


Figure 1.8: Illustration of the strategy used to divide the image into regions

The explained algorithm works pretty well, even if there is space for improvements.

At this point, it is required to match the bright points found before with stars, in order to "be oriented" in the sky. Here the star catalogues come into play. There are two methods to do it: use the position of the identified points in the image and compare it with star catalogues, or just project the star lists on the picture (with a certain transformation) and maximize the number of matches.

The first option is obviously more reliable but, at the same time, is more demanding from the implementation and computational point of view. The system of reasoning is the following: take each point found with the previous algorithm and convert the corresponding pixel location into a unit vector, using the pixel scale which is known thanks to the header file. Then, compute the dot product of each couple of vectors, which means to calculate the angle between every two points in the image. After that, select from the star catalogue only the stars that are in the field of view (maybe a bit larger in order to compensate for possible errors) and compute the angle between them as well. At the end, compare the two groups of angles and choose the configuration which discloses the greater number of pairings.

As it was said, this is quite difficult to implement and it could lead to mistakes, especially with a narrow field of view since the angles between the stars are small and very close to each other.

The second option, instead, is simpler and does not involve any of the procedures explained above, but that does not mean it has to be rejected. Actually, due to its simplicity, this strategy is chosen here to make the first simple attempt to realize an astrometry and photometry pipeline. The outline is:

1. Project the star catalogue restricted to the field of view into the picture through a coordinates transformation (from right ascension and declination, the format of the catalogue, to positions in pixels).
2. Consider testing offsets between the two lists.
3. Test quality of match.
4. Select the offset which guarantees the highest number of matches.

The chosen coordinates transformation is a simple linear model. Given the position in pixels and RA/dec of the central point and given the pixel scale (both these data can be read in the header file), the model is:

$$\begin{cases} RA = scale_{RA}\left(x - x_{ref}\right) + RA_{ref} \\ dec = scale_{dec}\left(y - y_{ref}\right) + dec_{ref} \end{cases} \tag{1.1}$$

where $scale_{RA}$ is the pixel scale along the x-axis and $scale_{dec}$ is the pixel scale along the y-axis. So, solving the equations expressed in (1.1) to calculate the positions (x, y) in the image, it is possible to obtain:

$$\begin{cases} x = \frac{RA - RA_{ref}}{scale_{RA}} + x_{ref} \\ y = \frac{dec - dec_{ref}}{scale_{dec}} + y_{ref} \end{cases} \tag{1.2}$$

In order to see how it works, it can be tested with the image shown in Figure 1.9.

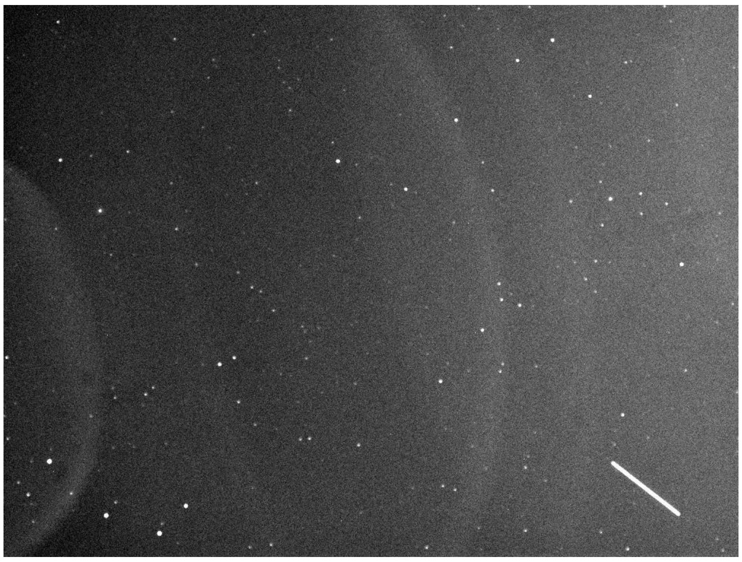


Figure 1.9: Example of sky image taken using a telescope

The picture shown in Figure 1.9, taken on June 16, 2016, displays as a line the trajectory of the OA-6 Cygnus[6] spacecraft. Indeed there are two ways to observe a target in the sky: keeping the telescope fixed on the body or keeping it fixed on the background. In the first case the stars will be represented as lines, in the second case the target will. Therefore, this is an example of the second case. The information on the picture is:

- Size: 965 pixels x 725 pixels
- Pixel width = 4.5 μm, pixel height = 4.5 μm
- Focal length = 96.2 mm
- Rotation = 179.98 °
- Pixel scale = 19.29" x 19.29" [7]

[6] Cygnus CRS OA-6, also known as Orbital Sciences CRS Flight 6, is the sixth flight of the Orbital ATK unmanned resupply spacecraft Cygnus and its fifth flight to the International Space Station under the Commercial Resupply Services contract with NASA. On March 23, 2016 Cygnus CRS OA-6 was successfully launched by the Atlas V into Low Earth orbit [14].

[7] ' defines arcminutes and " defines arcseconds, units of angular measurements. 1 ° =60'=3600".

- Field of view = 310.3' x 233.1'
- Central point: x=457.42, y=332.12, RA=216.088°, dec=-17.111°

As an additional notice, it is necessary to watch out for a possible rotation of the field of view, as in this situation. It happens because, as the Earth rotates, celestial objects will appear to move in an arc around the celestial pole. Then, if a telescope tracks a celestial object, it will remain centered in the eyepiece, but will rotate with respect to time: this is called field rotation.

In order to remove this phenomenon, once the coordinates transformation is done, a rotation around the center of the image has to be added. This can be fulfilled using a rotation matrix.

Therefore, the final position of each star in the picture is:

$$\begin{bmatrix} x_{final} \\ y_{final} \end{bmatrix} = R\left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix}\right) + \begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix} \tag{1.3}$$

where x, y are the values obtained with equation (1.2) and R is the rotation matrix, computed as:

$$R = \begin{bmatrix} cos\vartheta & -sin\vartheta \\ sin\vartheta & cos\vartheta \end{bmatrix} \tag{1.4}$$

with ϑ equal to the rotation angle.

Applying what has been said so far to Figure 1.9, the result is shown in Figure 1.10, where the blue circles are the detections in the image and the red crosses are the projected stars.

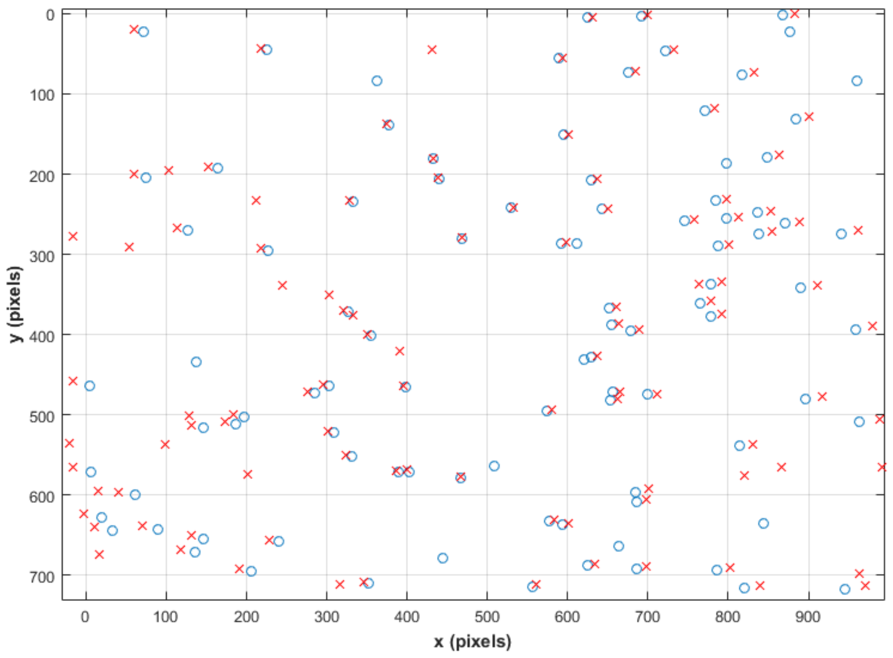


Figure 1.10: Result of the matching process on Figure 1.9

The result is really remarkable even without taking into account the offset, probably because the information are surprisingly precise. The only issue is an horizontal displacement that increases with the distance from the center of the image, probably due to the distorsion of the camera and which could be fixed quite easily. Anyway, as illustrated in the previous outline, in most of the cases it is required to introduce an offset in order to match the crosses and the circles as much as possible.

But to be fair, the result shown in Figure 1.10 is possible because the picture has a relatively large field of view (more than 5 degrees). Trying to do the same with the image exhibited in Figure 1.11 (with a small field of view, 62.7' x 50.2'), different outcomes are achieved (Figure 1.12).

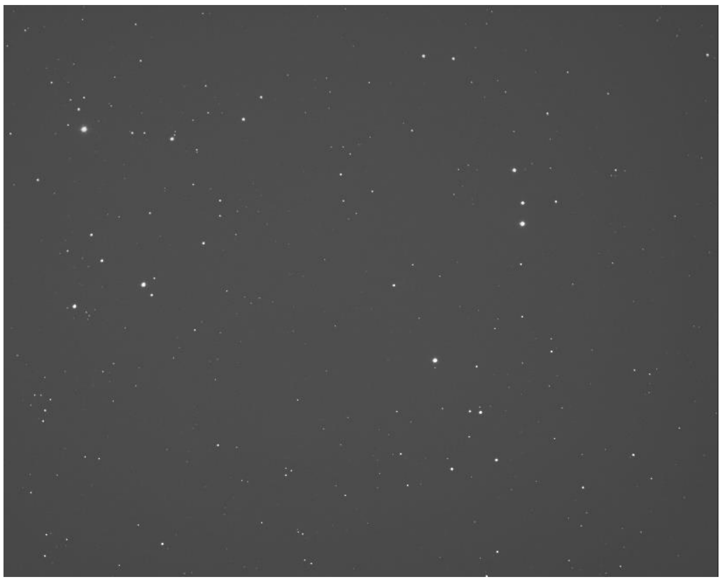


Figure 1.11: Example of sky image with a small field of view

The reason of this apparent failure shall not be ascribed to wrong equations or other theoretical problems because, as it can be seen in Figure 1.12, the few matches are almost perfect. The blame must be put on the scarcity of the used catalogue. Indeed, when the field of view is small, the density of the stars in the catalogue is not high enough to guarantee a sufficient number of references. Therefore, this reveals the need for a more complete catalogue, argument that is beyond the requirements for this first simple pipeline attempt.

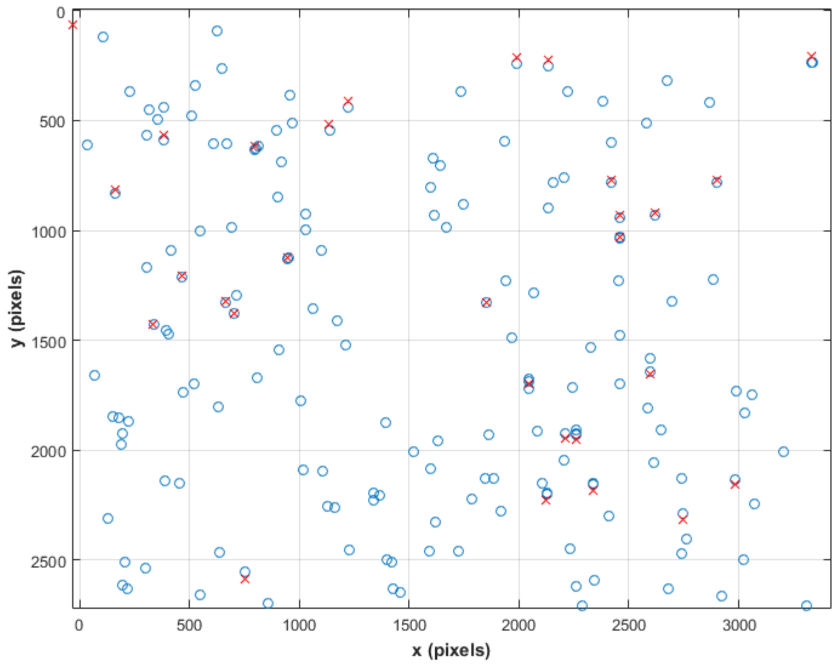


Figure 1.12: Result of the matching process on Figure 1.11: in this case it is not satisfactory

1.2.2.3 TLE propagation

One of the last steps to do is to check TLE predictions for the object (or objects) tracked in the field of view and to plot the predicted position on the image. The idea is to take TLE from Celestrak website (https://celestrak.com) and to propagate them until the moment in which the picture was taken by the telescope. It's useful to remember that, for geostationary or low orbit satellites, TLE older than 15-20 days with respect to the epoch of the measurement are not sufficiently accurate, so in order to have reasonable predictions it is required to use TLE within at maximum 10-15 days.

In order to do this, a MATLAB toolbox from Princeton Satellite Systems is used. The name of the toolbox is "Spacecraft Control Toolbox" (SCT, license available by Prof. Furfaro).

The mentioned toolbox covers an enormous amount of possible applications. For the scope of this work, two functions are employed:

- [x, n, t] = ConvertNORAD(a)

  This routine converts NORAD TLE string (the data format explained in paragraph 1.1.2.2) into a data structure.

```
------
Inputs
------
a              (:,1)    Data, file name or string
```

```
-------
Outputs
-------
x            (n)  Data structure
 .satelliteNumber  Object ID
 .launchYear       Year (part of international designator)
 .launchNumber     Launch number
 .launchpiece      Launch piece
 .epochYear        Two/one digit epoch year
 .epochDay         Epoch day number
 .n0Dot            1st Derivative of the Mean Motion (rad/min^2)
 .n0DDot           2nd Derivative of the Mean Motion
 .bStar            Adjusted ballistic coefficient
 .i0               Inclination (rad)
 .f0               Right Ascension of Ascending Node (rad)
 .e0               Eccentricity
 .w0               Argument of Perigee (rad)
 .M0               Mean Anomaly (rad)
 .n0               Mean Motion (rad/min)
The following fields are computed from the element data.
 .ballisticCoeff   Ballistic coefficient (m2/kg)
 .julianDate       Epoch Julian date
 .fullLaunchYear   Four digit launch year
 .sma              Semi-major axis (km)

n          (1,1)   Number of element sets

t          (3,n)   Time information if present on second line
```

- [rV, x] = NORAD(dStart, dEnd, nPts, model, pType, file, number)
  This routine propagates the NORAD TLE from dStart to dEnd using the given model (this paper makes use of SGP4[8]) and gives as output the positions and velocities of the satellite along the path (the number of steps is defined by the parameter nPts).

```
------
Inputs
------
dStart    (1,1) or (1,:)   Days from Epoch of start or time vector
                            (sec)
dEnd          (1,1)        Days from Epoch of end
nPts          (1,1)        Number of points
model         (1,:)        Model type 'SGP', 'SGP4', 'SDP4',
                            'SGP8', 'SGD8'
pType         (1,:)        Either '2d' or '3d' plot, 3d is default
```

[8] SGP4 is a simplified perturbations model used to calculate orbital state vectors of satellites and space debris relative to the Earth-centered inertial coordinate system. This model predicts the effect of perturbations caused by the Earth's shape, drag, radiation, and gravitation effects from other bodies such as the sun and moon. The SGP4 model has an error ~1 km at epoch and grows at ~1-3 km per day [15]. That is why these data are frequently updated by NASA [16].

```
file          (1,:)        File name
-------
Outputs
-------
rV            (n)     Structure for position and velocity vectors
                      rV.v    (3,:)
                      rV.r    (3,:)
                      rV.name (1,:)
x             (n)     Structure of NORAD element data
```

In Figures 1.13-1.14 it is possible to see two examples of propagation, where the green cross is the position of Tucson (AZ), the yellow circle is the starting point and the blue circle is the ending point. In the first case (Figure 1.13) the trajectory of the International Space Station (ISS, a low orbit satellite) from February 21, 2017 at 13:18:23 (UTC time) to February 21, 2017 at 19:24:32 (UTC time) is plotted. The second case (Figure 1.14), instead, illustrates the orbit of a geostationary satellite (ECHOSTAR 1) from February 21, 2017 at 02:02:03 (UTC time) to February 22, 2017 at 02:02:03 (UTC time): as expected the satellite is almost fixed with respect to the Earth and the starting and ending points almost coincide, since the orbital period of geostationary satellites is one sidereal day (23 hours, 56 minutes).

Once this procedure is done, the final position of the satellite (or satellites) to be tracked is known. The only thing left is to project this position on the image and find the nearest bright point. However, the output of the propagation is in ECI coordinates, but spherical equatorial coordinates (RA and dec) are required in order to apply equations (1.2)-(1.3) and compute the position in pixels of the satellite. The aforementioned transformation can be implemented with the help of the free SPICE toolbox developed by NASA. However, the precise explanation of this procedure is delayed to the following section.

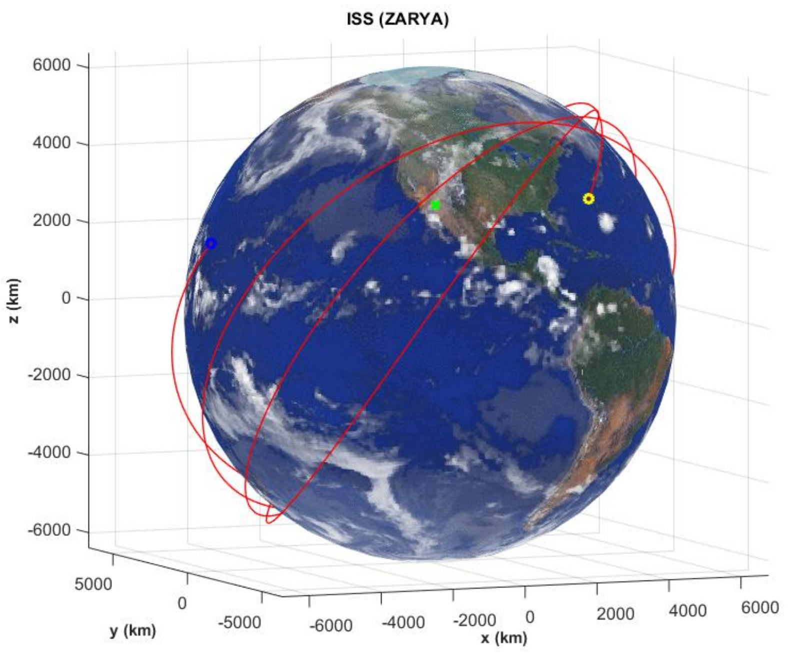


Figure 1.13: TLE propagation of the International Space Station from February 21, 2017 at 13:18:23 (UTC time) to February 21, 2017 at 19:24:32 (UTC time)

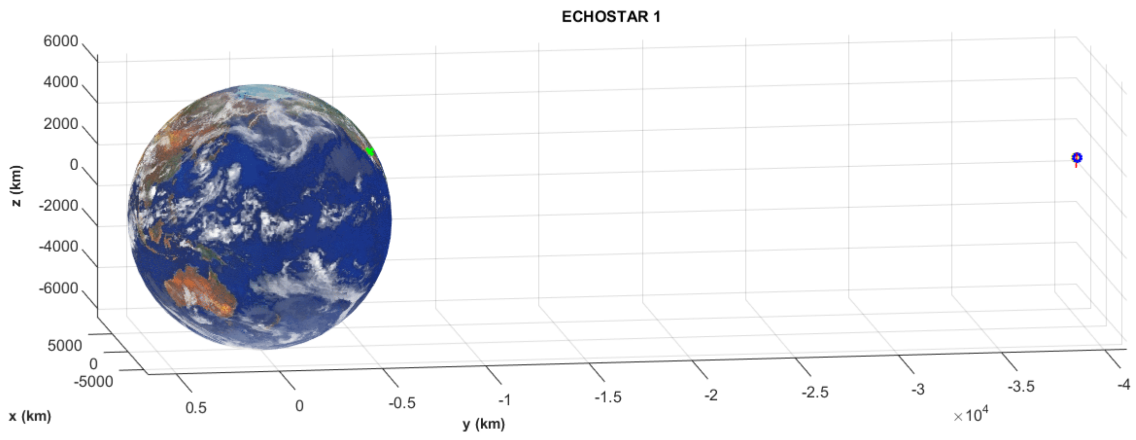


Figure 1.14: TLE propagation of ECHOSTAR 1 from February 21, 2017 at 02:02:03 (UTC time) to February 22, 2017 at 02:02:03 (UTC time)

The last step to implement is the photometry part. Photometry is about the computation of the flux from a point source, and it is measured by summing all the light recorded from the object and dividing it by the number of pixels. After the work done in paragraph 1.2.2.2, the requested task is not difficult to accomplish. In fact, the whole picture is divided in several objects, each of them composed of pixels with a given intensity. It is then possible to compute the intensity of each object summing all the pixels value (Figure 1.15). The flux, then, is obtained dividing the intensity by the number of pixels.

At the end, the relation between flux and magnitude can be obtained relating the flux of the detected stars (after the matching procedure) with the magnitude written in the catalogue.

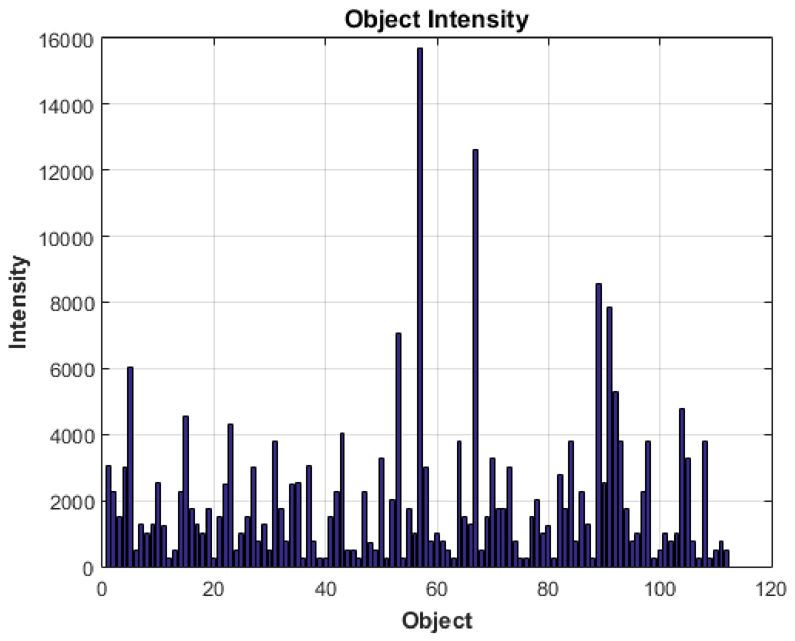


Figure 1.15: Example of objects intensity measured in an image

Within this section a simple solution to the astrometry and photometry has been taken and it has allowed to know and understand all the required passages and the theory behind. The proposed steps are working theoretically but, due to their simplicity, practical results would not probably meet the expectations. That is why in the following section a more powerful solution is given, which brings to a test with real images.

1.2.3 Next step: automated professional pipeline

As it was said at the end of previous section, the simple explained pipeline could work in practice but it would need significant improvement, which could take years of work. On the other hand, what it has been presented and mentioned so far is not useless: it can be considered as a very good starting point to understand more complete programs which can be found on the internet and make use of existing and reliable free functions.

Therefore, in order to realize a procedure that works in practice, it is decided to employ one of the few programs freely downloadable online. In particular, the choice falls on the Photometry Pipeline[9] (PP) developed by Dr. Mommert, post-doctoral researcher at the Department of Physics and Astronomy at Northern Arizona University in Flagstaff (AZ), whom the author of this work had the privilege and the honour to meet and to work with for a week. During this week it was possible to learn the working principles and the ideas behind this automated data reduction and analysis pipeline that is currently being used in different observational programs. The only problem is that this code is designed for asteroids, so it is necessary to integrate and expand it with an original solution for satellites, which is discussed in the next paragraphs.

1.2.3.1 Photometry Pipeline (PP)

As the documentation reports, the Photometry Pipeline (PP) is a Python software package for automated photometric analysis of imaging data from small to medium-size observatories. It uses Source Extractor and SCAMP to register and photometrically calibrate images based on catalogues that are available online; photometry is measured using Source Extractor aperture photometry. PP has been designed for asteroid observations, but can be used with any kind of imaging data [17].

The code runs only on Linux or Linux-based operating. PP has been tested by the author of this work also on a virtual machine in a Windows environment, so there should not be any problem related to the compatibility.

PP can be run in two different modes, providing different levels of user interaction:

- FULLY AUTOMATED MODE
  It is possible to run PP in a fully automated way just running it locally on all fits files in a specified directory, with the code:
  ```
  cd directory
  pp_run *fits
  ```

- SEMI-AUTOMATED MODE
  While `pp_run` performs all the steps automatically, each of the individual basis functions can be called manually, which allows to tweak the analysis process. The functions are:
  - `pp_prepare()`: prepare the input images and implant rough WCS[10] information into the image header

[9] All the code and documentation about this program is available and downloadable from GitHub at https://github.com/mommermi/photometrypipeline

[10] WCS stands for World Coordinate System. An elaborate set of FITS conventions has been defined to specify the physical, or world, coordinates to be attached to each pixel of an N-dimensional image. By world coordinates, one means coordinates that serve to locate a

- `pp_register()`: use SCAMP to register all input images (which means to transform different sets of data into one coordinate system) based on the implanted rough WCS information; different catalogues are tried until all images have been registered; this function calls `pp_extract()` automatically
- `pp_photometry()`: derive instrumental magnitudes using a curve-of-growth analysis, or a manually provided aperture radius
- `pp_calibrate()`: photometrically calibrate instrumental magnitudes and create a SQLite database file for each image
- `pp_distill()`: extract target information from the photometry databases created by the previous task

All the passages are extensively and accurately taken on in the next paragraphs, starting from the two freely available software which are used profusely inside the code: Source Extractor and SCAMP.

Source Extractor (SE)

As explained in previous section, the first step coincides with the identification of the bright and interesting points of the image. The Photometry Pipeline developed by Dr. Mommert makes use of Source Extractor in order to manage this task (and, later, the photometry part).

Source Extractor is a program that builds a catalogue of objects from an astronomical image. It is used for the automated detection and photometry of sources in FITS image-files. The steps of SE are the following (illustrated in Figure 1.16) [18]:

1. Measure the background and its RMS noise: the parameter called BACK_SIZE regulates the estimate. In an area of BACK_SIZE value, the mean and the standard deviation (σ) of the distribution of pixel values is computed. Then the most deviant values are discarded and median and σ are computed again. This is repeated until all the remaining pixel values are within mean $\pm 3\sigma$. If σ dropped with less than 20% per iteration, the field is considered not crowded. The value for the background in that area is, therefore, the mean in the non-crowded case and $2.5 \cdot \text{median} - 1.5 \cdot \text{mean}$ in the crowded case (both the mean and the median are the ones computed in the last iteration, where the mean is the average and the median is the average of

measurement in some multi-dimensional parameter space. A common example is to specify the right ascension and declination on the sky associated with a given pixel location in a celestial image. The WCS conventions are defined in a set of formal paper, four of which have been officially approved as part of the FITS standard by the IAU-FWG.

all the values except the most extreme one). The background map is then a bi-cubic-spline interpolation over all the areas of size BACK_SIZE, after filtering.

2. Subtract the estimated background and use the RMS noise to estimate errors.
3. Find objects (thresholding): the threshold parameters indicate the level from which SE should start treating pixels as if they were part of objects, determining information from them. In order to be considered as candidate objects, it is required that all the pixels are above the threshold, that all the pixels are adjacent to each other (either corners or sides in common), that there are more than the minimum number of pixels (defined parameter).
4. Deblend detections (break up detection into different objects): debleding is the part of SE where a decision is made whether or not a group of adjacent pixels above the threshold is a single object or not. If there is a saddle point in the intensity distribution (i.e., there are two peaks in the light distribution distinct enough), the object is split in two different ones.
5. Measure shapes and positions.
6. Clean: all the detections are checked to see if they would have been detected if their neighbors were not there. The contributions of the neighboring objects are so computed using particular models or profiles, and they are then subtracted.
7. Perform photometry: photometry is done on each object by dividing up the intensity of the shared pixels.
8. Classify: all the objects are classified into stars and galaxies (everything non-star) by a neural network.
9. Write an output catalogue with all the objects and their measured parameters (ellipticity, size, etc.).

In conclusion, this procedure is a more complex version of the one developed from scratch in section 1.2.2. In particular, the background map, deblending and cleaning operations guarantee a very good result.

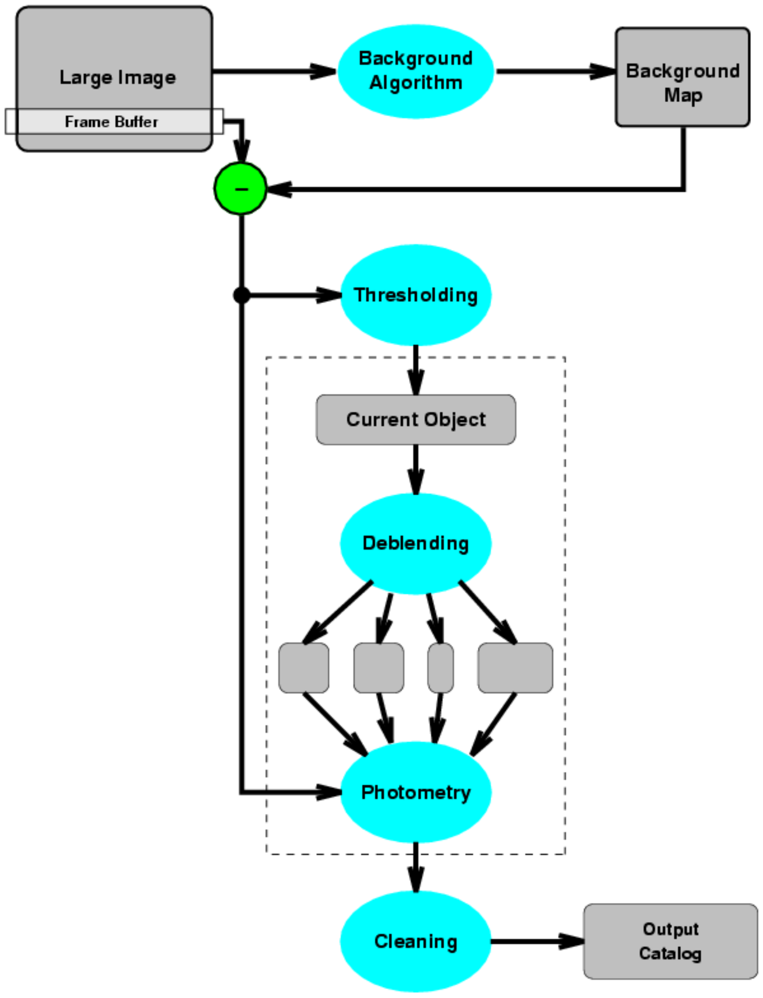


Figure 1.16: Diagram that shows the principal steps done by SE

SCAMP

SCAMP (Software for Calibrating AstroMetry and Photometry) is a program that computes astrometric projection parameters from source catalogues derived from FITS images. The computed solution is expressed according to the WCS standard. One of the main features of SCAMP is the compatibility with Source Extractor FITS catalogue format in input, so basically in the presented code SCAMP reads Source Extractor catalogues and computes astrometric and photometric solutions for any arbitrary sequences of FITS images in a complete automatic way.

The work can be decomposed into several steps (Figure 1.17) [19]:

1. Input catalogues (from SE) and headers are read and checked for content. SCAMP sorts catalogues by position on the sky ("groups of fields") as well as astrometric and photometric contexts ("instruments"). A field group is an ensemble of overlapping exposures in a given part of the sky: field groups are assembled on the basis of the relative positioning of fields. To understand the concept of astrometric instrument,

instead, it is important to realize that in order to derive an accurate astrometric solution, it is convenient to isolate contexts within which the instrument behaves in a predictable way. In SCAMP two different astrometric instruments may represent either two physically distinct devices like two different cameras, or the same device operating in two different conditions from the point of view of non-linear astrometric distorsions. Photometric instruments, finally, are to photometry what astrometric instruments are to astrometry. However, photometric instruments are generally less sensitive than astrometric ones, since there are no mechanical effects involved.

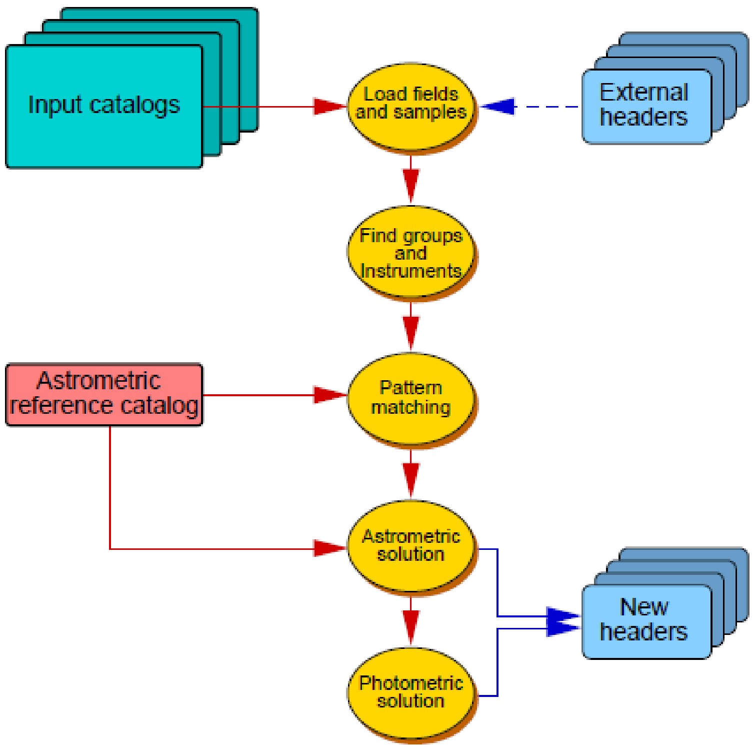


Figure 1.17: Diagram that shows the principal steps done by SCAMP

2. A catalogue of astrometric standards ("reference catalogue") is downloaded from the VizieR[11] database for every group (or a local file can be provided). Astrometric reference catalogues are required in SCAMP to locate precisely the exposures on the celestial sphere, and to attach the solution to a standard, well calibrated system. SCAMP picks only sources within a certain radius around each group.

[11] Explanation of VizieR was given in paragraph 1.2.2.1.

3. The reference catalogues are utilized by a pattern matching procedure to register all the input catalogues. Matching is a time-consuming operation and during it four parameters are derived. These parameters are corrections to the initial WCS information present in the original image headers: the pixel scale, the rotation angle and the shifts in right ascension and declination (comparing the projected positions for both catalogues).
4. All the detections and reference sources are cross-matched and an astrometric solution is computed (this solution is the result of a minimisation). This operation is repeated after clipping the outliers, that obviously alter the precision.
5. A photometric solution can be computed. The photometric capability of SCAMP however is not used in this pipeline because its operation is not totally clear: a do-it-yourself solution is preferred instead.
6. Updated astrometric and photometric calibration data are written to external headers.

In conclusion, this procedure is very similar to the one explained in section 1.2.2 but of course the matching part is quite more complex and guarantees better results. After this step, thanks to the matching with the stars in the catalogue, a very precise relationship between the position of an object in spherical equatorial coordinates (RA/dec) and the corresponding position in the image (pixels) is obtained.

Aperture radius

After these two steps, the astrometry part is completed and the photometry part seems really straightforward. Indeed, it was already anticipated that the program Source Extractor is used to complete the task. But it is true as well that some expedients have to be taken in order to increase the precision of the solution and to best exploit the capabilities of Source Extractor.

The first of these tricks is the definition of the aperture radius. As it was copiously said, the computation of the magnitude of an object, that is the computation of the flux derived from it, involves the sum of all the pixel values inside a circular aperture with a certain radius around the object, the division by the number of pixels and the subtraction of the background contribution. Source Extractor can work with different kinds of aperture, manually given or automatically calculated. In this solution, an optimal aperture radius is evaluated and given as input to Source Extractor. The logic of the choice is the following: the fractional combined flux of the target and of the background bodies is calculated together with the SNR (Signal to Noise Ratio) for an aperture radius ranging from 2 to 10 pixels. Obviously, the bigger the aperture radius, the greater the fractional combined flux (saturating at 100% after a certain value because the whole object is included in the circle) but the smaller the SNR (because more background is considered). Moreover, the fractional combined flux of the target and of the background bodies must differ from each other of 5% at maximum (in particular, the target parameter is always bigger than the one related to background bodies), because otherwise the brightness of the target would be underestimated.

So, in conclusion, the optimal aperture radius is the one for which the fractional combined flux of both target and background bodies is above a certain threshold (fixed to 70%) and the two fractional combined fluxes are inside an interval of 5%. Finally, the smallest pixel value that satisfies these requirements is chosen in order to guarantee the smallest SNR.

Photometry (calibration)

From the flux obtained with the aperture photometry explained in the last paragraphs, the magnitude is derived with the equation (1.5).

$$m = -2.5 \log_{10} f \tag{1.5}$$

where f is the flux.

However, this is not the real, or apparent, magnitude but is called the instrumental magnitude. The instrumental magnitude is defined for a specific telescope in a specific time because it depends on the characteristics of the telescope, the atmospheric conditions and many other factors. In order to obtain the apparent magnitude from the instrumental one, it is necessary to compute the magnitude zeropoint, that is simply an offset. So, at the end, equation (1.5) takes the form:

$$m = -2.5 \log_{10} f + zeropoint \tag{1.6}$$

The procedure to derive the magnitude zeropoint is quite easy:

1. Download a star catalogue from the VizieR database.
2. Match the stars with the detections found in the image.
3. Choose some reference stars between the matches (the number must be greater than 3, but obviously the greater the better).
4. Obtain the magnitude zeropoint as the minimization of the sum of the squared errors between the catalogue magnitude and the one computed with equation (1.6) for each reference star.

Once terminated these steps the apparent magnitude of all the objects inside the picture can be calculated and the astrometry and photometry problem is brilliantly solved.

Obviously, what has been presented so far is valid for one single image as well as for plenty of them.

Target identification

The last part is about target identification, that is how to get the position of the target and recognize it in the picture. The pipeline developed by Dr. Mommert is devised for asteroids (or eventually for other space objects) but not for satellites and that is the reason why a tailored solution to be integrated is necessary.

Dr. Mommert's solution solves this problem reading the name of the target and the time instant from the header file of the image and querying the JPL's HORIZONS system[12] through the web interface for the position of the target at that time. After this, the closest detection to the predicted position is chosen to perform photometry. As simple as that. But here comes the issue. JPL's HORIZONS system contains information about asteroids and other solar-system bodies, but it produces data only for a few satellites, which makes it useless for the purpose of this paper.

The integration of the pipeline, planned with Dr. Mommert, is written in Python language (because the Photometry Pipeline is a Python software package) and is reported in Appendix A. The way of thinking is quite easy: the program takes as input the target name, the Julian date in which the image was taken and the code of the observatory and gives as output the target position in spherical equatorial coordinates. Indeed, as explained in paragraph 1.1.2.3, the RA/dec coordinates of a satellite change with respect to the position of the observer.

The first part of the code (lines 59-166) takes TLE from Celestrak website (or, as an alternative, takes the TLE provided as optional input) and propagates it until the given time instant. For the task, a SGP4 model was used. This is straightforward and it is nothing new, but the matter is that the predicted position returned by the model is expressed in ECI coordinates and not in spherical equatorial coordinates as the pipeline needs. It means that a transformation of coordinates is required. In particular the transformations are two:

1. The model for TLE propagation gives back ECI TEME coordinates, so the first conversion is from ECI TEME coordinates to ECEF coordinates.
2. The second transformation is from ECEF reference frame to a spherical equatorial reference frame (RA/dec).

The first change of coordinates is performed in lines 169-190. In order to do it, it is necessary to obtain the Greenwich Mean Sidereal Time (GMST). Therefore, it is useful to introduce the concept of GMST and how to find it starting from a Julian date in Universal Time. The sidereal time is a direct measure of the Earth's rotation and it is measured positively in the counter-

[12] The JPL's HORIZONS on-line solar system data and ephemeris computation (i.e., position at a given time) service provides access to key solar system data and flexible production of highly accurate ephemerides for solar system objects (729155 asteroids, 3451 comets, 178 planetary satellites, 8 planets, the Sun and other secondary bodies). HORIZONS is provided by the Solar System Dynamics Group of the Jet Propulsion Laboratory [20].

clockwise direction when viewed from the North pole. Ideally, the observations of a star would suffice for determining sidereal time. But the changing in the instantaneous axis of rotation causes station locations to continually change. This produces a small difference in the time of meridian transits, depending on the star's declination. Because this effect vanishes at the equator, it is better to use stars with small declinations.

As described in paragraph 1.1.2.3, the equinox is defined to be always on the equator and, thus, the sidereal time is designated as the hour angle of the equinox relative to the local meridian. Because the equinox is the reference point, the sidereal time associated with the Greenwich meridian is termed Greenwich Mean Sidereal Time, ϑ_{GMST}.

The equation (1.7) shows the relation between the GMST expressed in seconds and the Julian date expressed in Universal Time [21].

$$\begin{aligned}\vartheta_{GMST} = 67310.54841 + (876600 \cdot 3600 + 8640184.812866)\, T_{UT1} + \\ + 0.093104\, T_{UT1}^2 - 6.2 \cdot 10^{-6}\, T_{UT1}^3\end{aligned} \tag{1.7}$$

where T_{UT1} is the number of Julian centuries from a particular epoch (J2000) and is equal to:

$$T_{UT1} = \frac{JD - 2451545}{36525} \tag{1.8}$$

Then ϑ_{GMST} must be divided by 240 (because $1^s = 1/240^\circ$) and multiplied by $\pi/180$ in order to pass from seconds to radians. At the end, this quantity must be reduced to a result within the range $[0,2\pi)$.

Once the GMST is computed, the transformation of coordinates can be done with this equation [22]:

$$\boldsymbol{r_{ECEF}} = R\, \boldsymbol{r_{TEME}} \tag{1.9}$$

where R is the rotation matrix, given by:

$$R = \begin{bmatrix} \cos(\vartheta_{GMST}) & \sin(\vartheta_{GMST}) & 0 \\ -\sin(\vartheta_{GMST}) & \cos(\vartheta_{GMST}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

Actually, equation (1.9) is a simplified version of the relationship. Indeed, the complete version keeps into account also the polar motion[13] of the Earth. The computation of the polar-motion matrix is not easy to obtain and, since the polar motion is a very small effect, the improvements would be negligible. For this reason, it is decided to keep the solution as simple as possible without losing precision on the result.

The second conversion (ECEF to RA/dec, lines 192-303) is applied to the coordinates obtained previously and is more challenging. In order to achieve this goal, NASA's Navigation and Ancillary Information Facility (NAIF) toolkit called SPICE is used. The SPICE toolkit is a large collection of free user-level application program interfaces (APIs), subroutine, functions and utility programs. It is offered in many languages (C, FORTRAN, MATLAB, etc.) but, since NAIF has not yet implemented a Python version, the Python interface SpiceyPy created by Andrew Annex is utilized. It is not an official product but it has been repeatedly vetted and tested by users and it is suggested by NAIF itself.

At the beginning (lines 193-281), the observer location in ECI J2000 reference frame is calculated with the aid of the program pinpoint. Pinpoint produces the ephemeris for every kind of object that is still or has a constant velocity with respect to a specific reference system. In this case, it is used to generate the ephemeris of the observer, starting from its latitude, longitude and altitude. These coordinates are obtained starting from the observatory code thanks to the list supervised by the Minor Planet Center and available at http://www.minorplanetcenter.net/iau/lists/ObsCodes.html[14].

Then (lines 284-288), the position of the target is converted from ECEF to ECI J2000 coordinates with the help of SpiceyPy pxform function that returns the matrix that transforms position vectors from one specified frame to another at a specific epoch.

At the end (lines 290-303), the relative position of the target with respect to the observer is computed and the RA/dec coordinates are finally obtained with the equations:

$$RA = \tan^{-1}\left(\frac{\rho(2)}{\rho(1)}\right) \tag{1.11}$$

$$dec = \sin^{-1}\left(\frac{\rho(3)}{\|\rho\|}\right) \tag{1.12}$$

[13] Polar motion of the Earth is the motion of the Earth's rotational axis relative to its crust. This is measured with respect to the ECEF reference frame and the variation is only a few meters.

[14] This list gives the observatory code, longitude (in degrees east of Greenwich) and the parallax constants (equal to $\rho\cos\varphi$ and $\rho\sin\varphi$ where ρ is the geocentric distance in Earth radii and φ is the latitude). Knowing them, ρ and φ can be easily computed.

where $\rho = \begin{bmatrix} \rho(1) \\ \rho(2) \\ \rho(3) \end{bmatrix}$ is the relative position between the target and the observer and $\|\cdot\|$ is the L^2 norm.

In the interest of verifying the accuracy of this solution, it is made a comparison between it and the result given by JPL's HORIZONS system for the few satellites it contains. It turns out that the declination is accurate and the right ascension suffers from an average offset of $16^s = 240''$ probably due to some approximations or secondary effects. Adding that value to the result given by equation (1.11), the error of the predicted right ascension and declination is never higher than 20 arcseconds. This error could be explained by the fact that HORIZONS uses the latest GPS-derived timing data, while SPICE uses a fixed value which could be off a fraction of a second. However, the result is sufficiently good.

In conclusion, with this integration the pipeline is able to work also with satellites, as it is possible to see in the next section.

1.2.4 Results

As anticipated in the previous section, the complex developed procedure is tested on a real image depicting a satellite, in order to verify its efficiency. The telescope image used for the test is shown in Figure 1.18. The image, courtesy of Prof. Vishnu Reddy, was taken on March 5, 2017 at 2:43:19.7 UTC time from Tucson, Arizona. Its main characteristics are:

- Focal length of the telescope = 1523.2 mm
- Aperture diameter of the telescope = 521 mm
- Field of view = 81.9' x 81.9'
- Pixel scale = 2.4" x 2.4"
- Exposure time = 2 s

The image represents, almost at its center, the GOES-14 satellite (see Figure 1.19). GOES-14 is an American weather satellite, which is part of the U.S. National Oceanic and Atmospheric Administration (NOAA)'s Geostationary Operational Environment Satellite (GOES) system. It was launched on June 27, 2009 and it reached the geostationary orbit on July 7, 2009.

Figure 1.18: Test image depicting GOES-14 satellite

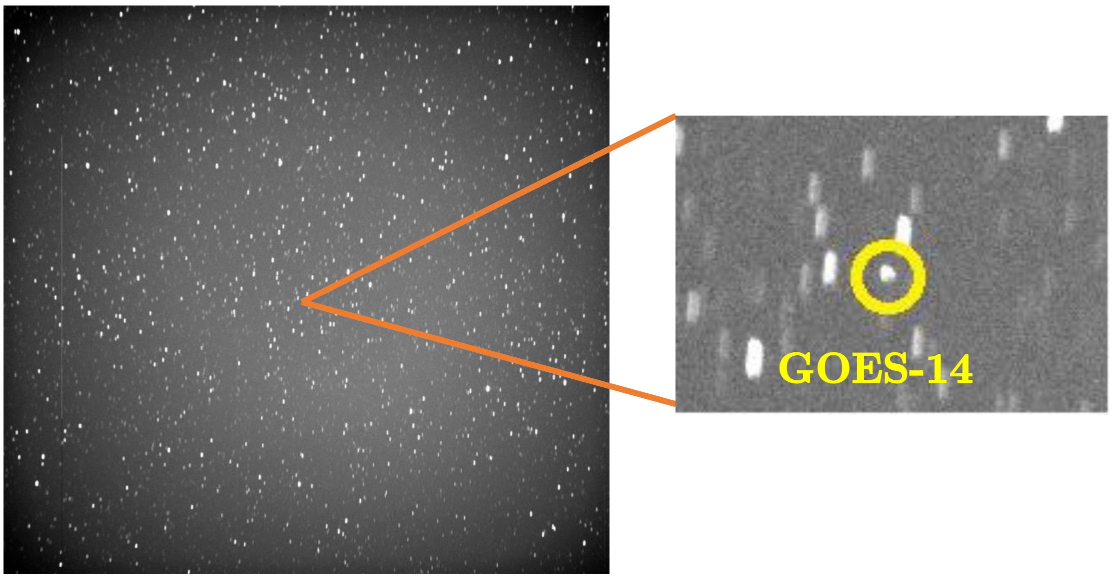


Figure 1.19: Detail of Figure 1.18 showing the position of GOES-14 satellite

As it should be clear now, the first step is the detection of the bright points (called “sources”) in the image. In this case, Source Extractor is able to find 1940 sources, which are reduced to 786 after the deblending and cleaning operations. Then, the matching procedure takes place: SCAMP queries the database VizieR for the GAIA catalogue in the portion of the sky centered with the image and with a radius that is slightly greater than the field of view (in order to compensate for potential errors and offsets). GAIA is one of the biggest star catalogues available; it is the first data release of the space observatory Gaia (September 13, 2016) and it contains the position and the magnitude of 1142679769 stars. In this case, GAIA retrieves 940 reference stars which are matched by SCAMP with the sources extracted before. Although the quality of the picture is not perfect, the result of the matching procedure can be noticed in Figure 1.20

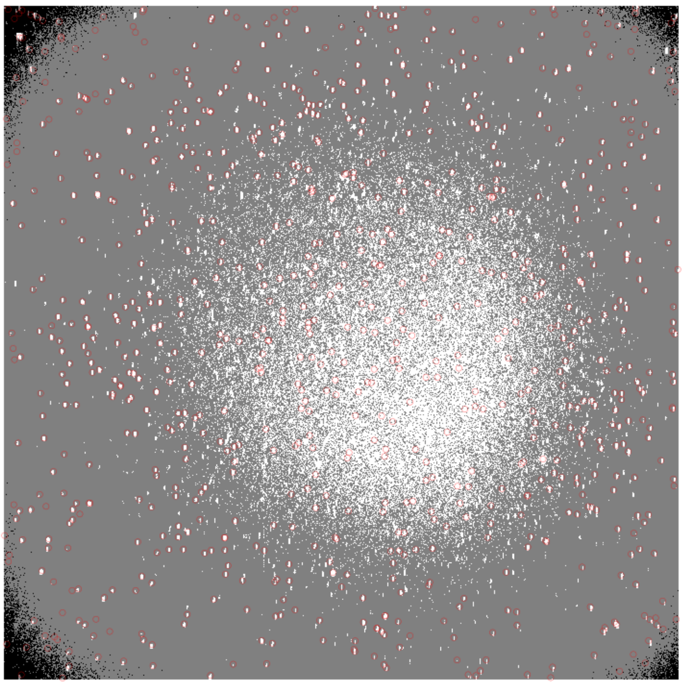


Figure 1.20: Result of the matching procedure on Figure 1.18 using the GAIA catalogue

After this step, a very precise relationship between the position of an object in spherical equatorial coordinates (RA/dec) and the corresponding position in the image (pixels) is obtained. Therefore, given the angular coordinates of the GOES-14 satellite obtained with the procedure explained in the paragraph entitled “Target identification”, the pipeline is able to identify the target in the image. As it can be seen in Figure 1.21, the identification is correct and precise (the green cross is the predicted position and the red circle is the real position). For the record, the obtained angular position of GOES-14 is $RA = 100.48045977°, dec = -5.25704468°$.

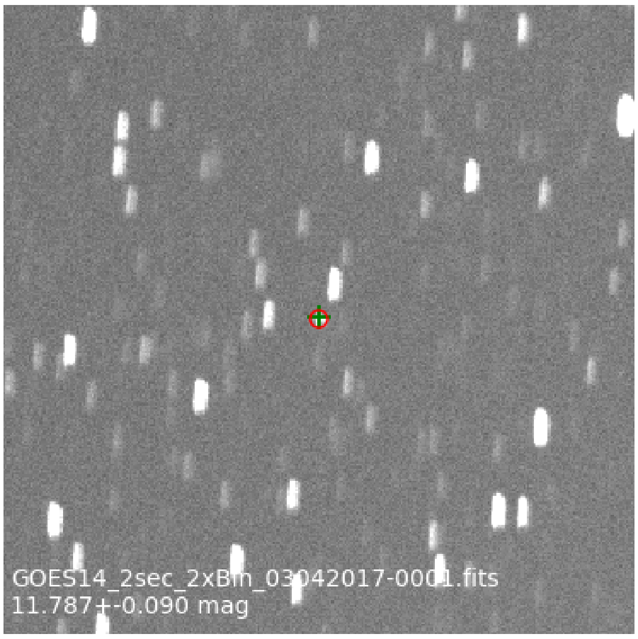


Figure 1.21: GOES-14 identification in Figure 1.18

After the identification part, the program tries to find the optimal aperture radius for the photometry, which is the last step. Following the rules described in the paragraph entitled "Aperture radius", in this case an optimal aperture radius of 3.68 pixels is chosen (Figure 1.22).

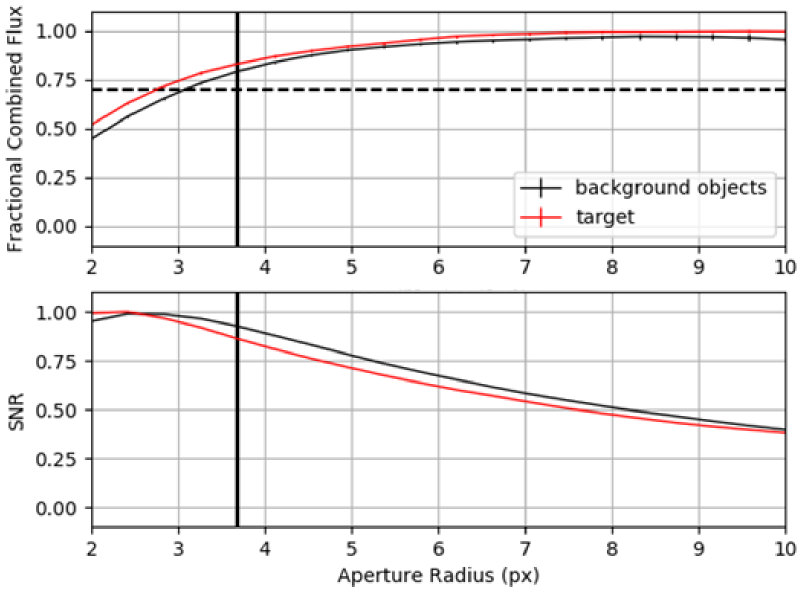


Figure 1.22: Study of the optimal aperture radius for Figure 1.18

The photometry part is completed with the calibration (i.e., the computation of the magnitude zeropoint) and the calculation of the apparent magnitude. For the calibration procedure, 17 matched reference stars are taken (Figures 1.23 and Table 1.2) and the magnitude zeropoint is obtained minimizing the sum of the squared errors between the catalogue magnitudes and the ones computed with equation (1.6). In this case the derived magnitude zeropoint is 22.6105 ± 0.0899 (Figure 1.24).

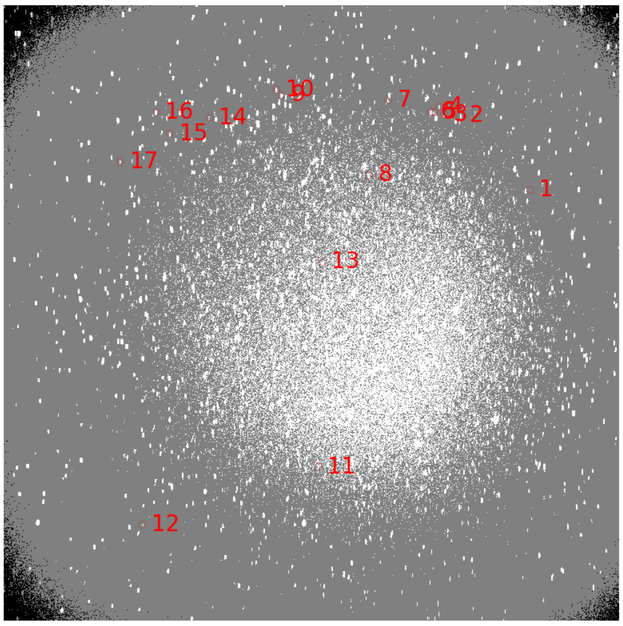


Figure 1.23: Illustration of the 17 matched reference stars used for the computation of the magnitude zeropoint

Finally, after the calibration, the apparent magnitude of the GOES-14 satellite is obtained and is equal to 13.677 ± 0.104 (Figure 1.25).

At the end, it is possible to say that the pipeline is working properly.

Idx	Name	RA	Dec	Catalog (mag)	Instrumental (mag)	Calibrated (mag)	Residual (mag
1	21081082	100.80392800	-5.75390500	14.328+-0.010	-8.185+-0.106	14.425+-0.139	0.097
2	21083752	100.96165500	-5.59050400	12.454+-0.012	-10.176+-0.017	12.435+-0.091	-0.019
3	21083797	100.96306700	-5.55393900	12.798+-0.029	-9.807+-0.024	12.804+-0.093	0.006
4	21083800	100.97942700	-5.54208300	12.942+-0.016	-9.702+-0.026	12.908+-0.094	-0.034
5	21083827	100.96659500	-5.53171400	13.692+-0.020	-9.023+-0.049	13.588+-0.102	-0.104
6	21083828	100.96702700	-5.52404700	11.747+-0.025	-10.947+-0.008	11.663+-0.090	-0.084
7	21084108	100.98787700	-5.42766700	13.406+-0.027	-9.414+-0.034	13.196+-0.096	-0.210
8	21084305	100.81853800	-5.39350500	12.874+-0.014	-9.762+-0.025	12.849+-0.093	-0.025
9	21084607	100.98674300	-5.19006400	12.248+-0.010	-10.299+-0.015	12.311+-0.091	0.063
10	21084608	100.99825800	-5.17747800	12.681+-0.028	-9.955+-0.021	12.655+-0.092	-0.026
11	21203163	100.16047600	-5.31504500	13.089+-0.010	-9.597+-0.029	13.013+-0.094	-0.076
12	21204738	100.01110100	-4.93093700	13.385+-0.028	-9.132+-0.044	13.479+-0.100	0.094
13	21204890	100.61848200	-5.29983000	14.565+-0.033	-7.942+-0.132	14.669+-0.160	0.104
14	21205769	100.92777600	-5.03245100	13.926+-0.022	-8.711+-0.065	13.900+-0.111	-0.026
15	21205816	100.88689000	-4.94639100	12.629+-0.019	-9.975+-0.020	12.636+-0.092	0.007
16	21205862	100.93335800	-4.91303500	12.426+-0.012	-10.073+-0.019	12.537+-0.092	0.111
17	21206303	100.81883800	-4.83932500	13.029+-0.015	-9.573+-0.029	13.037+-0.095	0.008

Table 1.2: List of the 17 reference stars used for the computation of the magnitude zeropoint. Each star is reported with its position, catalogue magnitude, instrumental magnitude and, finally, calibrated magnitude

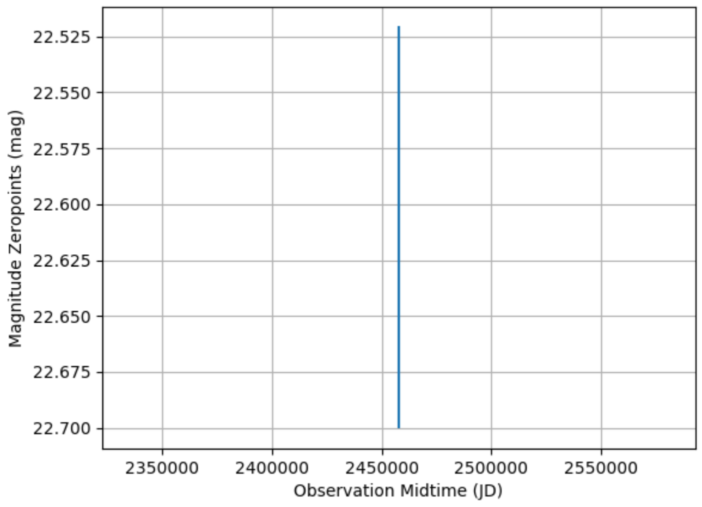


Figure 1.24: Illustration of the derived magnitude zeropoint

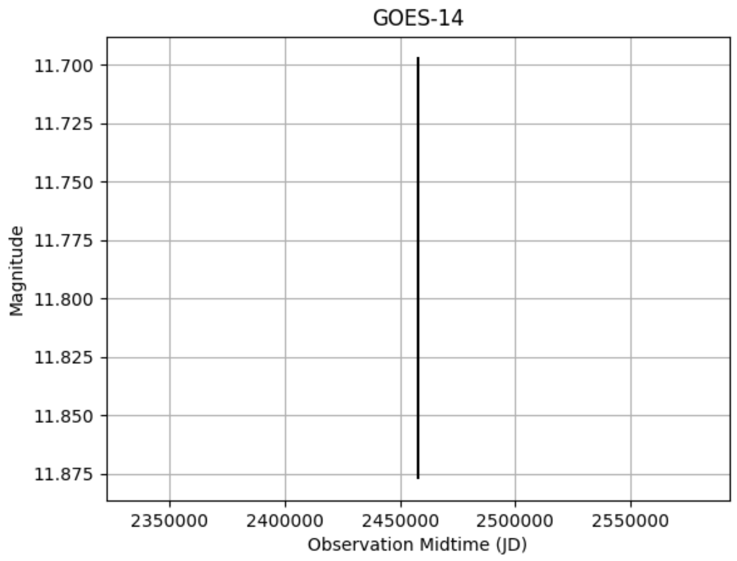


Figure 1.25: Illustration of the apparent magnitude of the GOES-14 satellite

Chapter 2
Object classification

"A breakthrough in machine learning would be worth ten Microsofts"
(Bill Gates, Chairman, Microsoft)

This chapter is focused on the subject called object classification. More precisely, it faces the interesting challenge of classifying space objects (in particular their shape and, for active bodies, their control) from their light curve, which may come from an astrometry and photometry pipeline as explained in the previous chapter. Machine learning techniques are used as tools to deal with the problem and great emphasis is given to surprising algorithms recently developed, the Extreme Learning Machines (ELM).

Therefore, the first section defines the problem and presents some previous work, the second one gets the reader in the deep world of machine learning, explaining all the techniques utilized, and the last section discloses the obtained results.

2.1 Problem setting

Tracking and characterizing both active and inactive Space Objects (SO) is required for protecting space assets (an example of possible classification is shown in Figure 2.1). Moreover, characterizing and classifying space debris is critical to understanding the threat they may pose to active satellites and manned missions. For this reason, the near-geosynchronous SO population is primarily tracked with optical sensors which provide astrometric and photometric measurements. Although the amount of light collected from these SOs is small, information can still be extracted from photometric data which can be used to determine shapes and other properties. Attitude estimation and extraction of other characteristic using light curve data has been demonstrated in several articles.

Indeed, traditional measurement sources for SO tracking, such as radar and optical, have been shown to be sensitive to shape, attitude, angular velocity and surface parameters. The state-of-the-art in the literature has been advanced over the past decade and in recent years has seen the development of multiple model, nonlinear state estimation and full Bayesian inversion approaches for SO characterization. The key shortcoming of approaches in literature is their overall computational cost.

Classification of SOs is a key challenge in Space Situational Awareness (SSA) and has many applications. The current state-of-the-art uses physics-based models to process observations and then classification is performed by determining which class of models best fits the data. These models are computationally expensive to evaluate and with the increase in the number of objects being tracked, these approaches are not practical for use on the entire catalogue. In [23], for example, a Multiple Model Adaptive Estimation (MMAE) classification approach is used to model the dynamics and physics in order to estimate relevant parameters and classify SOs. In general, there are a lot of unknown parameters that should be included in the estimation process, but the dimensionality in this complex problem does not lend itself to computationally efficient solutions.

Therefore, modern works investigate data-driven classification approaches that are not based on models but rather use the observations or data to detect relationships and use these relationships for classification. In particular, machine learning, especially in its deep learning part, is attracting more and more attention by the scientific community. In fact, recent advancements in deep learning have demonstrated ground breaking results across a number of domains. Deep learning approaches mimic the function of the brain by learning nonlinear hierarchical features from data. [24] investigates Convolutional Neural Networks (CNNs) for supervised classification of SO observational data. As opposed to the traditional approaches, [24] produces a new way of processing SO observations where quick determinations of SO classes are made possible directly from observational data.

This work considers other machine learning techinques for the purpose of classification. In particular, great attention is paid to a new learning algorithm called Extreme Learning Machine (ELM). In theory, this algorithm tends to provide good generalization performance at extremely fast learning speed.

The light curve data analysed in this chapter are kindy provided by Prof. Linares, assistant professor at University of Minnesota. They are generated by a complex model taking into account objects with different shapes and controls (to be precise, by the Ashikhmin-Shirley (AS) model, whose description, deeply given in [24], is out of the scope of this thesis). Moreover, from the same object different light curves are produced, changing for example the angular velocity of rotation and other parameters. The five different classes (debris (passive), rocket body (passive), sun pointing payload (active), spin stabilized payload (active) and nadir pointing payload (active)), which represent the label of the data, consist of objects with different shapes and controls (if they are active). This chapter considers general space objects and not only geostationary or low orbit satellites, because the main goal is to demonstrate that machine learning algorithms can be used to solve this kind of classification problems. Obviously, what is reported in this chapter can be applied to satellites too, with the appropriate amendments.

Although the approaches used here are trained with simulated data, once trained these models can be applied to real-data classification examples.

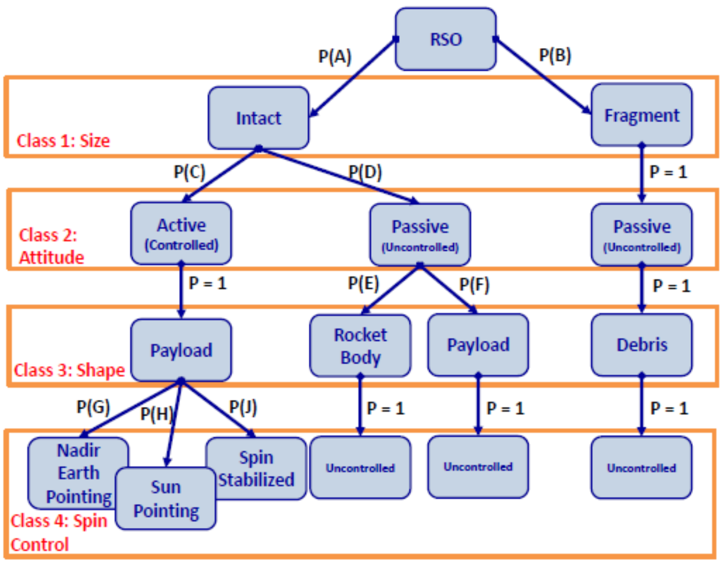


Figure 2.1: Example of possible classification procedure

2.2 Machine learning

In order to explain the main concepts of machine learning it is necessary to start understanding the definition of learning. Tom M. Mitchell, author of the textbook "Machine Learning", provided a formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [25].

Machine learning is so defined as the subfield of Artificial Intelligence (AI) and computer science that gives machines the ability to learn without being explicitly programmed (Figure 2.2). The knowledge comes from experience and induction.

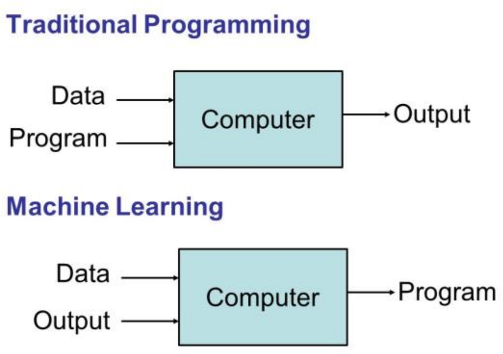


Figure 2.2: Traditional programming vs machine learning

Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Machine learning is becoming widespread in several fields like computer vision and robotics, speech recognition, biology and medicine, finance, web search, video gaming and space exploration, just to mention but a few.

Machine learning tasks are typically classified into three broad categories [26]. These are:

- SUPERVISED LEARNING. The machine is presented with example inputs and their desired outputs, given by a supervisor, and the goal is to learn a general rule that maps inputs to outputs.
- UNSUPERVISED LEARNING. No labels are given to the learning algorithm, leaving it on its own to find structure in the inputs. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
- REINFORCEMENT LEARNING. The machine interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided with feedback in terms of rewards and punishments as it navigates its problem space.

This chapter concerns only the supervised part since the objective is to learn relationships from data and use these relationships for classification. The reinforcement part, instead, is the main concept of chapter 3.

Supervised learning is the largest, most mature and most widely used sub-field of machine learning. Given a training data set including desired outputs, $D = \{< x, t >\}$, from some unknown function f, the goal is to find a good approximation of f that generalizes well on test data. Input variables x are also called features, predictors or attributes and ouput variables t are also called targets, responses or labels. If t is discrete the problem is named classification (Figure 2.3), otherwise if t is continuous it is named regression. As the title of the chapter, only classification is dealt with hereafter.

Classification has, at the end, to accomplish the task of assigning the inputs into one of K discrete classes C_k, where $k = 1, \dots, K$. There are tens of thousands of classification algorithms, hundreds new every year, but they can be easily distinguished according to their representation component, which is the way in which they represent knowledge (or the model they use). In the following only two examples of this component (neural networks and support vector machines) are explained, the ones applied later to the problem at stake.

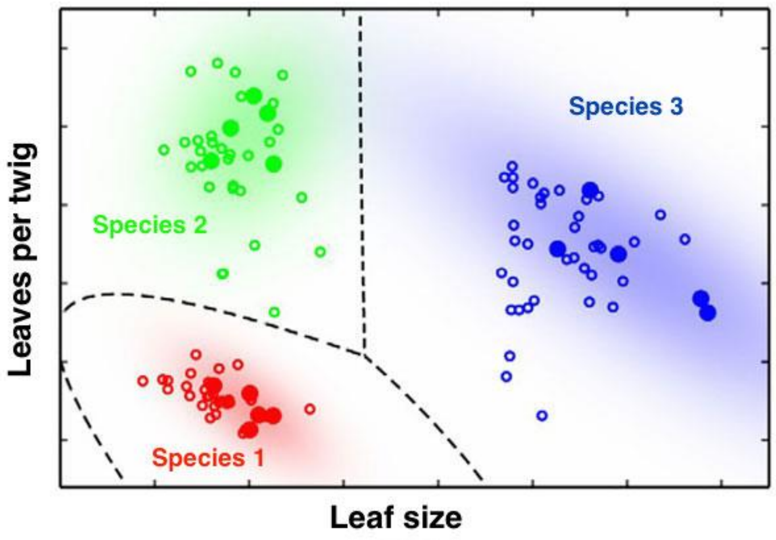


Figure 2.3: Example of classification applied to leaves

2.2.1 Neural Networks (NN)

To get started, it is better to explain a type of artificial neuron called perceptron. Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. They are a mathematical model of a biological neuron. While in actual neurons the dendrite receives electrical signals from the axons of other neurons, in the perceptron these electrical signals are represented as numerical values.

Practically, a perceptron takes several real-valued inputs $x_1, x_2, \ldots$ and produces a single binary output [27].

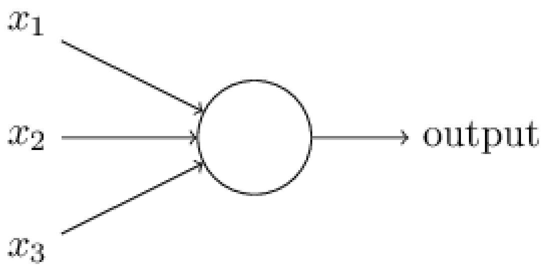


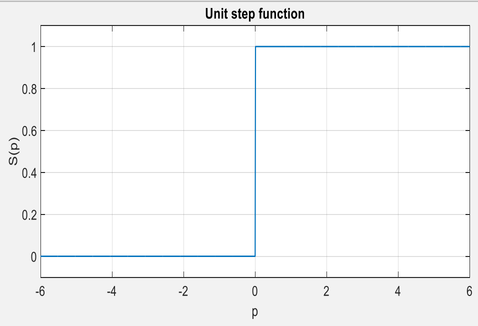
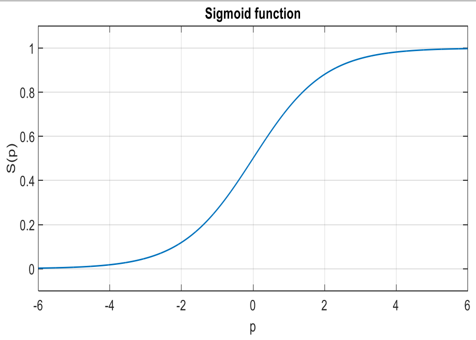
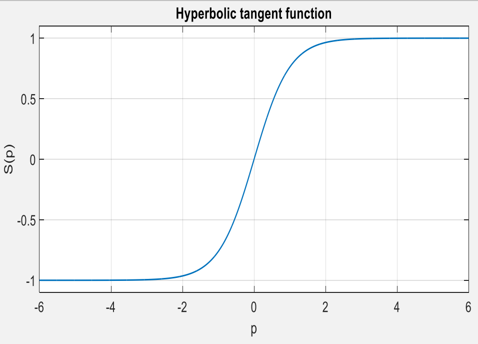
Figure 2.4: Illustration of a perceptron

In the example shown in Figure 2.4, the perceptron has three inputs, x_1, x_2, x_3. In general it could have more or fewer inputs. Rosenblatt proposed a simple rule to compute the output. He introduced weights, $w_1, w_2, \ldots$, real numbers expressing the importance of the respective inputs to the output. This models the biological effect for which at the synapses between the dendrite and axons, electrical signals are modulated in various amounts. After that, the neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

$$output = \begin{cases} 0 \ \ if \ \sum_j w_j x_j \ \leq \ threshold \\ 1 \ \ if \ \sum_j w_j x_j > \ threshold \end{cases} \tag{2.1}$$

As in biological neural networks, this output is fed to other perceptrons.

Perceptron is the most simple kind of neuron, because the function which defines the output, called activation function, is a step and so the output is just binary. But this is obviously not very useful in practice and that is the reason why in common networks other kind of activation functions (and so neurons different from perceptrons) are used. Typical and most famous activation functions are summarized in Table 2.1.

Name	Equation	Graph
UNIT STEP	$S(p) = \begin{cases} 0 \ \ if \ p \leq 0 \\ 1 \ \ if \ p > 0 \end{cases}$	
LOGISTIC (SIGMOID)	$S(p) = \dfrac{1}{1 + e^{-p}}$	
HYPERBOLIC TANGENT	$S(p) = \dfrac{e^{p} - e^{-p}}{e^{p} + e^{-p}}$	

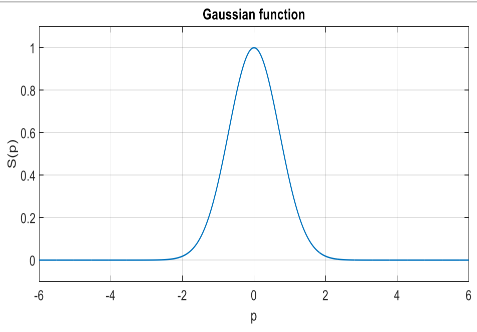
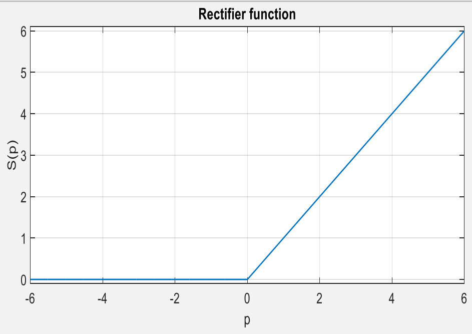
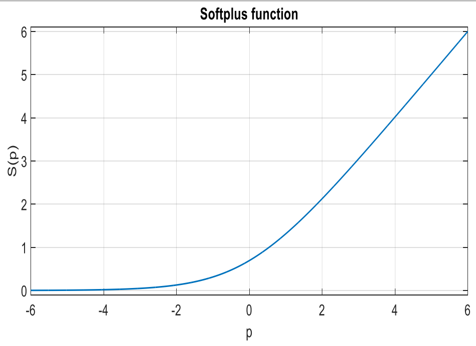
GAUSSIAN	$S(p) = e^{-p^2}$	
RECTIFIER	$S(p) = \begin{cases} 0 \ \ if\ p < 0 \\ p \ \ if\ p \geq 0 \end{cases}$	
SOFTPLUS	$S(p) = \ln(1 + e^{p})$	

Table 2.1: Summary of the most common used activation functions

Another very important activation function is the softmax, which is a generalization of the logistic one, that “squashes” a K-dimensional vector $\boldsymbol{x}$ of arbitrary real values to a K-dimensional vector $\boldsymbol{S}(\boldsymbol{x})$ of real values in the range (0,1) that add up to 1. The function is given by:

$$S(\boldsymbol{p})_j = \frac{e^{p_j}}{\sum_{k=1}^{K} e^{p_k}} \qquad \text{for } j = 1, \dots, K \tag{2.2}$$

This function is really useful, especially for classification tasks, because it can be used to represent a probability distribution over K different possible outcomes.

In all the described cases, p is equal to the weighted sum of the neuron inputs added to a bias:

$$p = \sum_j w_j x_j + b \tag{2.3}$$

Once defined the concept of neuron, it is obvious that a neuron is not a complete model of the human decision-making. But it should seem plausible that a complex network of them could make quite subtle decisions. This leads, therefore, to neural networks (Figure 2.5).

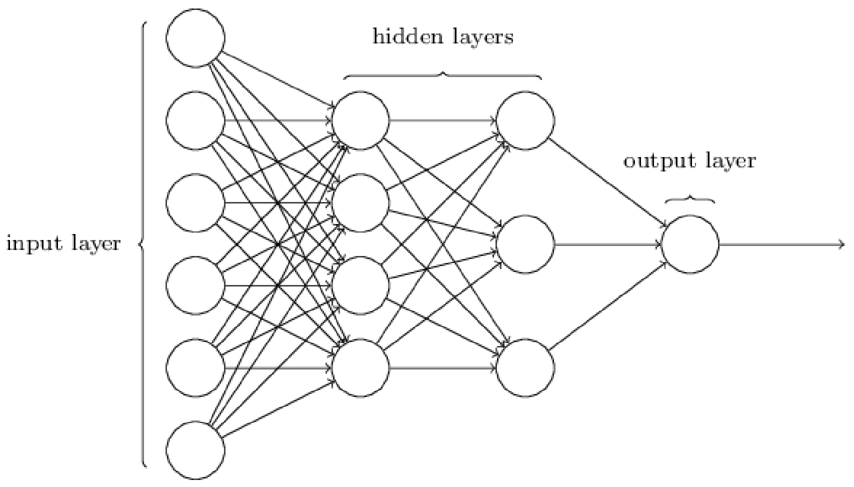


Figure 2.5: Example of neural network

As mentioned in Figure 2.5, the leftmost layer is called input layer, and the neurons within it are called input neurons. The rightmost or output layer contains the output neurons or, as in this case, a single one. The middle layers (one or more) are called hidden layers, since the neurons in it are neither inputs nor outputs. If the number of hidden layers is greater than one, the network is a deep neural network (DNN) and the relative procedure of learning the mapping between inputs and outputs is called deep learning.

Neural networks where the output from one layer is used as input to the next layer (as the one in Figure 2.5) are called feedforward neural networks. This means there are no loops, information is always fed forward, never fed back. Only this kind of networks is used in this work but for the sake of completeness it is fair to say that there are neural networks in which feedback loops are possible. These models are called recurrent neural networks and the basic idea is to have neurons which fire for some limited duration of time, before becoming quiescent.

After the design of the network, obviously it must be trained with a so called learning algorithm. For example, in the problem faced in this chapter the network has to learn how to divide input data (light curve) into classes. The standard learning algorithm for neural networks is backpropagation with gradient descent (or its variant stochastic gradient descent), described in next paragraph.

2.2.1.1 Backpropagation and gradient descent

The scope of these two algorithms, used in conjunction, is to find weights and biases so that the output from the network approximates target $\boldsymbol{t}$ for all training inputs $\boldsymbol{x}$. To quantify how well this goal is achieved it is necessary to define a cost function. For backpropagation to work the cost function has to satisfy two main assumptions:

- The cost function C can be written as an average over cost functions C_x for individual training examples, $\boldsymbol{x}$.
- The cost function C can be written as a function of the outputs from the neural network.

There are many possible choices for the cost function, this work employs the quadratic cost function, sometimes known as the mean square error or just MSE:

$$C(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{2n} \sum_i \| \boldsymbol{t}(\boldsymbol{x_i}) - \boldsymbol{a^L}(\boldsymbol{x_i}) \|^2 \tag{2.4}$$

In equation (2.4) n is the total number of training examples, the sum is over individual training examples, $\boldsymbol{x_i}$, $\boldsymbol{t} = \boldsymbol{t}(\boldsymbol{x_i})$ is the corresponding desired output, L denotes the number of layers in the network and $\boldsymbol{a^L} = \boldsymbol{a^L}(\boldsymbol{x_i})$ is the vector of actual output from the network when $\boldsymbol{x_i}$ is the input. Obviously, the cost function depends on the weights $\boldsymbol{w}$ and on the biases $\boldsymbol{b}$.

Looking at the form of the quadratic cost function, it is possible to realize that $C(w, b)$ is non-negative, since every term in the sum is non-negative. Furthermore, $C(\boldsymbol{w}, \boldsymbol{b})$ becomes small when the output $\boldsymbol{a^L}(\boldsymbol{x_i})$ is approximately equal to the desired one, $\boldsymbol{t}(\boldsymbol{x_i})$, for all training inputs $\boldsymbol{x_i}$. So the training algorithm does a good job if it can find weights and biases so that $C(\boldsymbol{w}, \boldsymbol{b}) \approx 0$ and, by contrast, it's not doing so well when $C(\boldsymbol{w}, \boldsymbol{b})$ is large. In other words, the goal is to find a set of weights and biases which make the cost as small as possible. The most common optimization method is the gradient descent (or its variant stochastic gradient descent).

One way of attacking the problem is to use calculus to try to find the minimum analytically. Derivatives could be computed and used to find places where C has an extremum. With some luck that might work when C is a function of just one or a few variables. But it becomes really difficult when there are many more variables, as in case of neural networks. Using calculus to minimize the cost function just will not work.

In order to understand the operating mode of the gradient descent it is necessary to think about what happens changing by a small amount weights w_i, $i = 1, \dots, W$ (with W equal to the total number of weights) and biases b_j, $j = 1, \dots, B$ (with B equal to the total number of biases). Calculus says that the cost function C changes as follows:

$$\Delta C \approx \frac{\partial C}{\partial w_1}\Delta w_1 + \cdots + \frac{\partial C}{\partial w_W}\Delta w_W + \frac{\partial C}{\partial b_1}\Delta b_1 + \cdots + \frac{\partial C}{\partial b_B}\Delta b_B \tag{2.5}$$

where $\frac{\partial(\cdot)}{\partial(\cdot)}$ indicates, as common notation, a partial derivative.

The choices of $\Delta w_i,\ i = 1, \dots, W$ and $\Delta b_j,\ j = 1, \dots, B$ should be made in order ΔC to be negative. In this way, C is going to decrease step by step. To figure out how to make such a choice it helps to define $\Delta \boldsymbol{v}$ to be the vector of changes:

$$\Delta \boldsymbol{v} \equiv (\Delta w_1, \dots, \Delta w_W, \Delta b_1, \dots, \Delta b_B)^T \tag{2.6}$$

Then, it is the turn to define the gradient of C to be the vector of partial derivatives:

$$\boldsymbol{\nabla C} \equiv \left(\frac{\partial C}{\partial w_1}, \dots, \frac{\partial C}{\partial w_W}, \frac{\partial C}{\partial b_1}, \dots, \frac{\partial C}{\partial b_B}\right)^T \tag{2.7}$$

With these definitions, the expression (2.5) can be rewritten as

$$\Delta C \approx \boldsymbol{\nabla C} \cdot \Delta \boldsymbol{v} \tag{2.8}$$

where $(\cdot)$ defines the dot product between two vectors.

Equation (2.8) helps explaining why $\boldsymbol{\nabla C}$ is called the gradient vector: $\boldsymbol{\nabla C}$ indeed relates changes in $\boldsymbol{v}$ to changes in C. But what is really exciting about the equation is that it shows how to choose $\Delta \boldsymbol{v}$ so as to make ΔC negative. In particular, suppose to choose

$$\Delta \boldsymbol{v} = -\eta \boldsymbol{\nabla C} \tag{2.9}$$

where η is a small, positive parameter (known as the learning rate). Then, equation (2.8) reveals that:

$$\Delta C \approx \boldsymbol{\nabla C} \cdot (-\eta \boldsymbol{\nabla C}) = -\eta \|\boldsymbol{\nabla C}\|^2 \tag{2.10}$$

Because $\|\boldsymbol{\nabla C}\|^2 \geq 0$, this guarantees that $\Delta C \leq 0$ and so that C will always decrease, never increase, if the weights and the biases are changed according to the expression (2.9). Writing out the gradient descent update rule in terms of components, it is possible to obtain:

$$w_i \rightarrow w_i' = w_i - \eta \frac{\partial C}{\partial w_i} \tag{2.11}$$

$$b_j \rightarrow b'_j = b_j - \eta \frac{\partial C}{\partial b_j} \tag{2.12}$$

By repeatedly applying this update rule it is possible to find a minimum of the cost function, moving at each step in the direction opposite with respect to the gradient (Figure 2.6). In other words, this is a rule which can be used to learn in a neural network.

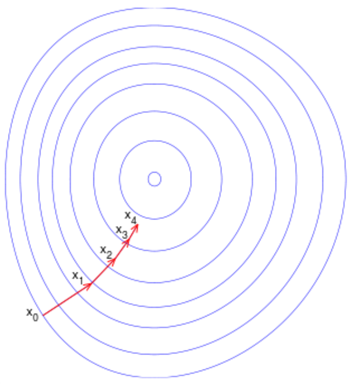


Figure 2.6: Illustration of the steps performed by gradient descent

In order to make the gradient descent work correctly, it is needed to choose the learning rate η to be small enough so that equation (2.8) is a good approximation. Otherwise, it could be possible to end up with $\Delta C > 0$, which is obviously not desirable. At the same time, η cannot be too small, since that will make the changes $\boldsymbol{\Delta v}$ tiny, and thus the gradient descent algorithm will work very slowly. There is so a trade-off.

Although gradient descent often works extremely well, it can get stuck in local minima. One way to try to solve the problem of local minima is to initialize the network in different modes. Another way (which can be used in conjuction with multiple initializations) is to slightly modify the update rule (2.11) as (same for (2.12)):

$$w_i(t+1) \rightarrow w_i(t+2) = w_i(t+1) - \eta \frac{\partial C}{\partial w_i(t+1)} + \alpha\big(w_i(t+1) - w_i(t)\big) - \lambda\eta w_i(t+1) \tag{2.13}$$

where the term $\alpha\big(w_i(t+1) - w_i(t)\big)$ is called momentum and is used to diminuish the fluctuations in weight changes over consecutive iterations (this helps in avoiding local minima).

The term $\lambda\eta w_i(t+1)$, instead, is called weight decay and penalizes the weight changes (this is a regularization[15] term). Obviously, here t defines the time step.

There is, however, a problem that is worth mentioning: recalling the expression (2.4) of the cost function, C can be rewritten as:

$$C = \frac{1}{n}\sum_i C_{x_i} \tag{2.14}$$

which means that it is an average over costs $C_{x_i} \equiv \frac{\|\boldsymbol{t(x_i)}-\boldsymbol{a^L(x_i)}\|^2}{2}$ for individual training examples. In practice, to compute the gradient $\boldsymbol{\nabla C}$ it is necessary to compute the gradients $\boldsymbol{\nabla C_{x_i}}$ separately for each training input, $\boldsymbol{x_i}$, and then average them:

$$\boldsymbol{\nabla C} = \frac{1}{n}\sum_i \boldsymbol{\nabla C_{x_i}} \tag{2.15}$$

Unfortunately, when the number of training inputs is very large this can take a long time, and learning thus occurs slowly. And here comes a variant of the gradient descent, called stochastic gradient descent, which can be used to speed up learning. The idea is to estimate the gradient $\boldsymbol{\nabla C}$ by computing $\boldsymbol{\nabla C_{x_i}}$ for a small sample of randomly chosen training inputs (or even a single one). By averaging over this small sample it turns out that it is possible to get quickly a good estimate of the true gradient $\boldsymbol{\nabla C}$, and this helps to speed up learning.

So far, the computation of the gradient of the cost function has been taken for granted. That is not a trivial task but fortunately the aforementioned backpropagation comes to the aid. The backpropagation algorithm was originally introduced in the 1970s, but its importance was not fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton and Ronald Williams. That paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural networks to solve problems which had previously been insoluble. Today, the backpropagation algorithm is the workhouse of learning in neural networks.

Before listing the theoretical contributions offered by backpropagation, in order not to get confused it is important to explain the notation which is used in the following:

- w_{jk}^l will denote the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer;

[15] Regularization is a technique used to avoid the problem of overfitting, which happens when the model fits the training data but does not have good predicting performance and generalization power. Regularization acts constraining the norm of the coefficients, controlling so their growth, their variance and the variance of the output of the network.

- b_j^l will define the bias of the j^{th} neuron in the l^{th} layer;
- a_j^l will identify the output of the j^{th} neuron in the l^{th} layer.

The equation relating all the previous terms is:

$$a_j^l = S\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{2.16}$$

where $S(\cdot)$ is the activation function. Equation (2.16) can be rewritten in matrix form as:

$$\boldsymbol{a^l} = S\left(W^l \boldsymbol{a^{l-1}} + \boldsymbol{b^l}\right) \tag{2.17}$$

where W^l is the weight matrix for the layer l (it means that the element in the j^{th} row and k^{th} column is w_{jk}^l), $\boldsymbol{a^{l-1}}$ and $\boldsymbol{b^l}$ are respectively the output vector and the bias vector for layer $l-1$ and l.

Backpropragation, whose task is to understand how changing the weights and biases in the network changes the cost function, is based on four fundamental equations (reported in the block (2.19)). These equations relate three important parameters: the partial derivatives $\frac{\partial C}{\partial w_{jk}^l}$, $\frac{\partial C}{\partial b_j^l}$ and the error in the j^{th} neuron of the l^{th} layer, δ_j^l. This last term is defined as:

$$\delta_j^l \equiv \frac{\partial C}{\partial p_j^l} \tag{2.18}$$

where p_j^l is the input of the activation function for the j^{th} neuron in the l^{th} layer. As before, $\boldsymbol{\delta^l}$ is used to denote the vector of errors associated with layer l.

$$\boxed{\begin{array}{c} 1.\ \ \delta_j^L = \frac{\partial C}{\partial a_j^L} S'\left(p_j^L\right) \\[2em] 2.\ \ \delta_j^l = \left(\sum_m w_{mj}^{l+1} \delta_j^{l+1}\right) S'\left(p_j^l\right) \\[2em] 3.\ \ \delta_j^l = \frac{\partial C}{\partial b_j^l} \\[2em] 4.\ \ a_k^{l-1} \delta_j^l = \frac{\partial C}{\partial w_{jk}^l} \end{array}} \tag{2.19}$$

In block (2.19), $S'(\cdot)$ indicates the derivative of the activation function with respect to the input p. These equations could be expressed in matrix form using the Hadamard product[16] and they could be mathematically demonstrated, but that is out of the scope of this work. The only important thing regarding this result is that it allows to compute the error δ_j^l for each neuron and the partial derivatives $\frac{\partial C}{\partial w_{jk}^l}$, $\frac{\partial C}{\partial b_j^l}$ for each weight and bias, starting from the output layer and backpropagating the values towards the input layer (hence the name of the algorithm).

Although the gradient descent and the backpropagation perform well on simple neural networks (with only one hidden layer), they can fail with deep neural networks. The failure can be explained by the very probable possibility that the different layers in the deep neural network are learning at vastly different speeds. In particular, when later layers in the network are learning well, early layers often get stuck during training, learning almost nothing at all. The opposite situation (when early layers are learning well and later layers become stuck) is possible too. This instability is related to the use of gradient-based learning techniques (as gradient descent) and is caused mainly by two problems: the vanishing gradient and the exploding gradient problems.

The common issue is that the gradient is getting smaller (vanishing) or larger (exploding) as moving backward through the hidden layers. This means that neurons in the earlier layers learn, respectively, much more slowly or much more fastly than neurons in later layers. More generally, it turns out that the gradient in deep neural networks is unstable, tending to either explode or vanish in earlier layers. This instability, related also to the derivative of the activation function, is a fundamental problem for gradient-based learning in deep neural netoworks and it must kept in mind when dealing with these situations.

There are possible solutions for these issues, even if correct working is not guaranteed. Vanishing gradient problem depends principally on the choice of the activation function. For example, sigmoid function and hyperbolic tangent function (expressed in Table 2.1) compress their input into a very small output range in a very nonlinear fashion. As a result, there are large regions of the input space which are mapped to an extremely small range. In these regions of the input space, even a large change in the input will produce a small change in the output (hence a small gradient). It is possible to avoid this by using activation functions which do not have the property of compressing the input space. A popular choice in last years is the rectifier function.

Another solution (explained in details later) which can be adopted is the layer-by-layer unsupervised pre-training approach (like autoencoders) in order to improve the weights initialization and to reduce the risk of vanishing gradients.

However, the choice of activation functions which do not limit the output can cause the opposite problem, the exploding gradient. A solution used in practice is called gradient clipping

[16] The Hadamard product $g \odot h$ between two vectors g and h denotes the elementwise product of the two vectors.

and lies in keeping the gradient under a certain threshold in order to prevent it from getting too large.

2.2.2 Support Vector Machines (SVM)

SVM are one of the possible representations of a classification algorithm. Actually, they are one of the best methods for classification. They have a long history but they were invented in the present form in the late 90's by Boser, Guyon and Vapnik.

A support vector machine is a discriminative classifier (inherently binary classifier) formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new samples. In order to understand the meaning of optimal hyperplane, it is useful to give an example. The simple problem is the following: for a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line [28].

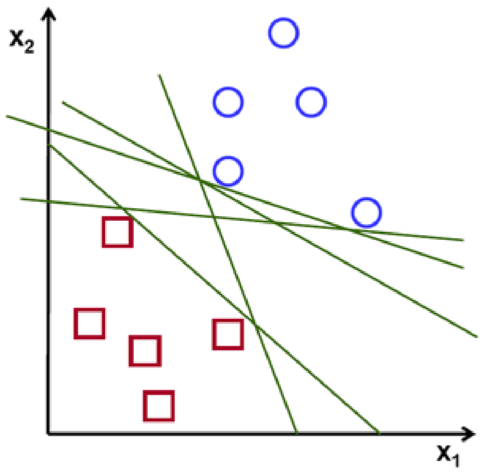


Figure 2.7: The problem of finding a separating straight line between two classes of points

Figure 2.7 shows that there exist multiple lines that offer a solution to the problem, some better than others. Indeed, it is reasonable to say that a line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. Therefore, the goal should be to find the line passing as far as possible from all points. Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, the distance receives the important name of margin within SVM's theory. Therefore, the optimal separating hyperplane maximizes the margin of the training data.

In the following a simple but efficient description of the working principles of SVM is given [29]. There are two things which must be defined: how to maximize the margin and how to decide whether new data belong to one class or to the other (called decision rule). Starting with the decision rule and supposing the optimal hyperplane is drawn (Figure 2.8), the first thing to do is to build a vector $\boldsymbol{\omega}$ of any length constrained to be perpendicular to the hyperplane.

When an unknown point in the plane needs to be classified (whether it is a circle (+) or a square (-)), it is important to understand if that point is on the left or on the right of the hyperplane, projecting the vector $\boldsymbol{u}$ that defines its position on $\boldsymbol{\omega}$ and so perpendicularly to the hyperplane.

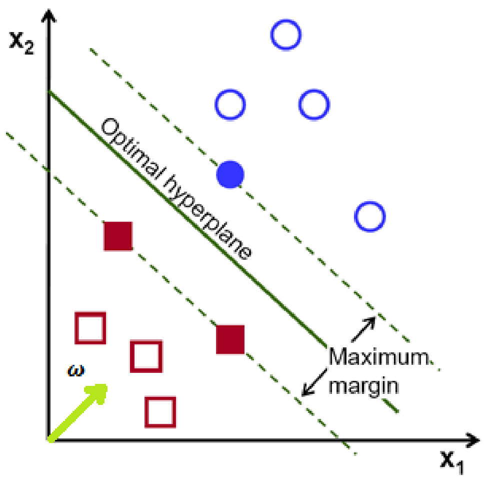


Figure 2.8: Graphical representation of the optimal hyperplane

The decision rule is so easily determined: the unknown data is a + if and only if

$$\boldsymbol{\omega} \cdot \boldsymbol{u} + b \geq 0 \tag{2.20}$$

for a vector $\boldsymbol{\omega}$ and a scalar b (with $\boldsymbol{\omega}$ perpendicular to the hyperplane), otherwise it is a -.

In order to find $\boldsymbol{\omega}$ and b, it is necessary to add more constraints to the problem, demanding that:

$$\boldsymbol{\omega} \cdot \boldsymbol{x}_+ + b \geq 1 \tag{2.21}$$

$$\boldsymbol{\omega} \cdot \boldsymbol{x}_- + b \leq -1 \tag{2.22}$$

for all + samples $\boldsymbol{x}_+$ and $-$ samples $\boldsymbol{x}_-$ which represent the labeled training data.

Introducing the variable y_i such that:

$$y_i = \begin{cases} +1 & for + samples \\ -1 & for - samples \end{cases} \tag{2.23}$$

it is possible to rewrite the equations (2.21)-(2.22) as a single one:

$$y_i(\boldsymbol{\omega} \cdot \boldsymbol{x}_i + b) - 1 \geq 0 \tag{2.24}$$

for every sample (positive or negative) $\boldsymbol{x}_i$.

Additionally, it can be imposed that:

$$y_i(\boldsymbol{\omega} \cdot \overline{\boldsymbol{x}}_i + \boldsymbol{b}) - 1 = 0 \tag{2.25}$$

for all the samples $\overline{\boldsymbol{x}}_i$ on the margin (dashed lines in Figure 2.8). In general, the training examples $\overline{\boldsymbol{x}}_i$ that are closest to the hyperplane are called support vectors (hence the name of the algorithm).

The final goal of SVM, just to remind, is to maximize the margin of the training data. The margin can be expressed as:

$$margin = (\overline{\boldsymbol{x}}_+ - \overline{\boldsymbol{x}}_-) \cdot \frac{\boldsymbol{\omega}}{||\boldsymbol{\omega}||} \tag{2.26}$$

where $\overline{\boldsymbol{x}}_+$ is a + sample on the margin and $\overline{\boldsymbol{x}}_-$ is a − sample on the margin. Inserting equation (2.25) in (2.26), the result is:

$$margin = \frac{1-b+1+b}{||\boldsymbol{\omega}||} = \frac{2}{||\boldsymbol{\omega}||} \tag{2.27}$$

Maximizing (2.27) is the same as minimizing $||\boldsymbol{\omega}||$ which turns out to be equal to minimize $\frac{1}{2}||\boldsymbol{\omega}||^2$, just to make the expression mathematically convenient. The objective function to be minimized, $\frac{1}{2}||\boldsymbol{\omega}||^2$, is quadratic and the constraints (2.24) which must be satisfied are linear. Therefore, this appears to be a quadratic problem that can be solved with the aid of the Lagrangian function, transforming the problem from a constrained to an unconstrained one. The general expression of the Lagrangian function is:

$$L(\boldsymbol{\omega}, \boldsymbol{\alpha}, b) = f(\boldsymbol{\omega}) - \sum_i \alpha_i g_i(\boldsymbol{\omega}, b) \tag{2.28}$$

where $f(\boldsymbol{\omega}, b)$ is the objective function, $g_i(\boldsymbol{\omega}, b)$ are the inequality constraints (2.24) and α_i are called the Lagrange multipliers.

Given the problem stated above, Karush-Kuhn-Tucker (KKT) conditions for nonlinear optimization say that if $\boldsymbol{\omega}^*, b^*, \boldsymbol{\alpha}^*$ is a minimum of (2.28), then these conditions must be verified:

$$\boxed{\begin{array}{c} 1.\ \ \nabla L(\boldsymbol{\omega}^*, \boldsymbol{\alpha}^*, b^*) = 0 \\ \\ 2.\ \ g_i(\boldsymbol{\omega}^*, b^*) \leq 0 \\ \\ 3.\ \ \alpha_i^* \geq 0 \\ \\ 4.\ \ \alpha_i^* g_i(\boldsymbol{\omega}^*, b^*) = 0 \end{array}} \tag{2.29}$$

In particular, the last condition is really interesting: indeed it states that, for each sample, either $g_i(\boldsymbol{\omega}^*, b^*) = 0$ (this happens if the sample is on the margin) or $\alpha_i^* = 0$ (this means that its multiplier is equal to zero).

Applying all these theoretical concepts to the problem under consideration, the Lagrangian function takes the form:

$$L(\boldsymbol{\omega}, \boldsymbol{\alpha}, b) = \frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_i \alpha_i [y_i(\boldsymbol{\omega} \cdot \boldsymbol{x_i} + b) - 1] \tag{2.30}$$

and its derivatives with respect to $\boldsymbol{\omega}, \boldsymbol{\alpha}, b$ (which must be put equal to zero for condition 1 in (2.29)) are:

$$\frac{\partial L}{\partial \boldsymbol{\omega}} = \boldsymbol{\omega} - \sum_i \alpha_i y_i \boldsymbol{x_i} = 0 \quad \Rightarrow \quad \boldsymbol{\omega} = \sum_i \alpha_i y_i \boldsymbol{x_i} \tag{2.31}$$

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_i \alpha_i y_i = 0 \tag{2.32}$$

$$\frac{\partial L}{\partial \boldsymbol{\alpha}} = 0 \quad \Rightarrow \quad b = \frac{1}{N} \sum_i \left(y_i - \sum_j \alpha_j y_j \boldsymbol{x_i^T} \cdot \boldsymbol{x_j} \right) \tag{2.33}$$

where N is the number of samples. This solves the problem in a very intelligent way: in fact, due to condition 5 in (2.29), not all the samples should be considered for the computation of $\boldsymbol{\omega}$ and b, but only the ones on the margin (support vectors) since for the others the weight α_i is going to be zero. Although the presented algorithm seems surprisingly powerful, so far it can work only if the training data are linearly separable. But with a little effort its efficiency can be incredibly improved. Indeed, substituting equations (2.31)-(2.32) in (2.30) and doing some calculations, the Lagrangian function turns out to be:

$$L(\boldsymbol{\omega}, \boldsymbol{\alpha}, b) = \sum_i \alpha_i - \frac{1}{2} \sum_j \alpha_i \, \alpha_j y_i y_j \boldsymbol{x_i^T} \cdot \boldsymbol{x_j} \tag{2.34}$$

and the decision rule (2.20) becomes:

$$\sum_i \alpha_i y_i \boldsymbol{x_i^T} \cdot \boldsymbol{u} + b \geq 0 \tag{2.35}$$

It is clear from equations (2.34)-(2.35) that the Lagrangian function to be minimized and the decision rule depend only on the dot product between data. This means that it is possible to use the so called kernel trick and apply a transformation to the data projecting them in a certain space (even with infinite dimensions) where they are linearly separable. The strength of the kernel trick is that it make not indispensable the calculation of the transformation which can be sometimes almost impossible. All that is needed is a function, exactly the kernel function, defined as:

$$k: \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R} \tag{2.36}$$

where m is the dimension of the data. The kernel function satisfies the relationship:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \phi(\boldsymbol{x_i})^T \cdot \phi(\boldsymbol{x_j}) \tag{2.37}$$

in which $\phi(\cdot)$ defines the transformation which need not to be known though.

The most common used kernel functions are:

- LINEAR KERNEL

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i^T} \cdot \boldsymbol{x_j} \tag{2.38}$$

- POLYNOMIAL KERNEL

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i^T} \cdot \boldsymbol{x_j} + c)^d \tag{2.39}$$

- GAUSSIAN KERNEL

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp\left(-\gamma \|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2\right) \tag{2.40}$$

- HYPERBOLIC TANGENT KERNEL

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(a\boldsymbol{x_i^T} \cdot \boldsymbol{x_j} + c) \tag{2.41}$$

The last surprising aspect to be mentioned is that SVM algorithm always guarantees a convex weight optimization problem, and this means that it cannot be stuck in local minima.

2.2.3 Extreme Learning Machines (ELM)

After the illustration of the two main representations for classification algorithms, neural networks and SVM, in this paragraph a new and powerful technique is presented (and then tested in next section): the Extreme Learning Machines (ELM). Theoretically, ELM are a particular kind of feedforward neural networks with a single layer of hidden nodes which do not make use of backpropagation as learning algorithm. In fact, backpropagation is a step-by-step algorithm; ELM, instead, utilize a distinctive learning algorithm, explained later, which allows learning in a single step. Although ELM were originally conceived as feedforward neural network with one single hidden layer, their particular training algorithm can be applied to almost every kind of neural networks (CNNs [30], Local Receptive Fields NNs [31], Multi-Layer NNs [32],…).

Even if the main concepts had already been in the scientific community for years, ELM were formally described and introduced by Huang in 2004. According to their creators, these models are able to produce good generalization performance and learn thousands of times faster than networks trained using backpropagation.

Starting from simple single layer feedforward networks (SLFN, explained in paragraph 2.2.1), the universal approximation theorem states that any continuous target function $\boldsymbol{f(x)}$ can be approximated by SLFNs with some kind of hidden nodes and with appropriate parameters. In other words, given any small positive value ε, for SLFNs with enough number of hidden neurons (L) it is true that:

$$\|\boldsymbol{f}_L(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\| < \varepsilon \tag{2.42}$$

where $\boldsymbol{f}_L(\boldsymbol{x})$ is the output of the SLFN. However, the theorem does not touch upon the algorithmic learnability of the required parameters. Indeed, in real applications target function $\boldsymbol{f}$ is usually unknown and there is only the hope that it can be approximated by $\boldsymbol{f_L}$ appropriately. As described in paragraph 2.2.1, many learning methods mainly based on gradient descent or other iterative approaches have been developed over the past two decades. Backpropagation and its variants are the most popular. Although the basic idea is simple and quite efficient, neural networks with those learning methods have advantages and disadvantages [33] (Table 2.2).

PROS	CONS
• Widely used in various applications: regression, classification, etc.	• Usually different learning algorithms used in different SLFNs architectures • Some parameters have to be tuned manually • Overfitting • Local minima • Time-consuming

Table 2.2: Advantages and disadvantages of SLFNs with iterative learning methods

Of the five listed in Table 2.2, the main two issues of iterative training methods are local minima and the time consumption (obviously due to their iterative nature). And that is what ELM do better. The new learning theory on which ELM are based is a learning without iterative tuning: if a continuous target function $\boldsymbol{f}(\boldsymbol{x})$ can be approximated by SLFNs with adjustable hidden nodes, then the hidden node parameters (weights between inputs and hidden layer and biases) of such SLFNs need not to be tuned. Indeed, all these hidden node parameters can be randomly generated without the knowledge of the training data. That is, using the notation introduced in paragraph 2.2.1.1, for any continuous target function $\boldsymbol{f}(\boldsymbol{x})$ and any randomly generated set $\{W^1, \boldsymbol{b^1}\}$ it is true that:

$$\lim_{L\to\infty} \|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}_L(\boldsymbol{x})\| = 0 \tag{2.43}$$

holds with probability one if W^2, that is the output weight matrix, is chosen to minimize:

$$\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}_L(\boldsymbol{x})\| \tag{2.44}$$

Expression (2.44) turns out, after some substitutions, to be equal to:

$$\|A^1 W^2 - T\| \tag{2.45}$$

where T is the target matrix and A^1 is the hidden layer output matrix. The least squares solution for (2.45) is:

$$\left\|A^1\widehat{W^2} - T\right\| = \min_{W^2} \|A^{1T} W^2 - T\| \;\rightarrow\; \widehat{W^2} = (A^1)^+ T \tag{2.46}$$

where $(A^1)^+$ is the Moore-Penrose generalized inverse of hidden layer ouput matrix A^1.

The three steps of the new training model are therefore very easy: given a training set $Z = \{(\boldsymbol{x_i}, \boldsymbol{t_i}) |\, \boldsymbol{x_i} \in \mathbb{R}^n,\ \boldsymbol{t_i} \in \mathbb{R}^m, i = 1, \dots, N\}$ where n is the number of inputs and m is the number of outputs

- Generate random hidden node parameters W^1 and $\boldsymbol{b^1}$
- Calculate the hidden layer output matrix A^1
- Calculate the output weight matrix as $W^2 = (A^1)^+ T$

The fundamental consequences of such result are:

- Extreme Learning Machine is a simple tuning-free algorithm.
- The learning speed of ELM is extremely fast.

- The hidden node parameters W^1 and $\boldsymbol{b^1}$ are not only independent of the training data but also of each other.
- Unlike conventional learning methods which must see the training data before generating the hidden node parameters, ELM could generate the hidden node parameters before seeing the training data.
- Unlike traditional gradient-based learning algorithms which only work for differentiable activation functions, ELM works for all nonlinear piecewise continuous activation functions.
- Unlike traditional gradient-based learning algorithms facing several issues like local minima, improper learning rate and overfitting, ELM tends to reach the solution straightforward without such trivial issues.

Moreover, it is possible to keep into account also regularization according to the ridge regression theory in order to weight the values of W^2 and improve the generalization of the model preventing overfitting. The problem now is slightly different from before and the goal is to minimize:

$$\|A^1W^2 - T\|\,,\|W^2\| \tag{2.47}$$

The least squares solution for (2.47) based on KKT conditions (explained in paragraph 2.2.2) can be written as:

$$W^2 = A^{1T}\left(\frac{I}{C} + A^1A^{1T}\right)^{-1} T \tag{2.48}$$

where C is the regularization coefficient and I is an identity matrix of proper dimensions.

Moreover, it is possible to use in this case also the kernel trick. Indeed, one can apply Mercer's conditions for kernels on ELM and define a kernel matrix as follows:

$$\Omega_{ELM} = A^1A^{1T}: \Omega_{ELM_{i,j}} = \boldsymbol{a^1}(\boldsymbol{x_i}) \cdot \boldsymbol{a^1}(\boldsymbol{x_j}) = k(\boldsymbol{x_i}, \boldsymbol{x_j}) \tag{2.49}$$

where $\boldsymbol{a^1}(\boldsymbol{x})$ is the hidden layer output vector for input $\boldsymbol{x}$.

In this way, the output function of the neural network $\boldsymbol{f}(\boldsymbol{x}) = A^1W^2$, inserting (2.48) becomes:

$$\boldsymbol{f}(\boldsymbol{x}) = A^1A^{1T}\left(\frac{I}{C} + A^1A^{1T}\right)^{-1} T = \Omega_{ELM}\left(\frac{I}{C} + \Omega_{ELM}\right)^{-1} T \tag{2.50}$$

The output, so, depends only on the kernel matrix. This is a quite powerful result because it means that the hidden layer output matrix A^1 need not to be known to users; instead, its corresponding kernel (which must satisfy the Mercer's conditions, like for example the polynomial kernel or the Gaussian kernel) is given to users. The number of hidden nodes L need not to be given either. Moreover, the kernel-based ELM learning algorithm achieves similar or better generalization performance, is more stable compared to traditional ELM and it is faster.

Finally, a solution for using ELM with deep neural networks is presented. Indeed, although ELM technique is based on SLFNs it is possible to apply it also to multilayer networks taking advantage of the autoencoders idea for unsupervised learning. So far, it has been described the application of neural networks to supervised learning, in which there were labeled training examples. But if only a set of unlabeled training examples $Z = \{\boldsymbol{x_i} | \boldsymbol{x_i} \in \mathbb{R}^n, i = 1, \dots, N\}$ is known and the targets are, instead, unknown unsupervised learning with neural networks is still possible. An autoencoder SLFN (represented in Figure 2.9 for an input sample), in fact, is an unsupervised learning algorithm that sets the target values to be equal to the inputs (i.e., it fixes $\boldsymbol{t_i} = \boldsymbol{x_i}, i = 1, \dots, N$). The autoencoder tries to learn a function $\boldsymbol{t(x)} = \boldsymbol{\hat{x}} \approx \boldsymbol{x}$. In other words, it is trying to learn an approximation to the identity function. It seems a particularly trivial function to be trying to learn, but by placing constraints on the network, such as by limiting the number of hidden units, it is possible to discover interesting structures about the data (the so called features, which are essential for doing unsupervised learning).

Autoencoders are usually trained with backpropagation but, since they are SLFNs, following the previous reasoning the ELM technique can be applied to obtain an ELM-based autoencoder (ELM-AE). In particular, input data are used as output data and random hidden node parameters W^1 and $\boldsymbol{b^1}$ are chosen to be orthogonal. Orthogonalization, even if not necessary, tends to improve ELM-AE's generalization performance. The problem of computing the output weight matrix is really similar to the one studied for basic ELM, the only differences are the othogonalization of the hidden node parameters and the substitution of the targets with the inputs. The equations, at the end, are:

$$W^{1T} W^1 = I \tag{2.51}$$

$$\boldsymbol{b^{1T}} \boldsymbol{b^1} = I \tag{2.52}$$

$$W^2 = A^{1T} \left(\frac{I}{C} + A^1 A^{1T} \right)^{-1} X \tag{2.53}$$

where $X = [\boldsymbol{x_1}, \dots, \boldsymbol{x_N}]$ is the input matrix.

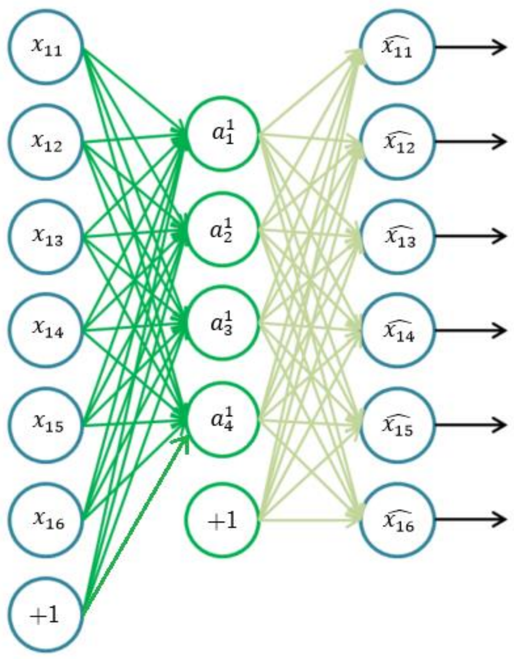


Figure 2.9: Structure of an autoencoder

Autoencoders can be used for unsupervised learning but, as anticipated in paragraph 2.2.1.1, they can be adopted for layer-by-layer unsupervised pre-training approach in deep networks in order to improve the weights initialization and to reduce the risk of vanishing gradients. The idea is that autoencoders can be "stacked" in a layerwise fashion in order to build a stacked autoencoder. Stacked autoencoder is so a neural network consisting of multiple layers of autoencoders in which the outputs of each layer are wired to the inputs of the successive layer. It is not easy to understand but the concept is that each pair of consecutive layers is considered and trained as a single autoencoder. The example described below and represented in Figure 2.10 should clarify possible doubts.

Suppose that the network to train has 6 input neurons, 4 neurons in the first hidden layer, 3 neurons in the second hidden layer and 2 output neurons. The steps to accomplish in order to obtain good parameters for the stacked autoencoder are illustrated in Figure 2.10.

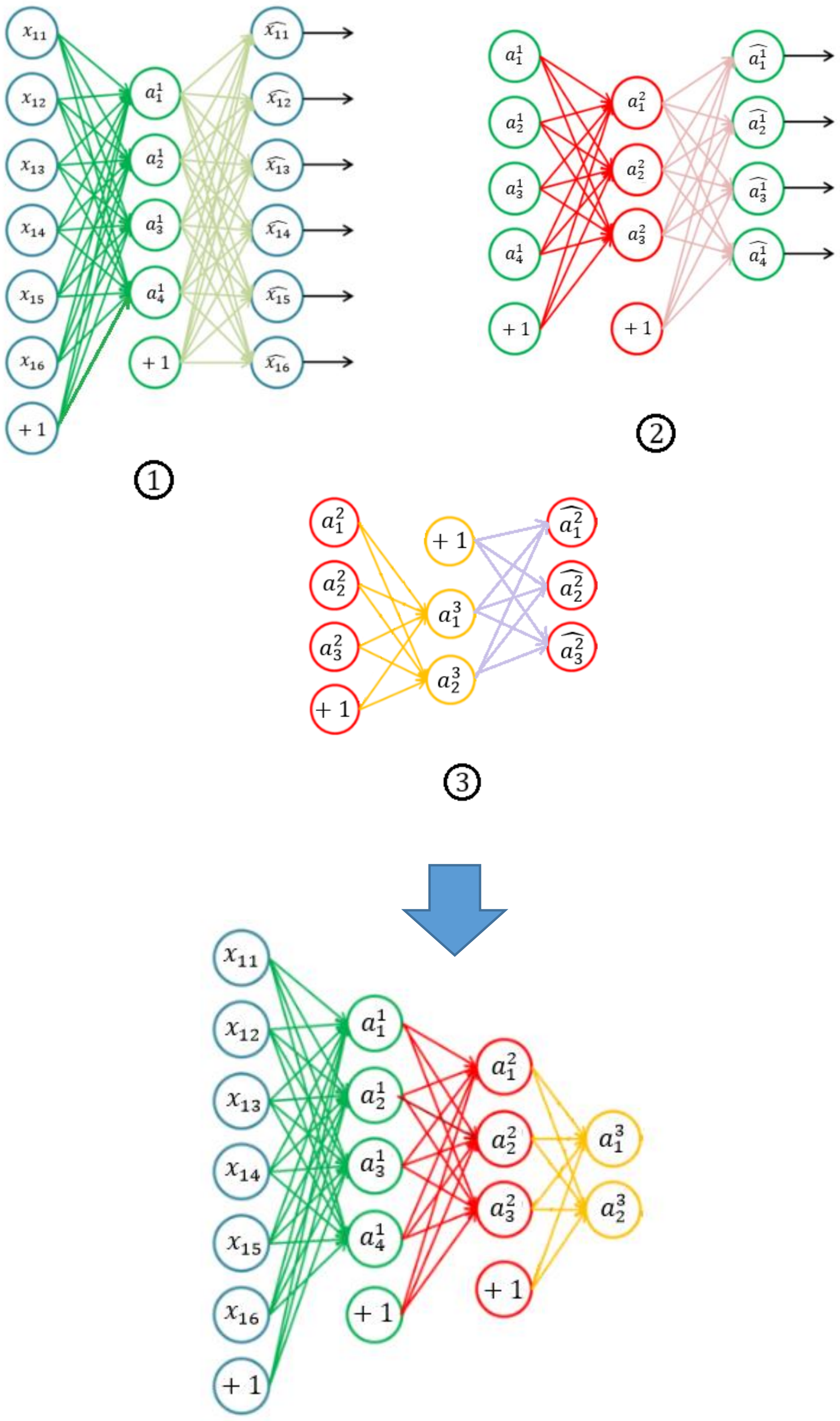


Figure 2.10: Steps for the training of stacked autoencoders

The application of the ELM technique to multilayer networks is based on the same procedure described in Figure 2.10. The only difference is that, instead of considering standard autoencoders and train them with backpropagation, ELM-based autoencoders are taken into account and trained with ELM method. In other words, each pair of consecutive layer is considered and trained as an ELM-AE, choosing random hidden node orthogonal parameters and computing the output weight matrix with equation (2.53). The calculated output weights will be then used to form the stacked autoencoder (Figure 2.11).

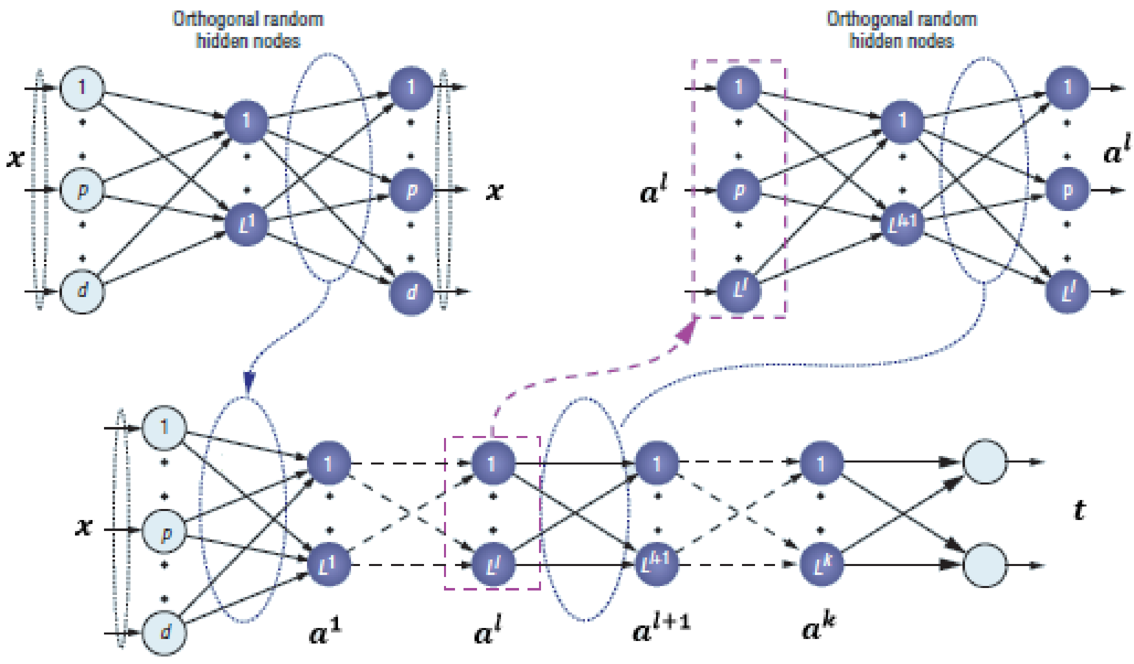

Orthogonal random hidden nodes
x
x
Orthogonal random hidden nodes
a^l
a^l
x
t
a^1
a^l
a^{l+1}
a^k

Figure 2.11: Steps for the training of ELM-based multilayer networks

2.3 Results

In this section, all the techniques seen previously are applied to the problem of space object classification from light curve data. Accuracies and speeds of the various methods are compared and discussed.

The light curve data kindly provided by Prof. Linares include 50000 labeled samples. Just to remind, they are generated by a complex model taking into account objects with different shapes and controls (to be precise, by the Ashikhmin-Shirley (AS) model, whose description, deeply given in [24], is out of the scope of this thesis). Moreover, from the same object different light curves are produced, changing for example the angular velocity of rotation and other parameters. The five different classes, which represent the label of the data, consist of objects with different shapes and controls.

Each sample is composed by 182 inputs (the apparent magnitude of the space object at 182 time instants, two examples of different classes are shown in Figures 2.12-2.13) and one output (the class of the object). It is useful to remind that the five possibile classes are: debris (passive), rocket body (passive), sun pointing payload (active), spin stabilized payload (active) and nadir pointing payload (active).

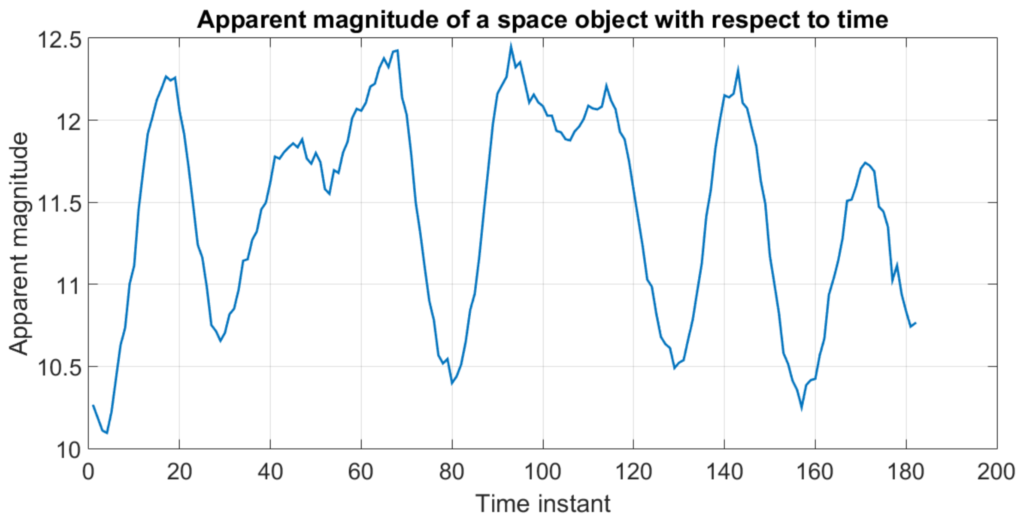


Figure 2.12: Example of the apparent magnitude of a space object with respect to time, class 4 (spin stabilized payload)

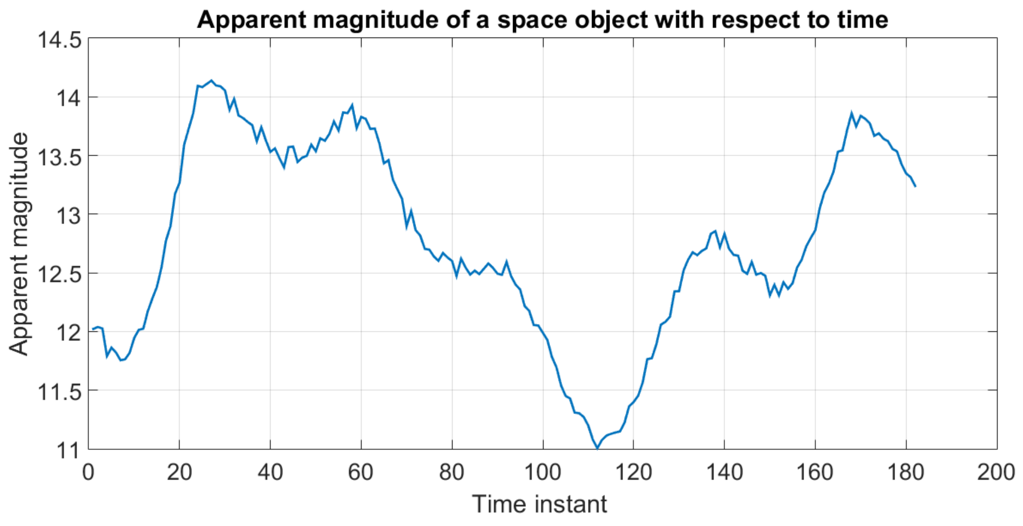


Figure 2.13: Example of the apparent magnitude of a space object with respect to time, class 1 (debris)

As said, the inputs represent apparent magnitude at a given time, so they are numbers ranging from 9 to 18 approximately. Usually, the first thing to do on inputs is to normalize them. Indeed, almost all the classification methods (also neural networks and SVM used later) are really sensitive about the format and the range of the inputs.

Mainly, there exist two kinds of data normalization:

- STANDARD SCORE (or Z-SCORE) NORMALIZATION. The normalization by standard score simply converts the group of data in a frequency distribution such that the mean is zero and the standard deviation is 1. Given $c_i, i = 1, \dots, N$ data, the normalized data by standard score are defined as:

$$\widetilde{c_i} = \frac{c_i - \mu}{\sigma} \tag{2.54}$$

  where μ is the mean:

$$\mu = \frac{1}{N} \sum_i c_i \tag{2.55}$$

  and σ is the standard deviation:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_i (c_i - \mu)^2} \tag{2.56}$$

- MEDIAN NORMALIZATION. This normalization method simply subtract the median from the data. The median of a set of data is the middle value when the values are sorted from the lowest to the highest. If the number of values is even, the median is the average of the two middle values. So, given $c_i, i = 1, \dots, N$ data, the normalized data by median subtraction are defined as:

$$\widetilde{c_i} = c_i - median \tag{2.57}$$

- MIN-MAX NORMALIZATION. In this case the data are scaled from a minimum number (m) to a maximum number (M), decided by the user. Given $c_i, i = 1, \dots, N$ data, the min-max normalized data are defined as:

$$\widetilde{c_i} = m + \frac{c_i - \min\limits_i c_i}{\max\limits_i c_i - \min\limits_i c_i}(M - m) \tag{2.58}$$

The best normalization method depends on the data to be normalized. For the data handled in this section, after some trials, the z-score normalization guarantees the best results. This is why in the following that kind of normalization is implicitly used. The 50000 samples are divided in 43000 for training and 7000 for testing. When some hyperparameters (i.e., learning rate, regularization, etc.) need to be set, the data are divided in 36000 for training, 7000 for validation (and choice of the hyperparameters) and 7000 for testing.

2.3.1 ELM classifier

The first adopted classifier is an Extreme Learning Machine. Obviously, it has 182 input neurons, a certain number of neurons in the hidden layer and 5 output neurons (the number of classes). At the end of the network there is a softmax classifier (see section 2.2) that returns as output the probability distribution over the 5 classes. The picked class is the one with the highest probability. As explained in paragraph 2.2.3, the training procedure for an ELM is really straightforward. The only things to do are:

- Generate random hidden node parameters W^1 and $\boldsymbol{b}^1$
- Set the hyperparameters: number of neurons of the hidden layer, regularization coefficient and activation function
- Compute the ouput weight matrix W^2 with the formula (2.48)

Hidden node parameters W^1 and $\boldsymbol{b}^1$ are simply chosen uniformly randomly distributed between -1 and 1. These initialization is, indeed, proposed by the formal definer of the ELM idea, Huang, and it works well.

The regularization coefficient C is set equal to 0.1 (remind that the smaller C the bigger the regularization effect) in order to try avoiding overfitting and guarantee an acceptable generalization property of the network. The number L of neurons present in the hidden layer and the type of activation function that lead to the best performance are chosen with the validation set. The training and testing accuracies for different values of L and different activation functions are shown in Figures 2.14÷2.19.

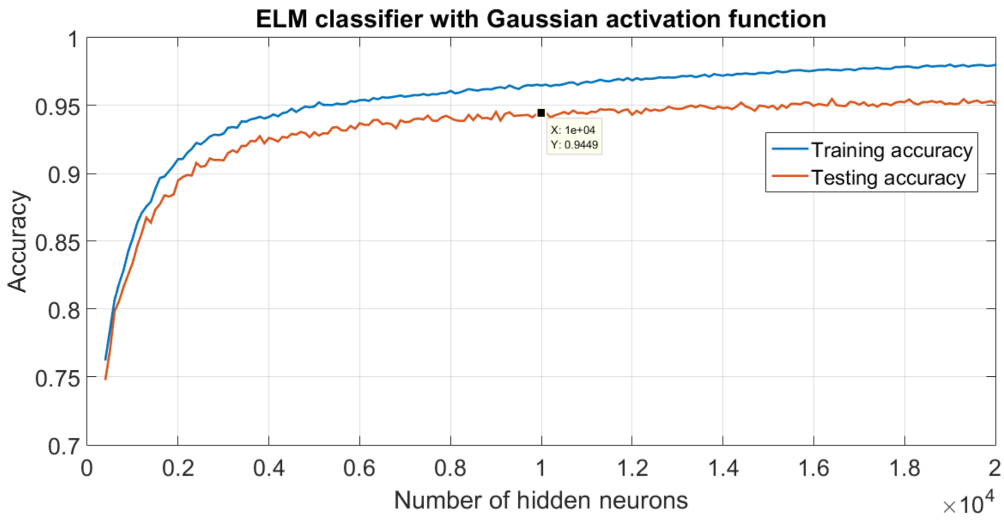


Figure 2.14: Training and testing accuracies of an ELM classifier with Gaussian activation function

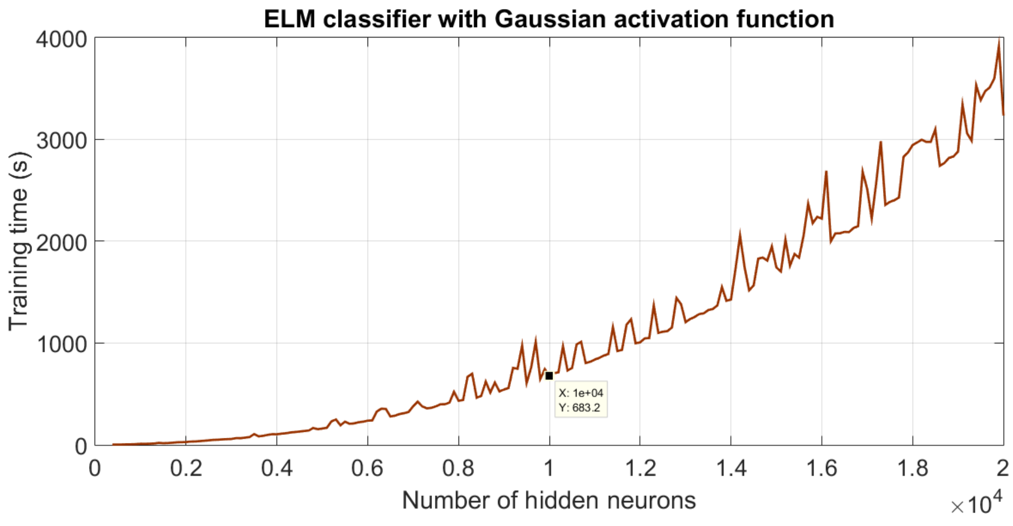


Figure 2.15: Training time of an ELM classifier with Gaussian activation function

For the Gaussian activation function it is chosen as “best point” the one corresponding with 10000 hidden neurons, because after that point the testing accuracy is quite the same but the the training time is increasing almost exponentially. All the tests presented in this chapter are performed on a high performance computer with Xeon Haswell E5-2695 Dual 14-core processors and a processor speed of 2.3 GHz.

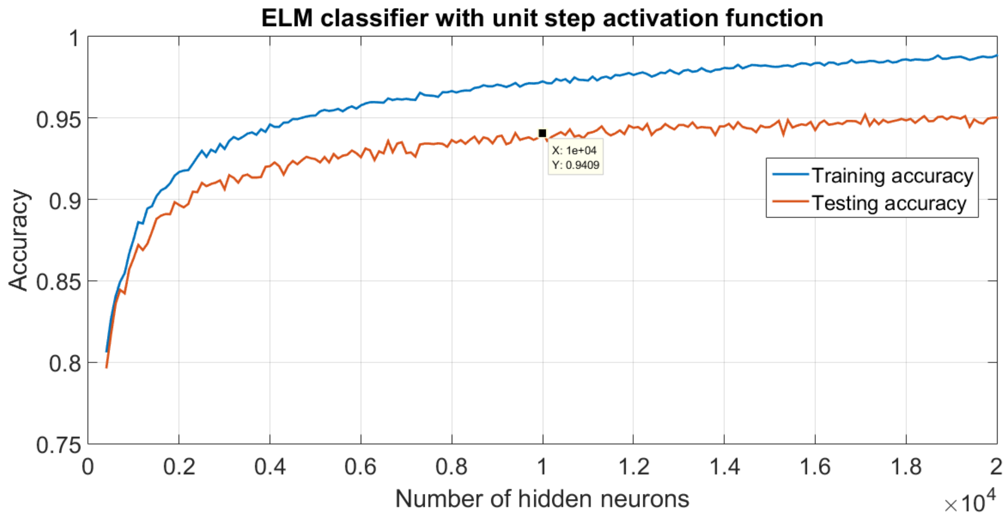


Figure 2.16: Training and testing accuracies of an ELM classifier with unit step activation function

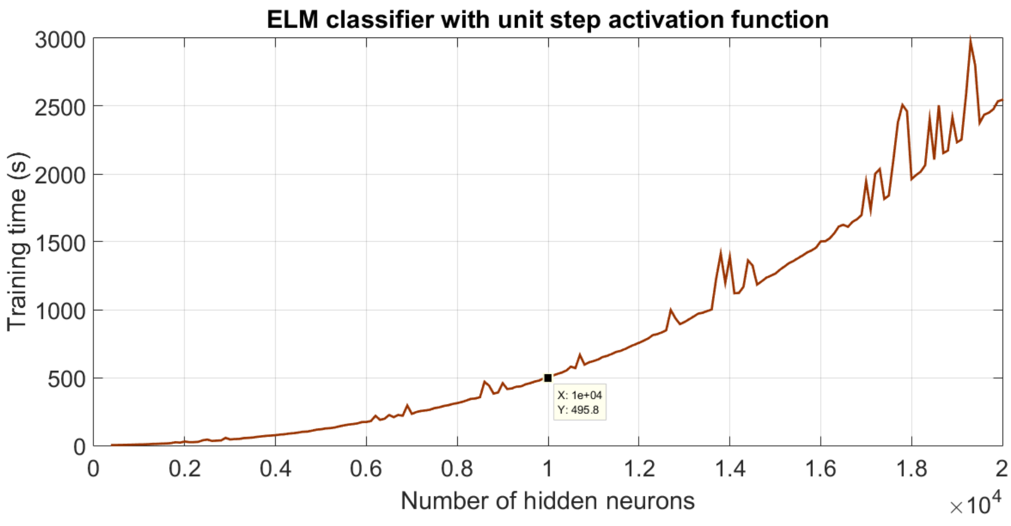


Figure 2.17: Training time of an ELM classifier with unit step activation function

For the unit step activation function it is chosen as “best point” the one corresponding to 10000 hidden neurons for the same reasons as before.

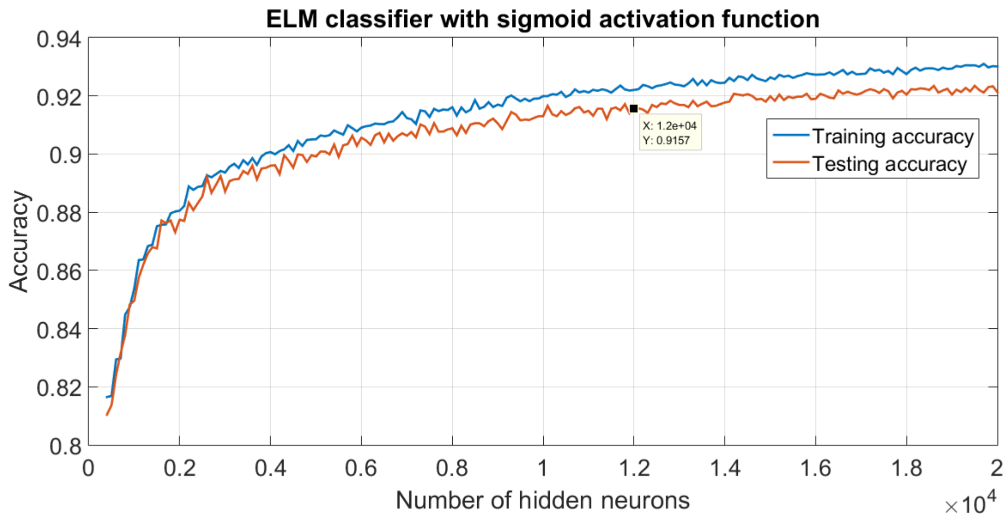


Figure 2.18: Training and testing accuracies of an ELM classifier with sigmoid activation function

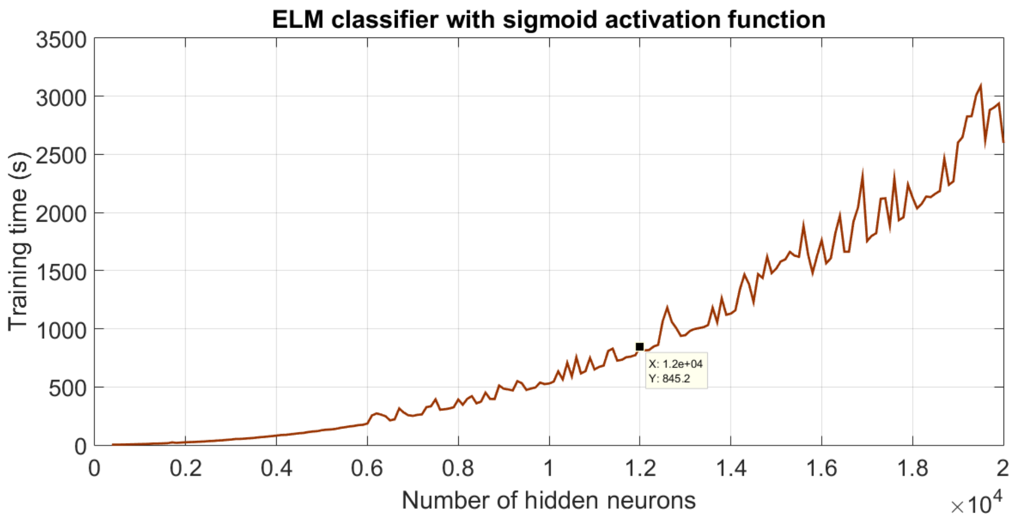


Figure 2.19: Training time of an ELM classifier with sigmoid activation function

For the sigmoid activation function it is chosen as "best point" the one corresponding with 12000 hidden neurons always for the same reasons.

Comparing the results for the three different activation functions, it is possible to notice that the sigmoid brings to the worst outcome both in terms of testing accuracy and training time. The other two (unit step and Gaussian) are very similar but, even if the unit step allows a lower training time, the Gaussian is preferred because the tendency to overfit (visible from the distance between the training accuracy and the testing accuracy) is smaller.

Therefore, the use of a Gaussian activation function and 10000 hidden neurons with the testing data produces the following last results for an ELM classifier:

- Training accuracy: 96.43%
- Testing accuracy: 93.80%
- Training time: 724.12 s
- Testing time: 3.83 s

2.3.2 Multilayer ELM (ML-ELM) classifier

This classifier is the multilayer version of the previous one (ELM classifier). There is not much to say about it, since all the training procedure, based on stacked autoencoder, was clarified in paragraph 2.2.3 and depicted in Figure 2.11. Hidden node parameters are simply chosen uniformly randomly distributed between -1 and 1. At the end of the network there is a softmax classifier.

So, in principle, the treatise is the same as the ELM classifier, but with more than one hidden layers. The validation data for different structures of the network are:

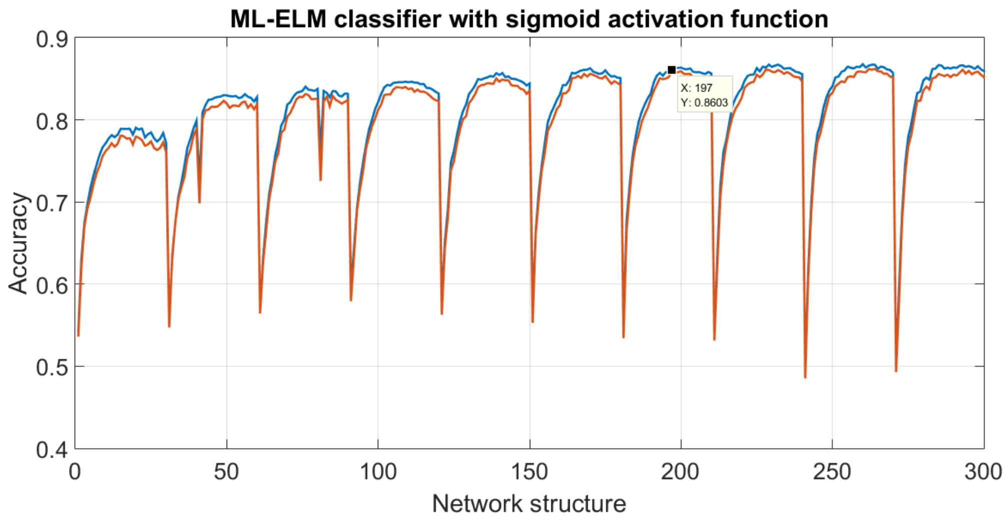


Figure 2.20: Training and testing accuracies of a ML-ELM classifier with sigmoid activation function

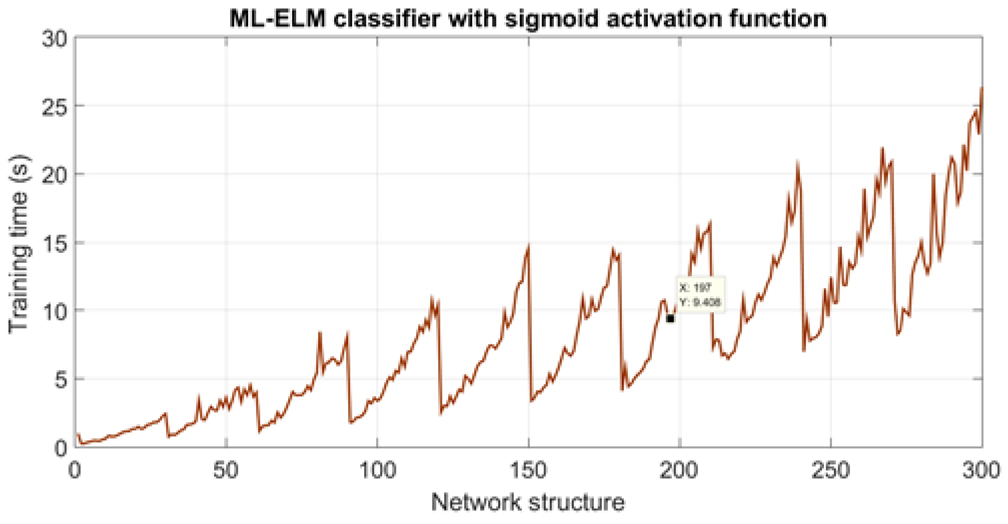


Figure 2.21: Training time of a ML-ELM classifier with sigmoid activation function

Figures 2.20-2.21 show the training and testing accuracies and the training time of a two-layer ML-ELM classifier. Only the sigmoid activation function is presented since in this case it allows similar performance to the other kinds of activation functions.

The result is not surprisingly good: even changing the regularization coefficient and other second order parameters, the accuracy is not significantly increasing and is lower than the single layer case. At least, the training time is really small. The cause of this outcome can maybe be ascribed to the fact that the training data (36000) are not enough to let the autoencoders recognize significant features on the data.

However, as highlighted in Figures 2.20-2.21, the best performance in term of training time and testing accuracy is obtained with 6100 neurons in the first hidden layer and 1700 in the second one. The results with the testing data are:

- Training accuracy: 86.34%
- Testing accuracy: 85.84%
- Training time: 10.72 s
- Testing time: 0.26 s

2.3.3 Kernel-based ELM (K-ELM) classifier

The treatise here is much simpler. Indeed, as underlined in paragraph 2.2.3, neither the activation function nor the number of hidden nodes need to be specified. Only the regularization parameter C (set equal to 0.1 as in the other cases) and the kind of kernel function must be decided. The kernel functions that are taken into account are:

- the linear kernel
- the polynomial kernel
- the Gaussian kernel

The efficiency of those kernels is tested on the validation set (Table 2.3) in order to pick the most satisfying one and try it finally with the testing set.

LINEAR KERNEL	Training accuracy: 48.14% Validation accuracy: 48.26% Training time: 40.19 s Testing time: 0.27 s
POLYNOMIAL KERNEL	Training accuracy: 78.84% Validation accuracy: 75.20% Training time: 51.59 s Testing time: 0.62 s with $c = 0.1$ and $d = 2$
GAUSSIAN KERNEL	Training accuracy: 95.57% Validation accuracy: 94.44% Training time: 64.99 s Testime time: 2.91 s with $\gamma = 1.1$

Table 2.3: Efficiency of different kinds of kernel in K-ELM classifier

As it is clear from Table 2.3, the Gaussian kernel guarantees the best performance. Therefore, using it with $\gamma = 1.1$ on the testing data, the following results for a K-ELM classifier are produced:

- Training accuracy: 95.22%
- Testing accuracy: 93.94%
- Training time: 58.98 s
- Testing time: 2.97 s

2.3.4 Backpropagation and gradient descent (BP+GD) classifier

One of the most famous methods for classification is neural networks trained with backpropagation and gradient descent. All the theory which these algorithms are based on was deeply explained in paragraph 2.2.1 and the implementation is helped by MATLAB built-in functions. The used training algorithm is a resilient backpropagation, whose purpose is to eliminate the vanishing gradient problem. At the end of the network, as usual, there is a softmax classifier that returns as output the probability distribution over the 5 classes.

The values of learning rate, momentum coefficient and weight decay are set to standard values proposed by MATLAB:

- Learning rate: 0.01
- Momentum coefficient: 0.9
- Weight decay: 1.2 when the weight is increasing, 0.5 when it is decreasing

Therefore, the only things to be decided are the number of hidden nodes and the activation function, which are chosen using the validation data.

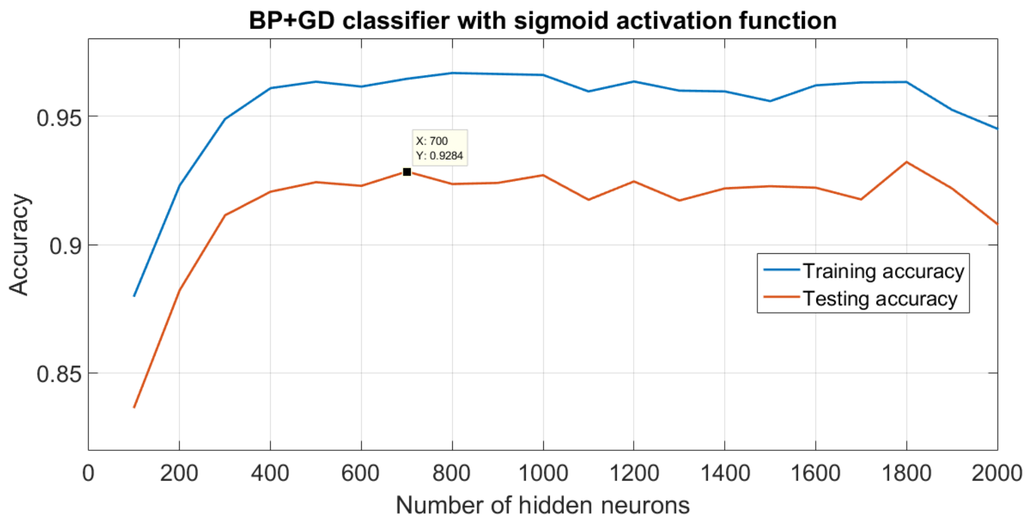


Figure 2.22: Training and testing accuracies of a BP+GD classifier with sigmoid activation function

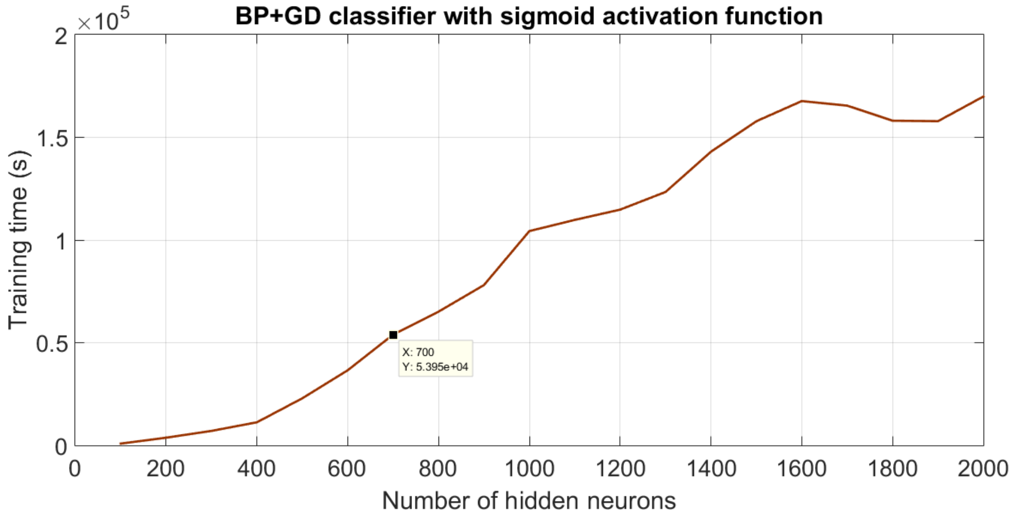


Figure 2.23: Training time of a BP+GD classifier with sigmoid activation function

Figures 2.22-2.23 show the training and testing accuracies and the training time for a BP+GD classifier with a sigmoid activation function. With other types of activation function the outcomes are really similar or even worse. Predictably, the network in this case needs a smaller number of hidden neurons than the ELM case. As depicted in Figure 2.22, the accuracy increases at the beginning, then it stabilizes and at the end it starts decreasing probably because the network is becoming more and more complex and so it can be stuck in local minima.

As a trade-off between testing accuracy and training time, a network with 700 hidden neurons is chosen to be tested with the testing data. The results are:

- Training accuracy: 95.31%
- Testing accuracy: 93.63%
- Training time: 44573.00 s
- Testing time: 18.62 s

Taking a look at the cost function and its gradient during the test (Figures 2.24-2.25), it is possible to understand the working principle of the gradient descent algorithm. Indeed, by repeatedly applying the update rule (2.9), it moves at each step in the direction opposite with respect to the gradient, trying to find a minimum of the cost function. In this case, both the cost function and its gradient are decreasing more and more towards zero, as the expected behavior.

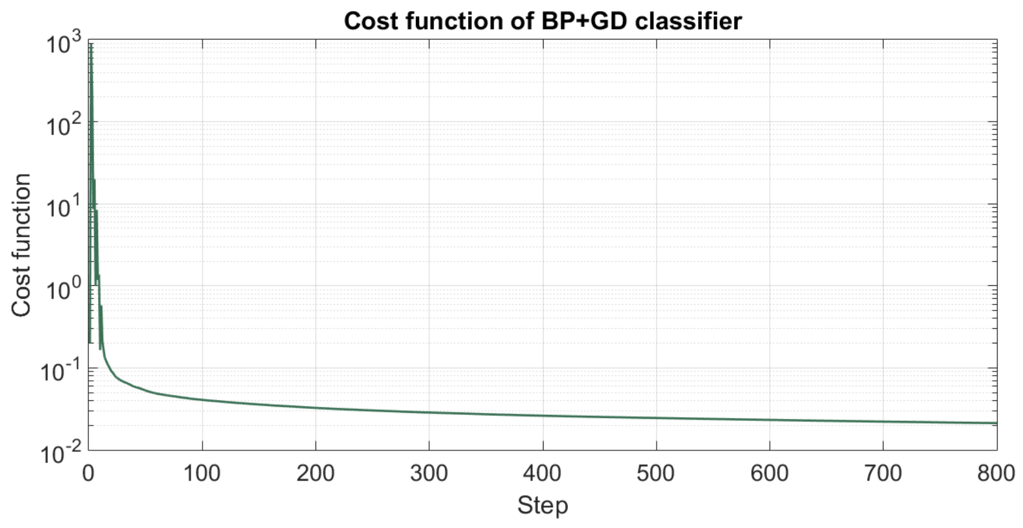


Figure 2.24: Step-by-step cost function of a BP+GD classifier

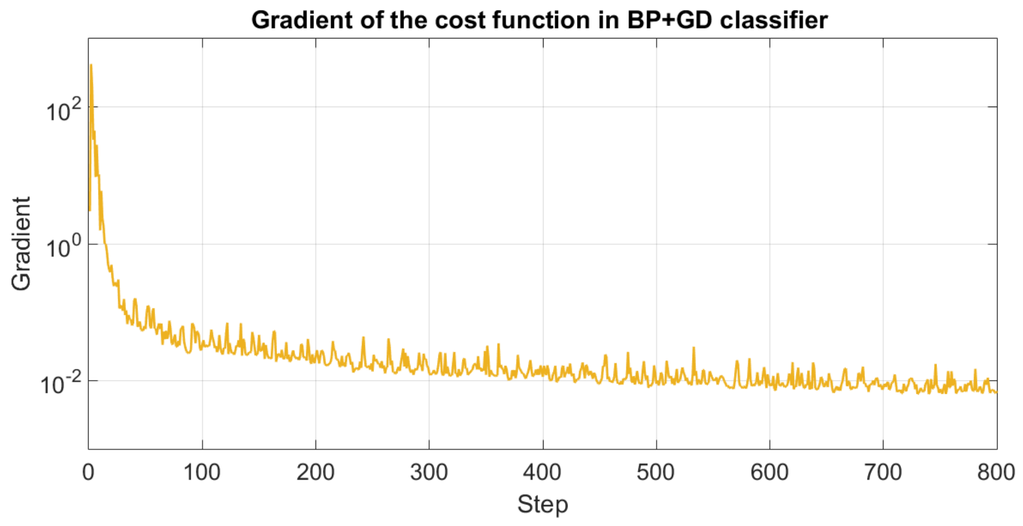


Figure 2.25: Step-by-step gradient of the cost function in the BP+GD classifier

2.3.5 Support Vector Machines (SVM) classifier

SVM are one of the best methods for classification, mainly because of the easiness of the algorithm. However, as anticipated in paragraph 2.2.2, they were originally conceived as binary classifiers and in the problem studied in this chapter there are 5 classes. But nothing is lost, since there exist two possible solutions:

- ONE-VS-ALL BINARY CLASSIFIERS. If the data need to be divided in D classes, the first idea is to build D one-vs-all SVM classifiers. This means that for the ith classifier, the positive examples will be the points in class i, the negative examples will be the points not in class i. In other words, each classifier is trying to separate one class from all the others. After the training of D classifiers, a new sample is categorized according to the maximum of the scores returned by the classifiers (the score is the left hand side of the decision rule (2.20)).

- ONE-VS-ONE BINARY CLASSIFIERS. The second idea is to build $D(D-1)/2$ one-vs-one classifiers. Each classifier tries to learn how to distinguish two different classes i and j. After the training part, a new sample is categorized using a voting strategy: each classifier is consulted and votes for one class. At the end, the class having the majority of votes wins.

Both the two abovementioned solutions work pretty well. However, the one-vs-one method usually works slightly better than the other if the number of classes is small, even if it is more computationally expensive. In this problem there are 5 classes, a small sumber, and that is the reason why a one-vs-one solution is adopted. MATLAB built-in functions are used for the implementation and 10 binary classifiers are trained.

The only parameters to be set are the regularization parameter (put equal to 0.1 as in the other cases) and the kind of kernel function. This latter is chosen with the aid of the validation set (results in Table 2.4).

GAUSSIAN KERNEL	Training accuracy: 98.75% Validation accuracy: 96.00% Training time: 20998.00 s Testing time: 31.89 s
POLYNOMIAL KERNEL	No convergence
LINEAR KERNEL	No convergence

Table 2.4: Efficiency of different kinds of kernel in SVM classifier

Here the choice of the kernel is very easy, since neither the polynomial kernel nor the linear one converge. The non convergence is probably due to the fact that the training data are barely linearly separable with the transformations caused by those kernels. Applying the Gaussian kernel SVM to the testing data produces these final results:

- Training accuracy: 98.75%
- Testing accuracy: 96.34%
- Training time: 21462.00 s
- Testing time: 33.63 s

2.3.6 Results

Summarizing, the results obtained with the different learning techniques are reported in Table 2.5.

ELM CLASSIFIER	Training accuracy: 96.43% Testing accuracy: 93.80% Training time: 724.12 s Testing time: 3.83 s
ML-ELM CLASSIFIER	Training accuracy: 86.34% Testing accuracy: 85.84% Training time: 10.72 s Testing time: 0.26 s
K-ELM CLASSIFIER	Training accuracy: 95.22% Testing accuracy: 93.94% Training time: 58.98 s Testing time: 2.97 s
BP+GD CLASSIFIER	Training accuracy: 95.31% Testing accuracy: 93.63% Training time: 44573.00 s Testing time: 18.62 s
SVM CLASSIFIER	Training accuracy: 98.75% Testing accuracy: 96.34% Training time: 21462.00 s Testing time: 33.63 s

Table 2.5: Summary of the results for the object classification problem with different kinds of classifiers

As it can be seen in Table 2.5, almost all the classifiers, properly tuned, present the same performance capability, except the SVM which is slightly better and the ML-ELM which is slightly worse. Moreover, the results show that the object classification problem can be brilliantly solved with machine learning algorithms.

[24] claims an overall accuracy of 99.60% with Convolutional Neural Networks. This is not unexpected, since CNN are recognised as very powerful neural networks. However, their training is computationally expensive and the required time is surely greater than the one of the algorithms examined in this chapter.

Finally, the most surprising thing is that the ELM learning algorithm is able to guarantee similar testing accuracy with respect to standard learning techniques but with much smaller training time. In particular, the K-ELM classifier is more than 750 times faster than BP+GD classifier and almost 370 times faster than the SVM classifier. This means that the subject is very interesting and certainly in the future it will get more and more attention by the scientific community.

Chapter 3
Sensor Tasking

"I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted".
(Alan Turing, Computer Scientist and Mathematician)

In this chapter the problems of catalogue maintenance and sensor tasking are addressed. As underlined in the introduction, there exists a catalogue that contains all the information about the known objects and this catalogue needs to be updated constantly. However, the network of sensors that provides data for this catalogue is limited, while the number of space objects is increasing more and more. The solution to this task, then, involves solving a large scale resource allocation and management problem. This chapter studies the problem of sensor management for space situational awareness via reinforcement learning, where the goal is to maintain knowledge over a given number of geostationary satellites (particular subset of SOs) using a limited number of sensing platforms.

In the first section the catalogue maintenance issue is introduced and initially addressed; then, in the second part the subject of reinforcement learning (and in particular of Q-learning) is illustrated together with its possible implementation. At the end, in the third section, the solution and results of the sensor tasking problem are presented.

3.1 Catalogue maintenance

It should be clear now that the main idea under the catalogue maintenance area of interest is to keep the catalogue of space objects updated. This is due to the fact that the known information about space objects under surveillance is uncertain, basically because of the uncertainty inherently present in the propagation models (explained in paragraph 1.2.2.3).

The scope of this chapter is to define a way in which one single telescope can track a certain number of geostationary satellites in order to keep the uncertainties associated to their position and velocity under a certain threshold. As profusely reminded in the previous chapters, the dissertation hereafter concerns only geostationary satellites, but obviously it can be valid for all the other bodies.

The first thing to do in order to accomplish the abovementioned task is to check how many and which satellites are visible from the observer place in the moment of the observation. This procedure is necessary at the beginning in order to let the telescope know where the objects to be observed are approximately located. Only after that, it can decide the sequence of the observations.

The personal program which carries out this job is written in Python and reported in Appendix B. The main idea of the code is to propagate the TLE of all the geostationary satellites and check whether they are or not inside the viewing cone from the observer place (in Figure 3.1 example of viewing cone from Tucson (AZ)). Therefore, the function takes as inputs the code of the observatory, the semi-opening angle of the viewing cone, the maximum magnitude seen by the telescope and optionally a specific Julian date and a TLE file. As a consequence, it gives as output the list of visible satellites (with names and positions).

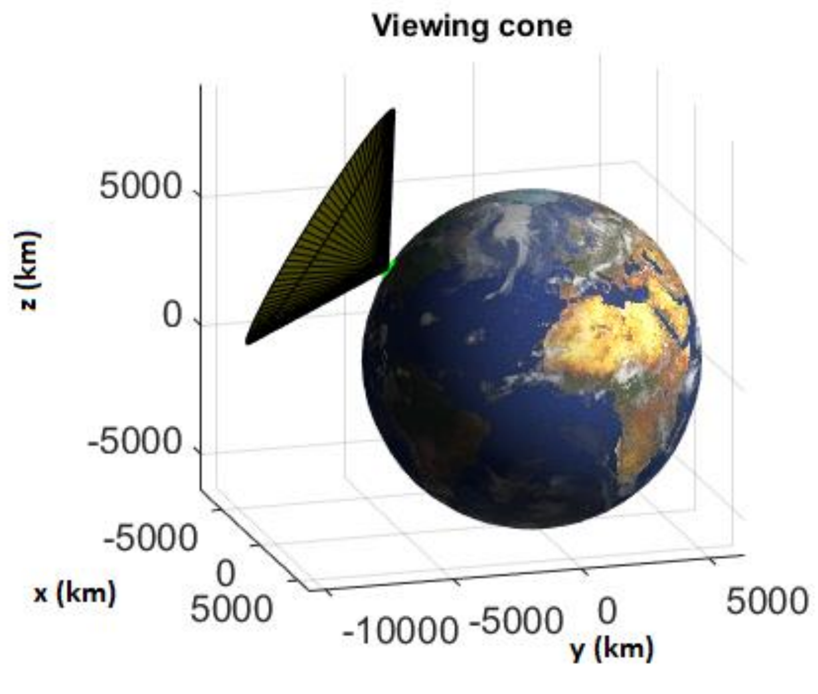


Figure 3.1: Example of possible viewing cone for a telescope located in Tucson (AZ)

At the beginning (lines 88-176) the adopted procedure is the same as the one explained in chapter 1 for TLE propagation. That is, the TLE data are taken from the user file (optional input) or, if it is not provided, from celestrak.org website.

Then (lines 178-228) the code gathers information about the position of the observatory. Starting from its code given as input and thanks to the list supervised by the Minor Planet

Center available at http://www.minorplanetcenter.net/iau/lists/ObsCodes.html, the latitude, longitude and altitude of the observatory are computed (in reality the information on the altitude is really inaccurate and so it is discarded without introducing any significant error). In order to transform the latitude, longitude and altitude in ECEF coordinates which are useful for successive purposes, the georec function of the SpiceyPy Python module is used. This function converts geodetic coordinates (latitude, longitude and altitude) to rectangular coordinates (ECEF), adopting a reference spheroid model specified by the user through the equatorial radius and the flattening coefficient[17].

This paper makes use of the wgs72 model, because it seems to be the most commonly used in the satellite tracking community and is probably the model behind most TLE data. For this choice, the required coefficients are [34]:

$$radius = 6370135\,m$$
$$flattening = 1/298.26$$

In lines 231-323 the TLE are propagated until the given Julian date or, if not present, until the actual one. The procedure is the same as the one described in chapter 1 in the paragraph which talks about target identification. From line 325 on, the interesting and original part of the solution takes place. Indeed, once the position of a satellite at a specific time is known, it is necessary to realize whether it is visible or not from the observer place. The answer to that question is found with the aid of the geometry, checking if the final point is inside the viewing cone (with a given opening angle) from the observatory.

First of all the final position of the satellite is projected onto the axis of the cone, which is the line that passes through the center of the Earth and the observer place. Defined A as a reference point (in this case the center of the Earth), B as the position of the observatory and P as the position of the satellite (all with respect to the ECEF frame), the easiest way to do that is to project vector $\overline{AP}$ onto vector $\overline{AB}$ and then add the resulting vector to point A:

$$\hat{P} = A + \frac{\overline{AP}\cdot\overline{AB}}{\overline{AB}\cdot\overline{AB}}\,\overline{AB} \tag{3.1}$$

In equation (3.1) $\hat{P}$ is the projection of point P, $\overline{AP}$ and $\overline{AB}$ represent the vector respectively from A to P and from A to B and, as usual, $\cdot$ is the dot product. Then, the distance from B to $\hat{P}$ must be computed, which is in practice the height of the cone. Since the points A, B and $\hat{P}$ are aligned, the relationship is simply given by:

[17] Flattening is a measure of the compression of a sphere along a diameter to form an ellipsoid of revolution.

$$distance = \left\|\overline{A\hat{P}}\right\| - \|\overline{AB}\| \tag{3.2}$$

This distance must be obviously greater than 0, otherwise it would mean that the projected point $\hat{P}$ stays inside the segment that connects the center of the Earth with the observatory. But this would make the satellite not visible. Additionally, the dot product $\overline{A\hat{P}} \cdot \overline{AB}$ has to be greater than 0 as well in order to demand that $\hat{P}$ and B are located "in the same half from the center of the Earth", necessary condition for the observability.

After, it is necessary to compare the radius of the cone with the distance between the point P and the axis: if this latter is smaller then the satellite is geographycally visible, otherwise it is not. The distance between the point P and the axis, with simple trigonometry, turns out to be:

$$distance_{radial} = \tan(\alpha\pi/180) * distance \tag{3.3}$$

where α is the semi-opening angle of the cone (in degrees) and $distance$ is defined by equation (3.2).

As a last effort, however, it is necessary to verify that the objects are actually visible as well as geographically visible (lines 355-384). As a matter of fact, a body in the field of view can be too faint to be observed by a particular telescope. Indeed, the maximum magnitude that can be seen by an ideal telescope is equal to:

$$m = 2.7 + 5\log_{10} D \tag{3.4}$$

where D is the aperture diameter in mm.

So, there is the need to introduce the concepts of optical brightness and visual magnitude. The brightness of objects in the visible band is typically described in term of their apparent visual magnitude m_v. The magnitude scale is logarithmic and is defined such that a difference of five magnitudes equal a factor of 100 in brightness, with the result that one magnitude equals a factor of 2.512. Thus, the relative brightness of two objects can be described by their magnitudes:

$$\frac{B_1}{B_2} = 10^{\frac{m_2 - m_1}{2.5}} \tag{3.5}$$

The original zero magnitude was taken to be the star Vega, dimmer objects have positive magnitudes and brighter ones have negative magnitudes. The sun has a magnitude of -26.73, the brightest star in the Earth's night sky (Sirius) is -1.5, the faintest star visible to the eye is about $+6$, and the faintest object detectable by the Hubble Space Telescope is about $+30$.

The visual magnitude of an object in orbit around the Earth can be written as:

$$m_v = -26.7 - 2.5\log_{10}(A\,\rho F(\varphi)) + 5.0\log_{10} r) \tag{3.6}$$

where A is the object's cross-sectional area, ρ is its reflection coefficient, r is its range (in the same unit as A), φ is the object's phase angle and F is a function that depends on the object's shape and orientation. The offset -26.7 is the sun's apparent visual magnitude.

The phase angle φ is the angle between the observer and the sun measured from the space object, so that a phase angle of 0 degrees means that to the observer the space body and the sun are opposite each other in the sky (generally the best circumstance for visual observation) and a phase angle of 180 degrees means that the sun is directly behind the object as seen by the observer. Concerning the function $F(\varphi)$, for the sake of simplicity it is possible to consider the satellite as a diffuse sphere[18] and so:

$$F(\varphi) = \frac{2}{3\pi^2}\left[(\pi - \varphi)\cos\varphi + \sin\varphi\right] \tag{3.7}$$

However, space objects are not perfect reflectors. A reflectivity (or albedo) of 0.1 is typically used in estimating the size of debris objects by optical measurements, although a reflectivity of 0.2 is more typical for intact spacecraft [35].

Applying the theory to the problem in question, A is set equal to $2.4^2\pi\ m^2$, supposing the satellites to be almost circular with an average radius of 2.4 m and ρ equal to 0.2 because satellites are considered as intact bodies.

In order to calculate the phase angle φ, the position of the sun must be known. This is not a simple procedure but can be done computing the ecliptic coordinates[19] of the sun in its apparent motion around the Earth. All the equations presented in the following are from the Astronomical Almanac [36]. At the beginning, it is useful to calculate n, the number of days (positive or negative) since the epoch J2000. Given the Julian date JD, n is equal to:

$$n = JD - 2451545.0 \tag{3.8}$$

The mean longitude of the sun, which is the celestial longitude at which it could be found if its apparent orbit were circular and free of perturbations, is:

[18] Diffuse reflection is the reflection of light from a surface such that an incident ray is reflected at many angles rather than at just one angle as in the case of specular reflection.

[19] The ecliptic coordinate system is a celestial coordinate system commonly used for representing the positions and orbits of Solar System objects.

$$L = 280.460° + 0.9856474°n \tag{3.9}$$

The mean anomaly (defined in the paragraph 1.1.2.1) of the sun is:

$$g = 357.528° + 0.9856003°n \tag{3.10}$$

Both L and g must be reduced to the range [0°,360°). Then, the celestial longitude of the sun is:

$$\lambda = L + 1.915° \sin g + 0.020° \sin(2g) \tag{3.11}$$

Finally, the obliquity of the ecliptic, which means the inclination of Earth's equator with respect to the ecliptic, can be approximated by:

$$\epsilon = 23.439° - 0.0000004°n \tag{3.12}$$

Given all these parameters, the distance of the sun from the Earth, in astronomical units, is:

$$R = 1.00014 - 0.01671 \cos g - 0.00014 \cos(2g) \tag{3.13}$$

and the unit vector pointing to the sun in the ECI frame is:

$$\boldsymbol{s_i} = [\cos\lambda, \cos\epsilon\ \sin\lambda\,, \sin\epsilon\ \sin\lambda] \tag{3.14}$$

The last part of the procedure includes the transformation of the sun unit vector from ECI frame to ECEF frame (in the same way as explained in chapter 1, using the SpiceyPy pxform function) and the multiplication by the distance of the sun from the Earth in order to get the sun vector. Obviously, appropriate changes of measurement units (i.e., from astronomical units to meters, etc.) are necessary. Finally, the phase angle φ, angle between the observer and the sun measured from the satellite, is computed as:

$$\varphi = \frac{\overline{SP}\cdot\overline{BP}}{\|\overline{SP}\|\|\overline{BP}\|} \tag{3.15}$$

where S is the position of the sun.

Therefore, it is possible to obtain the function $F(\varphi)$ with equation (3.7) and then the visual magnitude of the satellite with equation (3.6). Only the satellites with a visual magnitude smaller than the limit given by the user are included in the set of visible objects.

3.2 Reinforcement learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The aspect which makes reinforcement learning different from the other machine learning paradigms (supervised and unsupervised learning, illustrated in section 2.2) is that the learner is not told which actions to take. There is only a numerical reward signal and the agent must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics (trial-and-error search and delayed reward) are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is fundamental for many different areas (Figure 3.2), which include psychology, economics, mathematics, neuroscience, computer science and engineering. In particular, for engineering it is a part of the optimal control theory since it is conceived as a way of generalizing and extending ideas from optimal control to non-traditional control problems. Examples of reinforcement learning problems are: fly stunt manoeuvres in a helicopter, play many different Atari games better than humans, control a power station, manage an investment portfolio, make a humanoid robot walk, etc. [37]

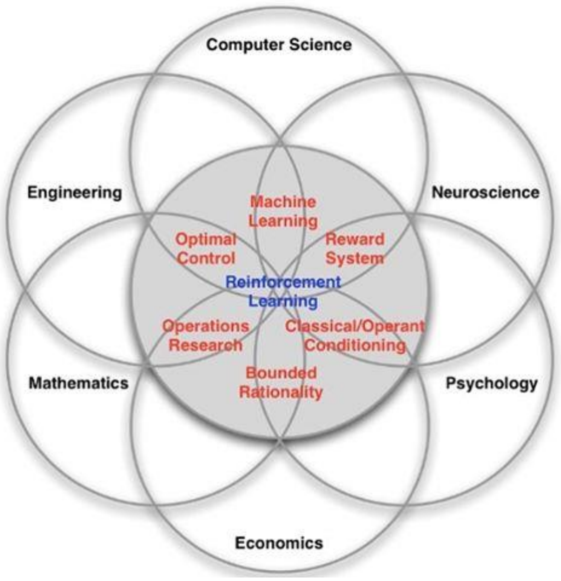


Figure 3.2: Many faces of reinforcement learning

In order to fully understand the subject of reinforcement learning, it is necessary to introduce some basic knowledge:

- REWARD. A reward r_t is a scalar feedback signal which indicates how well the agent is doing at step t. The reinforcement learning is based on the reward hypothesis which states that all possible goals can be described by the maximization of the expected cumulative reward (i.e., the return). This means that the agent's job is to select actions to maximize the expected cumulative reward. As a matter of fact, not all the rewards are instantaneous, there may exist actions with long term consequences and so with delayed rewards. Therefore, the agent should keep in mind that sometimes it is better to sacrifice immediate rewards to gain more long term ones. Examples which give credit to this last statement are financial investments (may take months to mature), helicopter refuelling procedures (might prevent a crash in several hours) and blocks to opponent moves (might help winning later).

- ENVIRONMENT. Everything outside the agent is called the environment. The agent and the environment interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations (or observations) to the agent. More formally, at each step the agent executes the action a_t, receives the observation o_t and the scalar reward r_t, the environment receives the action a_t and emits both the observation o_t and the scalar reward r_t (Figure 3.3).

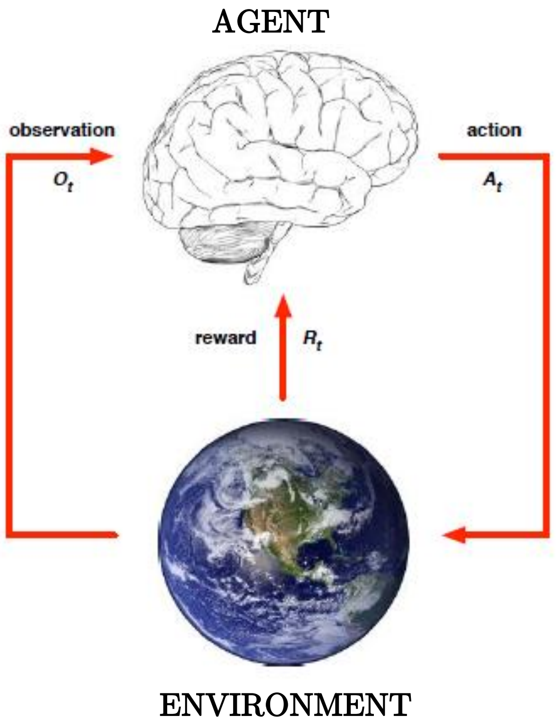


Figure 3.3: Agent-Environment interface

- HISTORY AND STATE. The history h_t is defined as the sequence of observations, actions and rewards up to time t.

$$h_t = o_1, a_1, r_1, \dots, a_t, o_t, r_t \tag{3.16}$$

Basically, all the facts that happen after time t depend on the history (i.e., the agent selects actions and the environment selects observations and rewards).
However, the history can be enormous and so not very manageable. That is the reason why the concept of state is introduced. The state s_t is the information used, separately by the agent and the environment, to determine what happens next. Formally, state is a function of the history:

$$s_t = f(h_t) \tag{3.17}$$

As said, the state is used separately by the agent and the environment. Indeed, there exist a state for the environment and a state for the agent. The environment state s_t^e is its private representation and so whatever it uses to produce the next observation and reward.
The agent state s_t^a, instead, is the agent's internal representation, that is whatever information the agent and the reinforcement learning algorithms use to select the next action. It can be any function of the history:

$$s_t^a = f(h_t) \tag{3.18}$$

Usually, the environment state is not visible to the agent and, even if it is, it may contain irrelevant information. The situation in which the environment state is visible to the agent is called full observability:

$$o_t = s_t^a = s_t^e \tag{3.19}$$

otherwise, when there is some hidden information, it is the case of partial observability. In this last situation, the agent indirectly observes the environment (for example, a robot with camera vision is not told its absolute location) and so the agent state is different from the environment one. Therefore, the agent must construct its own state representation s_t^a.
Additionally, a state can be defined as Markov if it contains all useful information from the history. More formally, a state s_t is Markov if and only if:

$$\mathbb{P}\,[s_{t+1}|s_t] = \mathbb{P}\,[s_{t+1}|s_1, \dots, s_t] \tag{3.20}$$

This means that the future is independent of the past given the present: so, once the state is known, the history may be thrown away because it does not tell anything new.

- POLICY. A policy π defines the agent's way of behaving at a given time. In other words, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine the behavior. The policy can be deterministic:

$$a = \pi(s) \tag{3.21}$$

or stochastic:

$$\pi(a|s) = \mathbb{P}[a_t = a|s_t = s] \tag{3.22}$$

- MODEL. In simple words, a model predicts what the environment will do next. The two functions used for this purpose are the state transition function $\mathcal{P}$, which predicts the next state given the present state and action:

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[s_{t+1} = s' \,|s_t = s, a_t = a] \tag{3.23}$$

and the reward function $\mathcal{R}$, which predicts the next immediate reward, given the present state and action:

$$\mathcal{R}_s^a = \mathbb{E}\,[r_{t+1}\,|s_t = s, a_t = a] \tag{3.24}$$

However, it is not necessary to build a model of the system. Some reinforcement learning algorithms, indeed, are model-free.

- VALUE FUNCTION. Whereas a reward function indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. For this reason, it is used to evaluate the goodness or badness of a state and, therefore, to select actions:

$$v_\pi(s) = \mathbb{E}_\pi\,[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots |s_t = s] \tag{3.25}$$

In the equation (3.25), the term:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.26}$$

is the return, that is the total discounted reward from the time step t. Moreover, $0 \leq \gamma \leq 1$ is the so called discount factor and represents the importance in the present of future rewards. γ close to 0 leads to “myopic” evaluations (i.e., future rewards mean nothing in the present), γ close to 1 leads to “far-sighted” evaluations (i.e., future rewards are as important as present rewards). As underlined in equation (3.25), the value function of a state obviously depends on the policy π, since it defines the sequence of actions to be taken.

Once the basic ideas of reinforcement learning are set, it is useful to introduce the concept of Markov Decision Process (MDP) to understand how reinforcement learning techniques can be applied in practice. Markov decision processes formally describe an environment for reinforcement learning in which all states are Markov and time is divided into stages. In particular, the environment is fully observable (i.e., the current state completely characterises the process). Officially, a Markov decision process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu >$ where:

- $\mathcal{S}$ is a finite set of states, which defines the number of different possible configurations
- $\mathcal{A}$ is a finite set of actions which the agent can select
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'}^{a}$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_{s}^{a}$
- γ is the discount factor, $\gamma \in [0,1]$
- μ is a set of initial probabilities over the states, $\mu_i^0 = \mathbb{P}\,[s_0 = i]$ for all i.

The interesting characteristic of MDPs is that they can formalise almost all reinforcement learning problems.

Associating to a MDP a policy π, which is a distribution over actions given the state, the state transition probability matrix and the reward function become dependent on the policy:

$$\mathcal{P}^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^{a} \tag{3.27}$$

$$\mathcal{R}^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_{s}^{a} \tag{3.28}$$

Moreover, given a policy π it is possible to define the utility of each state (procedure called Policy Evaluation) through the state-value function $v_\pi(s)$ characterized by the equation (3.25). For control purposes, however, rather than the value of each state it is easier to consider the value of each action in each state. The action-value function $q_\pi(s,a)$ is so the expected return starting from the state s, taking action a and then following policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi\,[G_t | s_t = s, a_t = a] \tag{3.29}$$

A fundamental property of $v_\pi(s)$ and $q_\pi(s,a)$ is that they satisfy particular recursive relationships. For any policy π and any state s, the state-value function can be decomposed into the immediate reward plus the discounted value of the successor state:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t = s\right] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s)\big(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\big) \end{aligned} \tag{3.30}$$

Equation (3.30) is the Bellman expectation equation for v_π. It expresses a relationship between the value of a state and the values of its successor states. Figure 3.4 illustrates the path from one state to its possible successor states. Each open circle represents a state and each solid circle represents a state-action pair. Starting from state s, the root node at the top, the agent could take any of some set of actions. From each of these, the environment could respond with one of several next states, s', along with a reward, r. The Bellman equation (3.30) averages over all the possibilities, weighting each by its probability of occurring.

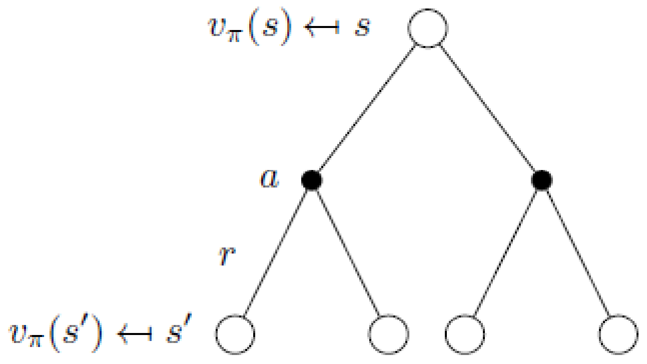
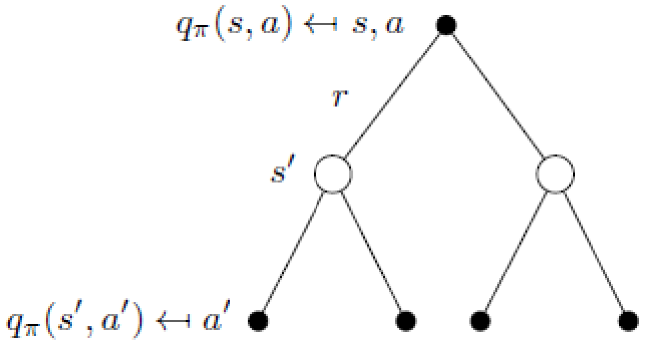

$v_\pi(s) \leftarrow\!\!\dashv s$
a
r
$v_\pi(s') \leftarrow\!\!\dashv s'$

Figure 3.4: Representation of the Bellman expectation equation for v_π

Similarly, with the aid of Figure 3.5, the action-value function can be decomposed as:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s, a_t = a\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right] \\ &= \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \\ &= \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} \pi(a'|s') q_\pi(s', a') \end{aligned} \tag{3.31}$$

Not surprisingly, equation (3.31) is the Bellman expectation equation for q_π.



$q_\pi(s,a) \leftarrow\!\!\dashv s, a$
r
s'
$q_\pi(s',a') \leftarrow\!\!\dashv a'$

Figure 3.5: Representation of the Bellman expectation equation for q_π

Solving a reinforcement learning task means, roughly, finding a policy that tries to maximize the expected reward. For MDPs, it is possible to precisely define an optimal policy due to the fact that the state-value function defines a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words:

$$\pi \geq \pi' \quad if \quad v_\pi(s) \geq v_{\pi'}(s), \forall s \in \mathcal{S} \tag{3.32}$$

Moreover, for any MDP there always exists at least an optimal optimal policy π_* that is better than or equal to all the other policies. Although there may be more than one, they share the same state-value function, called the optimal state-value function v_* and defined as:

$$v_*(s) = \max_\pi v_\pi(s) \tag{3.33}$$

Optimal policies also share the same optimal action-value function q_* defined as:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{3.34}$$

The optimal value function specifies the best possible performance in the MDP. Therefore, only once it is known the MDP can be considered "solved". An important theoretical result states that, for any MDP, there is always a deterministic optimal policy, which for example can be found maximizing over $q_*(s,a)$:

$$\pi_*(a|s) = \begin{cases} 1 & if\ a = arg \max_{a \in \mathcal{A}} q_*(s,a) \\ 0 & otherwise \end{cases} \tag{3.35}$$

Because v_* and q_* are value functions for a policy, they must satisfy the self-consistency conditions given by the Bellman equations (3.30)-(3.31). But because they are the optimal value functions, the consistency conditions can be written in special forms without reference to any specific policy. In particular, the Bellman optimality equation for v_* expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state (Figure 3.6). Formally:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}} q_*(s,a) \\ &= \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \textstyle\sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v_*(s') \right] \end{aligned} \tag{3.36}$$

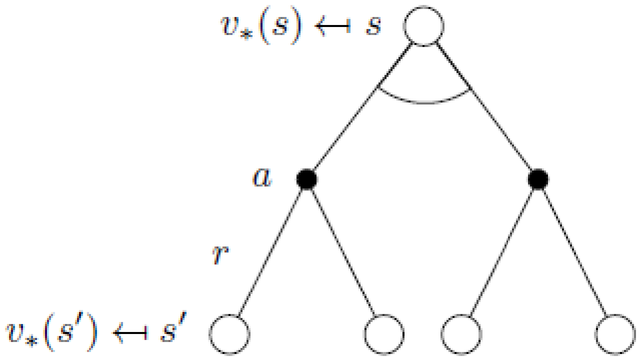


Figure 3.6: Representation of the Bellman optimality equation for v_*

Based on Figure 3.7, the equivalent Bellman optimality equation for q_* is:

$$\begin{aligned} q_*(s,a) &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \\ &= \mathcal{R}_s^a + \gamma \textstyle\sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} q_*(s',a') \end{aligned} \tag{3.37}$$

The Bellman equations (both expectation and optimality) need to be solved. The Bellman expectation equations can be expressed concisely in a matrix form and then solved directly. However, the disadvantage of this solution is the computational complexity: if the number of states in the MDP is high, the problem is unsolvable in reasonable time. That is the reason why approximate approaches like dynamic programming, Monte-Carlo or Temporal Difference methods are used for this purpose.

In this chapter, however, the main goal is to control a single telescope in order that it can track a certain number of geostationary satellites. Therefore, the optimal policy needs to be found and, in order to do that, the Bellman optimality equations for the specific MDP must be solved. Unfortunately, the Bellman optimality equations are nonlinear and so there do not exist closed form solutions for the general case. Therefore, it is necessary to use one of the many existing iterative solution methods (i.e., dynamic programming or reinforcement learning techniques). Since in this case there is no knowledge about the model, the reward function or other characteristics of the system, only reinforcement learning techniques can be used. In particular, this work makes use of the Q-learning method.

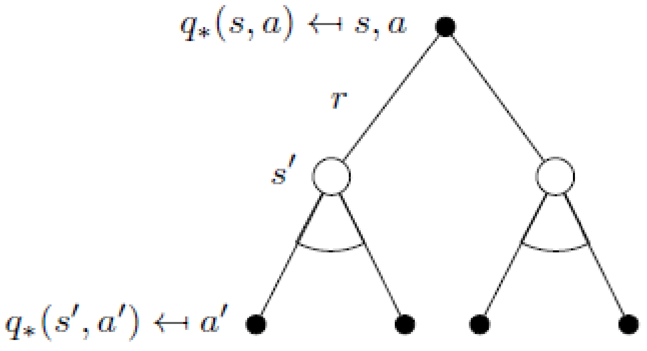

$q_*(s,a) \leftarrow\!\shortmid\, s,a$
r
s'
$q_*(s',a') \leftarrow\!\shortmid\, a'$

Figure 3.7: Representation of the Bellman optimality equation for q_*

3.2.1 Q-learning

Q-learning, introduced by Watkins in 1989, is one type of reinforcement learning algorithm used to calculate state-action values in a iterative way. It falls under the class of temporal difference (TD) algorithms, which suggests that time differences between actions taken and rewards received are involved. For this reason, updates are made after every action taken. Q-learning is important for model-free control, whose purpose is to optimize the value functions of an unknown MDP (and so find v_* and q_*). In other words, Q-learning has to answer to this question: if an agent is put in a new and unknown environment, how can he understand what to do in order to maximize the expected reward?

There are two different kinds of model-free control techniques: on-policy and off-policy. In on-policy methods, the state-action values are iteratively learned at the same time that the policy is improved. In other words, the updates to the state-action values depend on the policy. In contrast, off-policy methods do not depend on the policy to update the action-value function. Q-learning is an off-policy technique. It's advantageous because with off-policy methods it is possible to follow one policy while learning about another. For example with Q-learning, which is the best off-policy algorithm, one could always take completely random actions and yet still learning about the optimal policy.

In particular, going into details of Q-learning, the next action is chosen using the behavior policy μ ($a_{t+1} \sim \mu(\cdot\,|s_t)$), but the the alternative successor action is considered ($a' \sim \pi(\cdot\,|s_t)$, where π is the policy to be learned). The value of $Q(s_t, a_t)$ is then updated towards the value of the alternative action:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t)) \tag{3.38}$$

where $0 < \alpha \leq 1$ is the so-called learning rate and $r_{t+1} + \gamma Q(s_{t+1}, a')$ is referred as the target. Both behavior (μ) and target (π) policies need to be improved. The target policy π is greedy with respect to $Q(s, a)$ (i.e., it always selects the action which maximize the action-value function):

$$\pi(s) = arg \max_{a'} Q(s, a') \tag{3.39}$$

The behavior policy μ, instead, can depend on $Q(s, a)$ or not. A common choice is to make μ ϵ-greedy with respect to $Q(s, a)$ (i.e., it selects the best action the most of the time, but sometimes it takes random actions):

$$\mu(a|s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & if\ a^* = arg \max_{a} Q(s, a) \\ \frac{\epsilon}{m} & otherwise \end{cases} \tag{3.40}$$

In (3.40), m is the number of actions and ϵ is a real number ranging from 0 to 1 included. ϵ-greedy strategy is really interesting because it allows a continual exploration of the environment, without always trusting the information known at that moment. After (3.39)-(3.40), the Q-learning target then simplifies to:

$$r_{t+1} + \gamma Q(s_{t+1}, a') = r_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a') \tag{3.41}$$

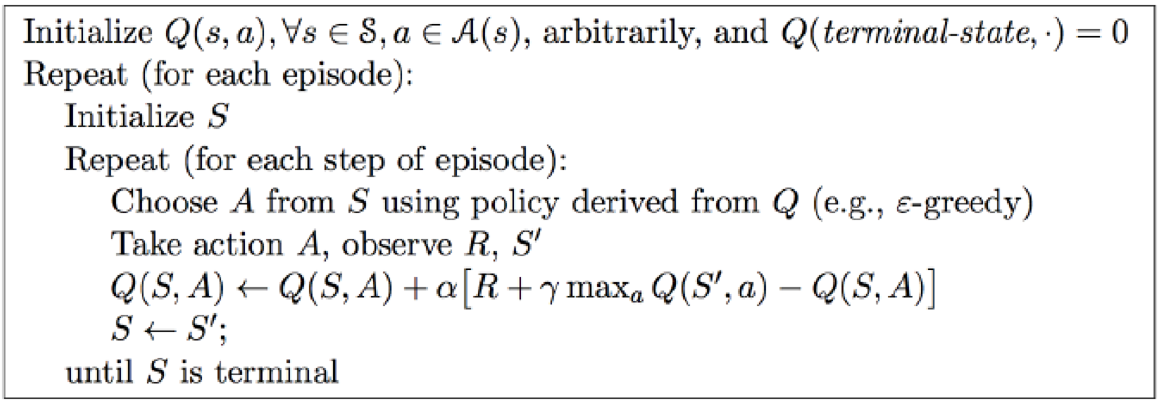
and then the update rule (3.38) becomes:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)) \tag{3.42}$$

The most important theoretical result concerning Q-learning control is the demonstration of its convergence to the optimal action-value function:

$$Q(s, a) \rightarrow q_*(s, a) \tag{3.43}$$

The complete Q-learning algorithm for off-policy control is, finally, reported in Figure 3.8.



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

Figure 3.8: Q-learning algorithm for off-policy control

The update rule (3.42) is tabular, which means that every possible state-action pair is updated and inserted in a lookup table. But in most interesting problems, included the one faced in this chapter, the state-action space is too large to be stored in a table. The function $Q(s,a)$, so, does not have to be just a lookup table. That is where function approximators come in. In this chapter, neural networks (thoroughly described in paragraph 2.2.1) are used to approximate the $Q(s,a)$ function. They accept as inputs a state and an action and give as output the corresponding state-action value. Importantly, unlike a lookup table, a neural network also has some parameters associated with it (weights and biases). Therefore, the Q function finally takes the form $Q(s,a,\theta)$, where θ is the vector of parameters. In this situation, instead of iteratively updating values in a table, it is necessary to update the θ parameters of the neural network (with one of the many learning algorithms available) so that it learns to provide better estimates of state-action values.

Since the network is not a table, the target used to train it is not computed with expression (3.42), but it is simply:

$$target = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \tag{3.44}$$

for the state-action that just happened.

As a final note, it is important to specify that equation (3.44) is true except when s_{t+1} is a terminal state (i.e., the last state in an episode). In that case, the target coincides with the reward:

$$target_{terminal} = r_{t+1} \tag{3.45}$$

3.3 Problem setting and solution

As profusely mentioned, the main goal of this chapter is to address a sensor management problem for space situational awareness via reinforcement learning in a simulated environment. The objective is to maintain knowledge over a given number of geostationary satellites using a single telescope, keeping the uncertainties associated to their position and velocity under a certain threshold. In particular, a Q-learning solution is presented.

3.3.1 Problem statement

First of all, the problem needs to be set and described as a Markov decision process in order to be solved with a reinforcement learning approach. As introduced in section 3.2, a Markov decision process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu >$ where:

- $\mathcal{S}$ is a finite set of states, which defines the number of different possible configurations
- $\mathcal{A}$ is a finite set of actions which the agent can select
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'}^{a}$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_{s}^{a}$
- γ is the discount factor, $\gamma \in [0,1]$
- μ is a set of initial probabilities over the states, $\mu_i^0 = \mathbb{P}\,[s_0 = i]$ for all i.

In this case, the agent is obviously a telescope and the environment is a simulated world. The six elements needed to define a MDP are specified as:

- ACTIONS. The possible actions that the agent can take coincide with the decision of making an observation of a particular geostationary satellite. The observation of a satellite has the consequence of reducing the uncertainties associated with its position and velocity. On the contrary, the uncertainties about not observed satellites are destined to increase more and more.
- REWARD FUNCTION. The reward model development is critical to the performance of the reinforcement learning policy. The one chosen in this work focuses on rewarding the policy's ability to maintain a given catalogue accuracy.
- DISCOUNT FACTOR. The discount factor γ is set to a high value in order to considerably take into consideration future rewards.
- STATES. The Markov decision process requires that all the states $\boldsymbol{s_t}$ are Markov. This assumption implies that the future states depend only on the current state and not on the events that occurred before it. For the problem considered here, the state is the probability density function of the position and the velocity for each satellite. It is assumed that the probability density functions are Gaussian, therefore only the mean and the covariance for each satellite are required. This assumption is really helpful because it allows to describe the state of the MDP given only the mean $\boldsymbol{q_t}$ and the

covariance P_t for each satellite. Additionally, in order to reduce the dimensionality of the state $\boldsymbol{s_t}$, not all the terms of the covariance matrix are considered. This work, indeed, assumes that the covariance matrix can be captured by the diagonal elements (i.e., the variance for each dimension of the satellite state vector). Under these assumptions, the state at time t can be written as:

$$\boldsymbol{s}(t) = [(\boldsymbol{q^1}(t))^T, \dots, (\boldsymbol{q^N}(t))^T, diag\{P^1(t)\}, \dots, diag\{P^N(t)\}]^T \tag{3.46}$$

where N is the total number of satellites.

- INITIAL PROBABILITIES OVER THE STATES. After the definition of the state, at the beginning of the experiment the position and velocity of each geostationary satellite are chosen randomly within a certain range. Moreover, an initial uncertainty about them is defined.
- STATE TRANSITION PROBABILITY MATRIX. The dynamics of the mean and covariance can be studied using the Kalman filter, in particular the version related with discrete-time measurements. Indeed, the system can be represented as a continuous-time model (the position and the velocity evolve continuously) while measurements are taken in a discrete-time fashion. Therefore, the system model and the measurement model are given by:

$$\begin{cases} \dot{\boldsymbol{q}}^i(t) = \boldsymbol{f}(\boldsymbol{q}^i(t)) \\ \boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{q}_k^j) + \boldsymbol{v}_k \end{cases} \qquad \boldsymbol{v}_k \sim \mathcal{N}(0, R_k) \tag{3.47}$$

where $\boldsymbol{f}(\boldsymbol{q}^i(t))$ is a nonlinear dynamics function, $\boldsymbol{q}_k^j = \boldsymbol{q}^j(t_k)$, $\boldsymbol{h}(\boldsymbol{q}_k^j)$ is the nonlinear observation function, i indicates the corresponding satellite, j defines the satellite observed at time t_k and $\boldsymbol{v}_k$ is the measurement noise. In this work the process noise is neglected. Since the system is nonlinear, an Extended Kalman Filter (EKF) is required. The propagation equations of the EKF for the mean and the covariance are:

$$\begin{cases} \dot{\boldsymbol{q}}^i(t) = \boldsymbol{f}\left(\boldsymbol{q}^i(t)\right) \\ \dot{P}^i(t) = F^i(t)P^i(t) + P^i(t)F^i(t)^T \end{cases} \tag{3.48}$$

where $F^i(t)$ is the linearized dynamics matrix of the i-th satellite and is defined as:

$$F^i(t) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}^i(t)} \tag{3.49}$$

With regard to the update equations, once the agent decides to observe one of the satellites (for example, the j-th) its mean and covariance change accordingly to the update equations of the EKF:

$$\begin{cases} \hat{\boldsymbol{q}}^j(t) = \boldsymbol{q}^j(t) + K^j(t)\ \left[\boldsymbol{z}_k - \boldsymbol{h}\left(\boldsymbol{q}^j(t)\right)\right] \\ \hat{P}^j(t) = \left(I - K^j(t)H^j(t)\right)P^j(t) \\ K^j(t) = P^j(t)H^j(t)^T\left[H^j(t)P^j(t)H^j(t)^T + R_k\right]^{-1} \end{cases} \tag{3.50}$$

where I is a proper identity matrix, the symbol ^ indicates the updated value and $H^j(t)$ is the linearized version of the nonlinear observation function:

$$H^j(t) = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\right|_{\boldsymbol{q}^j(t)} \tag{3.51}$$

Obviously, the mean and the covariance of the not observed satellites do not change.

3.3.2 Q-learning application

In the following, a possible application of Q-learning to the problem presented so far is explained. First of all, it is fair to mention that in [4] Linares and Furfaro propose for the same problem a solution based on the actor-critic policy gradient approach. Just to give an idea, the actor-critic is a reinforcement learning technique in which there is a critic that estimates the action-value function and an actor that computes the policy and updates it in the direction suggested by the critic. The results shown in [4] are quite satisfying. The scope of this paragraph and the following one is, therefore, to investigate whether a solution based on Q-learning is possible or not and, later, to compare the potential results.

Why Q-learning? First of all, because it is important to study the situation under different points of view and find different solutions to be compared. Secondly, lately Q-learning is becoming very popular, mainly because of its simplicity. Indeed, while actor-critic methods need to maintain two sets of parameters (one for the action-value function and one for the policy) and so two different approximators, Q-learning needs only one. The implementation then is really straightforward and the only equation to keep in mind is the one reported in (3.42) or the equivalent (3.44)-(3.45) if an approximator is used.

Moreover, Google DeepMind brought Q-learning to the fore with a paper published in Nature[20] on February 26, 2015 [39]. Citing the Google DeepMind website [40], the article describes a deep reinforcement learning system which combines deep neural networks with reinforcement learning at scale for the first time, and is able to master a diverse range of Atari 2600 games to superhuman level with only the raw pixels and score as inputs. For artificial agents to be considered truly intelligent they should excel at a wide variety of tasks that are considered challenging for humans. Until that point, it had only been possible to create individual algorithms capable of mastering a single specific domain. Google DeepMind showed that a novel reinforcement learning agent was able to surpass the overall performance of a professional human reference player and all previous agents across a diverse range of 49 game scenarios. The article [39] represents the first demonstration of a general-purpose agent that is able to continually adapt its behavior without any human intervention, a major technical step forward in the quest for general artificial intelligence.

Therefore, recently, Q-learning, mainly associated with powerful deep convolutional networks, has got a lot of attention in control problems (especially in vision-based control and autonomous driving). This chapter then performs also the task of presenting an adaptable implementation of the Q-learning technique for control purposes.

The version of Q-learning implemented by Google DeepMind, called Deep Q-Network (DQN), and applied also in this chapter, is successful because it introduces two main improvements regarding the updates. The updating part of the algorithm is shown in Figure 3.9.

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

- Using variant of stochastic gradient descent

Figure 3.9: Update algorithm of Deep Q-Network

[20] Nature is a weekly international journal publishing the finest peer-reviewed research in all fields of science and technology on the basis of its originality, importance, interdisciplinary interest, timeless, accessibility, elegance and surprising conclusions [38].

As it can be read in Figure 3.9, the two improvements that allow surprising results are:

- EXPERIENCE REPLAY. Experience replay consists in the storing of all the transitions state-action-reward-next state $(s_t, a_t, r_{t+1}, s_{t+1})$ encountered during the training in a memory. At each step, when the neural network parameters need to be updated with stochastic gradient descent, a random mini-batch of transitions (e.g., 32 of millions stored) is picked from the memory. This is very helpful because it decorrelates the trajectories, since it chooses random transitions and not the most recent ones.
- FIXED Q-TARGETS. The targets for the update of the neural network are computed with respect to old parameters which are kept fixed for some time (e.g., 50 episodes) and then changed to the new values. This strategy helps the stabilization of the neural network.

Before applying Q-learning to the main scenario, it is useful to start with a simpler problem. This problem is not very much related to the main one, but it is an example of Q-learning application to guidance. Moreover, it should help to understand the operation of Q-learning and to test its efficiency.

Suppose that there is a spacecraft (or a general object) which needs to be guided to a specific point. This is a typical guidance problem and it can be solved with Q-learning. In order not to complicate the scenario, suppose that the spacecraft can move only in one dimension (i.e., it can go right, left or remain still) and that it is equipped with two thrusters which allow only three possible accelerations (the measurement unit is not important since it is a simulated environment): -1 (i.e., 1 towards left), 0, 1. The destination point is chosen to be the origin of the one-dimensional reference system. Q-learning is applied as specified in Figure 3.9, with experience replay, fixed targets, deep neural network and stochastic gradient descent. At each step, the spacecraft is rewarded with a value that coincides with the distance from the desired final point. At the beginning, a certain number of random episodes is run without updating the network, in order to collect transitions and fill the memory. Then the learning part starts. In each episode, the spacecraft is initialized with a random position and a velocity of +1 or -1. The results in term of accuracy (i.e., the percentage of episodes in which the spacecraft reaches the desired point) during the training are reported in Figure 3.10.

As shown in Figure 3.10, the system is really fast in learning: after 100 episodes it is already capable of bringing the spacecraft to the final position almost every time. The average accuracy using the neural network trained after 5000 episodes is 100%. This is a perfect outcome, but it is not surprising because the problem is really easy and the agent has only to decide if going right or left. The scenario can be easily modified in order to be more interesting: suppose that the goal now is to guide the spacecraft towards a desired point with a final zero velocity. This is a more realistic guidance problem and Q-learning performs quite good (Figure 3.11). In this case the reward takes into account both the positional displacement and the velocity error. As it can be seen, the learning procedure is slower as it takes 1880 episodes to be almost stationary. The average accuracy using the neural network trained after 10000 episodes is 85.8%. It is a good result but obviously it can be improved finding better hyperparameters and increasing the number of episodes.

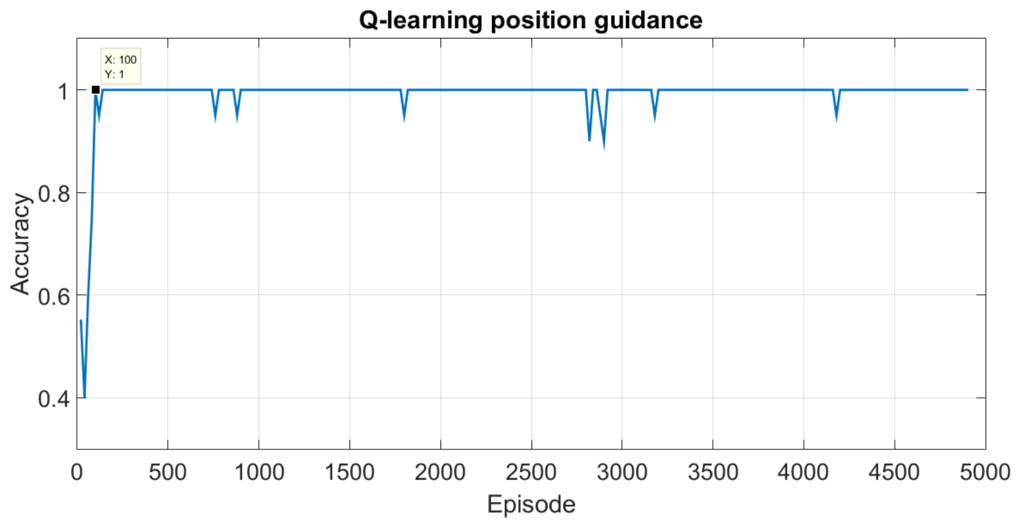


Figure 3.10: Accuracy of position guidance with Q-learning

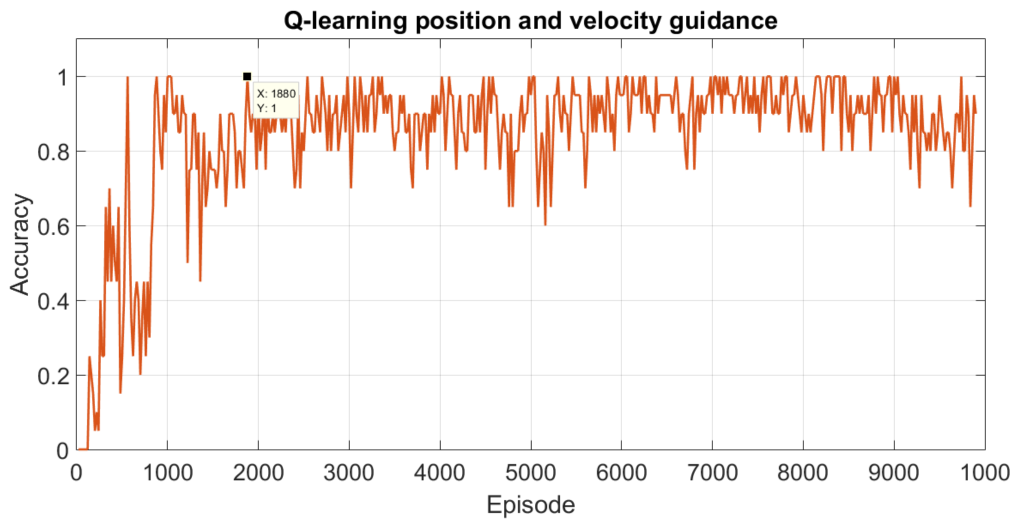


Figure 3.11: Accuracy of position and velocity guidance with Q-learning

The previous examples show that Q-learning is very efficient and reliable. Thus, the time has come to apply it to the main problem of this chapter: the sensor tasking.

3.3.3 Simulation models and results

In the following, the numerical simulation for satellites tracking and the corresponding Q-learning implementation are discussed. To show the effectiveness of the proposed ideas, a population of geostationary satellites is considered for the training. For the sake of simplicity, only two-dimensional motion is studied. The planar equations of motion for a geostationary object assuming only two body forces are given by (the dependency on time is implicit from now on):

$$\begin{cases} \ddot{x} + \frac{\mu x}{d^3} = 0 \\ \ddot{y} + \frac{\mu y}{d^3} = 0 \end{cases} \tag{3.52}$$

where μ is the Earth's gravitational constant expressed in km^3/s^2 (equal to $398600.436\, km^3/s^2$), $d = \|\boldsymbol{d}\|$ and the position vector is given by $\boldsymbol{d} = [x,\ y]^T$ and assumed to be in inertial coordinates. With regard to the observation model, this work assumes to measure the angle β of the satellite observation. For this reason, the observation model is described as:

$$\beta = \text{atan2}(u_y, u_x) \tag{3.53}$$

where $\boldsymbol{u} = \left[u_x,\ u_y\right]^T$ denotes the position of the satellite relative to the observer in inertial coordinates. The observer location is denoted by $\boldsymbol{d_{obs}}$ and the relative position is given by $\boldsymbol{u} = \boldsymbol{d} - \boldsymbol{d_{obs}}$. It is assumed that the observations are corrupted with a zero-mean white noise process with variance equal to $\sigma_\beta^2 = 0.0085\, deg^2$.

Just to remind, in this case the states consist of the mean and variance of the position and velocity for each satellite: $\boldsymbol{s}(t) = [(\boldsymbol{q}^1)^T, \dots, (\boldsymbol{q}^N)^T, diag\{P^1\}, \dots, diag\{P^N\}]^T$. In particular:

$$\left(\boldsymbol{q}^i\right)^T = [x^i,\ y^i,\ \dot{x}^i,\ \dot{y}^i] \tag{3.54}$$

and

$$diag\{P^i\} = \left[var\left(x^i\right),\ var\left(y^i\right),\ var\left(\dot{x}^i\right),\ var\left(\dot{y}^i\right)\right] \tag{3.55}$$

Then, the system model and the measurement model theoretically defined in (3.47) can be written as (the dependency on the specific satellite is not reported):

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ -\frac{\mu x}{\sqrt{x^2+y^2}} \\ -\frac{\mu y}{\sqrt{x^2+y^2}} \end{bmatrix} = \boldsymbol{f}(\boldsymbol{q}) \tag{3.56}$$

$$\boldsymbol{z} = \beta = atan2(x - x_{obs}, y - y_{obs}) + v = \boldsymbol{h}(\boldsymbol{q}) + v \qquad v \sim \mathcal{N}(0,0.0085) \tag{3.57}$$

Thus, from (3.56) and (3.57) it is possible to obtain the linearized dynamics which are used in the EKF propagation and update parts. The matrices are:

$$F = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\bigg|_{\boldsymbol{q}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ p & b & 0 & 0 \\ b & c & 0 & 0 \end{bmatrix}$$

$$p = \frac{3\mu x^2\sqrt{x^2+y^2} - \mu\left(\sqrt{x^2+y^2}\right)^3}{(x^2+y^2)^3}$$

$$b = \frac{3\mu xy\sqrt{x^2+y^2}}{(x^2+y^2)^3}$$

$$c = \frac{3\mu y^2\sqrt{x^2+y^2} - \mu\left(\sqrt{x^2+y^2}\right)^3}{(x^2+y^2)^3} \tag{3.58}$$

$$H = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\bigg|_{\boldsymbol{q}} = [m \quad n \quad 0 \quad 0]$$

$$m = -\frac{y - y_{obs}}{(x - x_{obs})^2 + (y - y_{obs})^2}$$

$$n = \frac{x - x_{obs}}{(x - x_{obs})^2 + (y - y_{obs})^2} \tag{3.59}$$

The satellites used for the simulation are sampled randomly using orbital elements given by a, e, ω and M, which are the semi-major axis, eccentricity, argument of periapsis and mean anomaly, respectively (see paragraph 1.1.2.1). Since only the planar motion is involved, just four orbital elements are needed. Then, the initial orbital elements state vector is given by $\boldsymbol{oe} = [a,\ e,\ \omega,\ M]^T$. Random initial elements are sampled from a normal distribution for a, ω and M, and uniform distribution for e. The parameters for the normal distributions are: $\mu_a = 42164\,km$, $\sigma_a = 100\,km$, $\mu_\omega = 0°$, $\sigma_\omega = 10°$, $\mu_M = 0°$ and $\sigma_M = 10°$. Here μ and σ denote mean and standard deviation, respectively. The eccentricity is sampled from a uniform distribution over the range $e \in [0.01, 0.02]$. These choices guarantee that the objects are visible to the observer, which is assumed to be initially in position $\boldsymbol{d_{obs}} = [6378.1363\,km, 0\,km]$ (on Earth's surface), and then to rotate with the Earth. A possible initial situation for 3 objects is depicted in Figure 3.12, where the green cross is the position of the observer, the red crosses represent the satellites and the circle is the Earth.

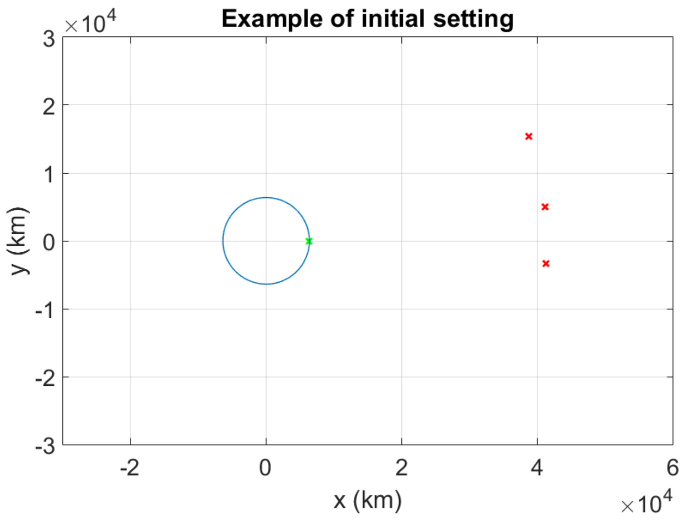


Figure 3.12: Example of initial setting for the sensor tasking problem

The initial orbital elements for each satellite are so converted into an initial position and velocity, which are propagated with equation (3.48) and eventually updated with equation (3.50). At the beginning, the uncertainties are set to 100 km in position coordinates and 1 km/s in velocity. The measurements are supposed to be taken every 50 seconds, which is reasonable for telescopes observing geostationary satellites. Finally, the episodes are run for almost 17 hours.

The Q-learning, then, is implemented as it was described in paragraph 3.3.2. The deep neural network has 3 hidden layers with, respectively, 200, 100 and 50 neurons. As input of the network there is the state $\boldsymbol{s}$ of the system. The number of output neurons is equal to the number of satellites (each output tells the value associated with the observation of a particular satellite given the state). The reward function used for this work is given by:

$$r(\boldsymbol{q}) = \begin{cases} -1 & if\ \sigma > \bar{\sigma} \\ +1 & if\ \sigma < \bar{\sigma} \end{cases} \tag{3.60}$$

where $\sigma = \max_i \left(\frac{1}{2}\text{trace}\left(P^i\right)\right)$ and $\bar{\sigma}$ is defined as the catalogue accuracy threshold, which in this work is set to 30 km.

In order to update the parameters of the deep neural network a stochastic gradient descent algorithm is used. For its implementation, the free MATLAB-based framework LightNet[21] is employed. The choices for the hyperparameters are:

- Learning rate: 0.01
- Weight decay: 0.001
- Gradient clipping: 5
- Momentum coefficient: 0.9

The performance of Q-learning applied to the sensor tasking problem with 3 satellites is shown in Figures (3.13)÷(3.15).

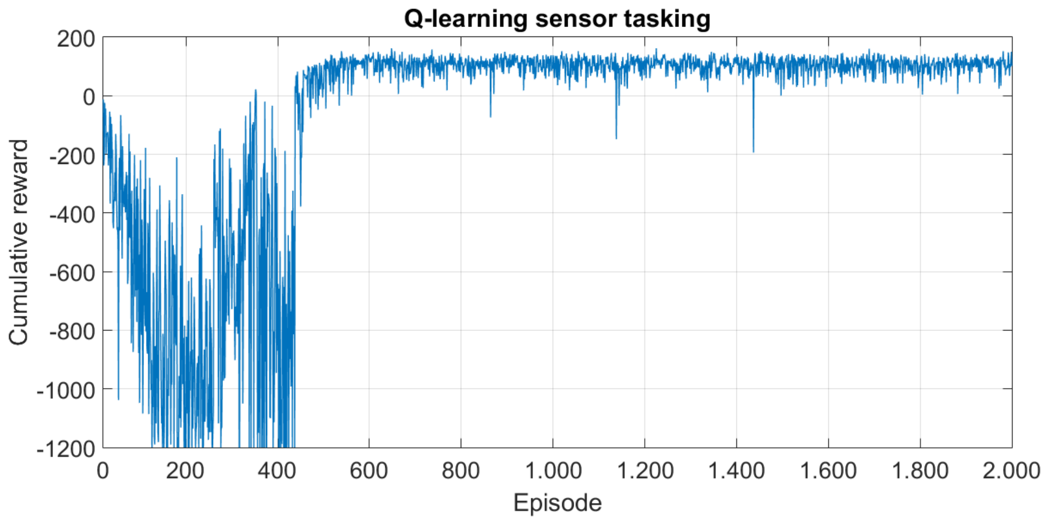


Figure 3.13: Cumulative reward of the Q-learning sensor tasking with 3 satellites

[21] LightNet is available and downloadable from GitHub at https://github.com/yechengxi/LightNet

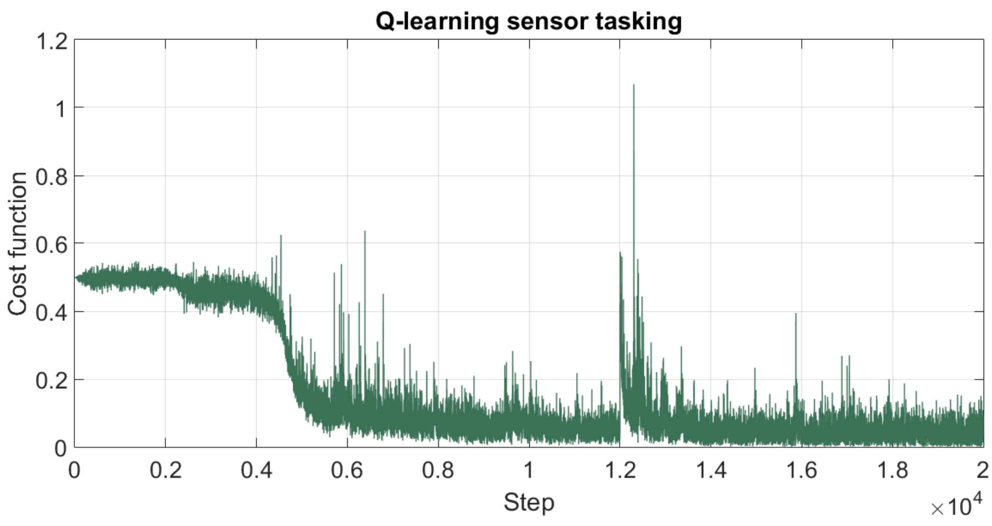


Figure 3.14: Detail of the cost function of the Q-learning sensor tasking with 3 satellites

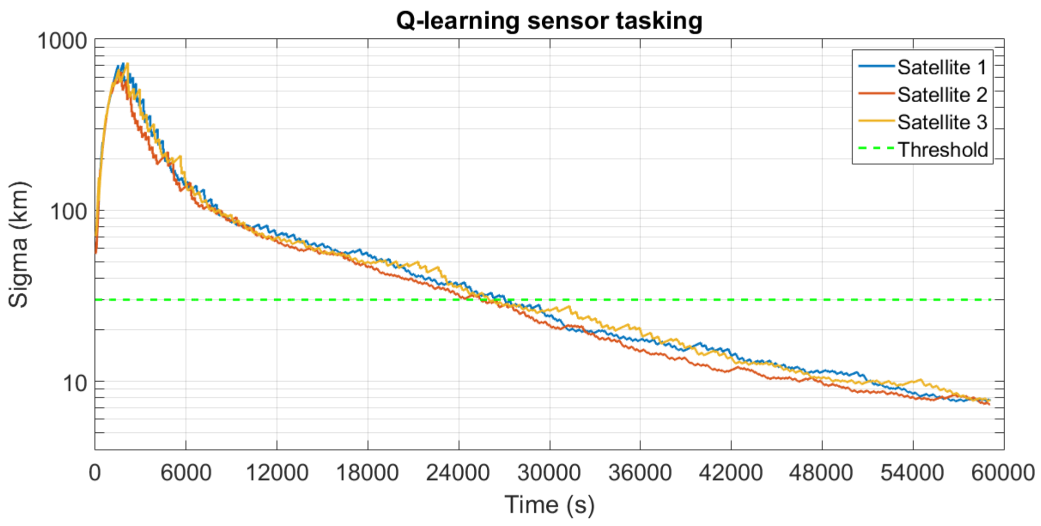


Figure 3.15: Evolution of the uncertainty with respect to time in Q-learning sensor tasking with 3 satellites

As illustrated in Figures (3.13)÷(3.15), the Q-learning is working properly: the cumulative reward is increasing during the training phase, the cost function is approaching zero (the spikes are due to the change in the Q-targets) and the application of the trained neural network after 2000 episodes to new data produces the result shown in Figure (3.15). The system is able to reduce the uncertainty σ^i on each satellite under the defined threshold and then maintain an almost steady state value. Moreover, the behavior is expected to enhance if a much greater number of episodes is considered.

It should be possible to apply this technique to more complex scenarios, for example one with more satellites or one with different kinds of satellites and not only geostationary ones. The outcomes should be satisfactory as well.

Comparing the results with the ones presented in [4] with an actor-critic method, it is clear that Q-learning is slower and shows worse convergence properties because of its off-policy nature. However, the implementation is very simple and its performance is still good.

Chapter 4
Conclusions and future works

This thesis studied and developed techniques in order to realize intelligent systems to observe, identify and classify space objects. In particular, it was focused on geostationary and low orbit satellites, but the discussion could be expanded to every kind of space objects, obviously subject to appropriate precautions. The project lied within the context of Space Situational Awareness (SSA), which refers to the ability to view, understand and predict the physical location of natural and manmade objects in orbit around the Earth, with the objective of increasing knowledge and avoiding collisions. The work is particularly important since intelligent systems for this purpose have never been made. However, since the idea regards a very wide subject, this thesis focused only on 3 main areas: the astrometry and photometry pipeline, the object classification and the sensor tasking problem.

In the first chapter, the task of extracting interesting information from telescope images was faced. In particular, a pipeline should be able to obtain the real position of all the bodies inside the picture, recognize the target and get brightness data (i.e., light curves) from it. Two solutions were presented: a simple one built "from scratch" and a more complex one which made use of already existing programs available online. At the end, the latter solution was shown to have a great performance on real images depicting satellites: the target was perfectly identified and light curves were extracted from it. Future works will be focused on the recognition of unknown objects (i.e., the ones which are neither stars nor the target) from images, with resulting data acquisition.

The second chapter was based on the object classification problem. Starting from the light curve of a body, the main goal was to classify it with machine learning techniques, which have the advantage of learning important features using only labeled training data. In particular, a shape/control classification was performed. The considered classic machine learning algorithms (neural networks with backpropagation and gradient descent, and support vector machines) presented good accuracy in the classification (around 95%), but the lion's share was taken by the surprising and recent techniques called the Extreme Learning Machines (ELM). Indeed, they were shown to have a similar performance capability but with an extremely fast learning speed (300 to 800 times faster than previous methods). It is reasonable to think that, for this reasons, this kind of algorithms will be used more and more in the time to come. Future works will expand the classification task to other features like the constituent material, the dimension or the attitude.

The third chapter, finally, presented results using the Q-learning method to solve the sensor tasking and management problem for space situational awareness. The Q-learning solved a Markov decision process to determine the optimal tasking to maximize the expected reward. The system was simulated for multiple trails during which the deep neural network was updated. The simulation case with 3 geostationary satellites was shown to have a good performance since it was able to reduce the uncertainty on each satellite under the defined threshold and then maintain an almost steady state value. This was an example of application of Q-learning to complex problems and it should encourage the implementation of Q-learning in interesting modern areas like the vision-based control for self-driving objects (e.g., cars or spacecraft, an example of deep learning applied to vision control in self-driving cars is shown in Figure 4.1). Future works will consider the multi-sensor tasking problem and analyse the cases where not enough observations are available to observe all the satellites. In these cases, the reinforcement learning approach should make a decision on which space object should be observed and which errors should be allowed to grow.

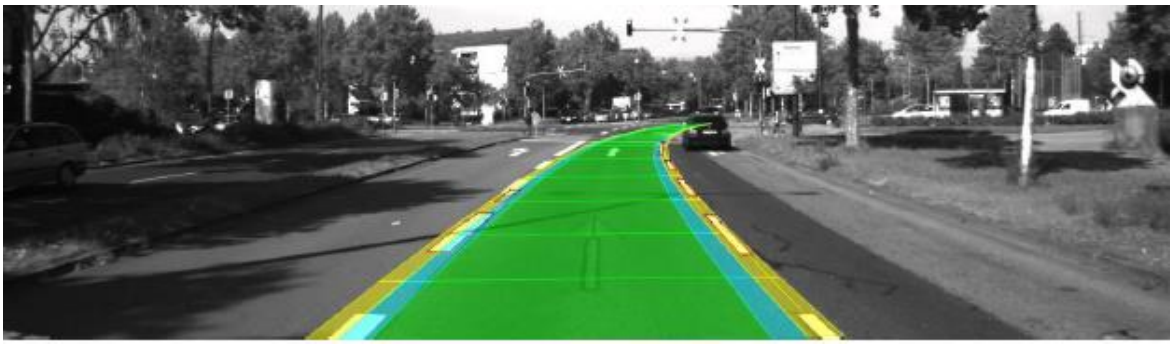


Figure 4.1: Example of deep learning applied to vision control in self-driving cars

Finally, a physical realization of the ideas described in this thesis, making use of Python interfaces with the telescope, will be the main future development.

Appendix A

```python
from spiceypy import *
from math import *
from numpy import *
import os
import re
import glob
import requests
import datetime
import time
from sgp4.earth_gravity import wgs72
from sgp4.io import twoline2rv

def R2P5(y):
    x=int(y)
    d=y-x

    if d>=0.5:
        x=x+0.5
    else:
        x=x-0.5

    return x

def JD2Date(jd): # obtain a date format from Julian date
    seconds=(jd-R2P5(jd))*86400

    jd0=int(jd+0.5)

    if(jd0<2299161):
        c=jd0
    else:
        b=int(((jd0-1867216)-0.25)/36524.25)
        c=jd0+b-int(b/4)+1

    c=c+1524

    d=int((c-122.1)/365.25)
    e=365*d + int(d/4)
    f=int((c-e)/30.6001)

    date=[0,0,0,0,0,0]
    date[1]=f-1-12*int(f/14)
    date[0]=d-4715-int((7 + date[1])/10)
    date[2]=int(c-e+0.5)-int(30.6001*f)

    date[3]=int(seconds/3600)
    seconds=seconds - 3600*date[3]
    date[4]=int(seconds/60)
    date[5]=seconds - 60*date[4]

    if (date[0]<=0):
        date[0]=datetime[0]-1

    return date

def Get_RAdec(Julian_Date, satellite_name, observatory_code, TLE_filename=None):

    # check if there is a TLE_filename or not
    if TLE_filename is None:

        # delete all previous created files
        filelist=glob.glob('*_TLE.txt')
        for file in filelist:
            os.remove(file)

        URL='http://www.celestrak.com/NORAD/elements/geo.txt'

        # the name of the txt file to be created is the date and time
        date_time=time.strftime('%Y-%m-%d-%H-%M-%S')
        file_name=date_time+'_TLE.txt'

        # get the text from URL
        page=requests.get(URL)

        # write the text in the txt file defined above
        f=open(file_name,'w')
        f.write(page.text)
        f.close()

        # save the text in name, line1 and line2 arrays
        with open(file_name) as f:
            for i, l in enumerate(f):
                pass

        # i+1 is the number of lines
        # create list of values (lines of name, line1 and line2)
        lines_name=arange(0,i+2,3).tolist()
        lines_1=arange(1,i+2,3).tolist()
        lines_2=arange(2,i+2,3).tolist()

        mylist=[]
        name=[]
        line1=[]
        line2=[]

        # read all the lines of the txt file
        with open(file_name) as f:
            mylist_temp = f.read().splitlines()

            f.close()

            # modify mylist_temp because has an empty line every 2 lines
            l=0
            while l*2<len(mylist_temp):
                mylist.append(mylist_temp[l*2])
                l+=1

            # record lines in lists
            for l in range(len(mylist)-1):
                if l in lines_name:
                    name.append(mylist[l])
                elif l in lines_1:
                    line1.append(mylist[l])
                elif l in lines_2:
                    line2.append(mylist[l])
        else:
            with open(TLE_filename) as f:
                for i, l in enumerate(f):
                    pass

            # i+1 is the number of lines
            lines_name=arange(0,i+2,3).tolist()
            lines_1=arange(1,i+2,3).tolist()
            lines_2=arange(2,i+2,3).tolist()

            mylist=[]
            name=[]
            line1=[]
            line2=[]

            with open(TLE_filename) as f:
                mylist = f.read().splitlines()

            f.close()

            for l in range(len(mylist)-1):
                if l in lines_name:
                    name.append(mylist[l])
                elif l in lines_1:
                    line1.append(mylist[l])
                elif l in lines_2:
                    line2.append(mylist[l])

        # search if there exists the satellite name in the list of names and
        # save the two lines associated
        regex=re.compile('.*('+satellite_name+').*')
        name_total=[]
        TLE_line1=[]
        TLE_line2=[]
        name_total=[m.group(0) for l in name for m in [regex.search(l)] if m]
        name_total=name_total[0]

        if name_total in name:
            index=name.index(name_total)
            TLE_line1=line1[index]
            TLE_line2=line2[index]

        # from TLE to actual position and velocity (wgs72 is the reference
        # standard)
        satellite=twoline2rv(TLE_line1,TLE_line2,wgs72)
        date=JD2Date(Julian_Date)

        # TLE propagation
        pos,vel=satellite.propagate(date[0],date[1],date[2],
                                        date[3],date[4],date[5])
        position_target_TEME=pos

        # from TEME to ECEF
        twopi=2.0*pi
        d2r=pi/180.0

        tut1=(Julian_Date-2451545.0)/36525.0

        gmst=-6.2e-6*tut1*tut1*tut1+0.093104*tut1*tut1+
        (876600.0*3600.0+8640184.812866)*tut1+
        67310.54841

        gmst=remainder(gmst*d2r/240.0,twopi)

        if gmst<0.0:
            gmst=gmst+twopi

        # build the rotation matrix
        st=[(cos(gmst),sin(gmst),0.0),
            (-sin(gmst),cos(gmst),0.0),
            (0.0,0.0,1.0)]

        st=array(st)

        position_target_ecef = dot(st,position_target_TEME)

        # unload all kernels and clear the kernel pool, set the directory
        spiceypy.kclear()
        os.chdir('/home/simone/Scrivania/ECEF2RAdec/kernels')

        # load SPICE kernel
        spiceypy.furnsh('/home/simone/Scrivania/ECEF2RAdec/kernels'
                        '/kernel.txt')

        # read the Julian date
        JulianDate=str(Julian_Date)
        JulianDate='JD'+JulianDate
        tMeas=spiceypy.str2et(JulianDate)

        # compute observatory latitude, longitude and altitude
        filelist2=glob.glob('Observatories.txt')
        for file in filelist2:
            os.remove(file)

        URL='http://www.minorplanetcenter.net/iau/lists/ObsCodes.html'

        file_name='Observatories.txt'

        # take the list from the website of Minor Planet Center
        page2=requests.get(URL)

        # write the text in a txt file
        g=open(file_name,'w')
        g.write(page2.text)
        g.close()

        data=genfromtxt('Observatories.txt',delimiter=[5,8,8,8,50],
                        skip_header=2,skip_footer=1,
                        dtype=('S3',float,float,float,'S50'),
                        names=('obscode','lon','cos','sin','name'))

        obs_code=str(observatory_code)

        # adjust the code if with 1 or 2 digits
        if len(obs_code)==1:
            obs_code='00'+obs_code
        if len(obs_code)==2:
            obs_code='0'+obs_code

        # collect the information concerning the observatory
        obs=data[data['obscode']==obs_code.encode('utf-8')]

        # parallax constants
        cosine=obs[0][2]
        sine=obs[0][3]
        observatory_name=obs[0][4]
        long=obs[0][1]*pi/180

        # compute the latitude in radians
        lat=atan(sine/cosine)

        # altitude put equal to zero because it cannot be computed
        # from the website
        alt=0

        objName=observatory_name
        objID=obs_code
        refFrameName='IAU_EARTH'

        # generate setup file
        f=open('/home/simone/Scrivania/ECEF2RAdec/kernels/'+objName+'.sut',
               'w+')

        endChar='\n'

        f.write('\\begindata'+endChar)
        f.write("SITES        =('"+objName+"')"+endChar)
        f.write(objName+'_CENTER = 399 '+endChar)
        f.write(objName+"_FRAME = '"+refFrameName+"'"+endChar)
        f.write(objName+'_IDCODE = '+objID+endChar)
        f.write(objName+'_LATLON =('+str(rad2deg(lat))+','+str(rad2deg(lon))+
                ','+str(alt)+') '+endChar)
        f.write('\\begintext '+endChar)

        f.close()

        # run pinpoint
        string='./pinpoint -def '+objName+'.sut '+
        '-pck ./pck00010.tpc -spk '+objName+'.bsp'
        os.system(string)
        spiceypy.furnsh('/home/simone/Scrivania/ECEF2RAdec/kernels/'+
                        objName+'.bsp')

        xGS=spiceypy.spkezr(objID,tMeas,'J2000','NONE','399')

        position_obs_eci=xGS[0][0:3]

        # compute the rotation matrix from ECEF to ECI J2000
        mat=spiceypy.pxform('IAU_EARTH','J2000',tMeas)

        mat=array(mat)

        position_target_eci=dot(mat,position_target_ecef)

        position_target_eci=array(position_target_eci)
        position_obs_eci=array(position_obs_eci)

        spiceypy.kclear()

        # compute the relative position
        rho=position_target_eci-position_obs_eci

        # calculate RA and dec in degrees
        RA=atan2(rho[1],rho[0])*180/pi

        dec=asin(rho[2]/linalg.norm(rho))*180/pi

        return RA,dec


    RA,dec=Get_RAdec(Julian_Date,satellite_name,observatory_code,TLE_filename
```

Appendix B

```python
import time
import datetime
import pytz
import requests
import glob
import os
import spiceypy
from numpy import *
from math import *
from sgp4.earth_gravity import wgs72
from sgp4.io import twoline2rv

def Date2JD(datetime):
    datetime[0]=int(datetime[0])
    datetime[1]=int(datetime[1])
    datetime[2]=int(datetime[2])
    datetime[3]=int(datetime[3])
    datetime[4]=int(datetime[4])
    datetime[5]=int(datetime[5])

    fracday = (datetime[3]+(datetime[4]+datetime[5]/60)/60)/24

    a = 10000*datetime[0]+100*datetime[1]+datetime[2]+fracday

    if datetime[1]<=2:
        datetime[1]=datetime[1]+12
        datetime[0]=datetime[0]-1

    if a<=15821004.1:
        b=-2+int((datetime[0]+4716)/4)-1179

    else:
        b = int(datetime[0]/400)-int(datetime[0]/100)+int(datetime[0]/4)

    jd=365*datetime[0]+b+int(30.6001*(datetime[1]+1))+datetime[2]+fracday+
    1720996.5

    return jd

def R2P5(y):
    x=int(y)
    d=y-x

    if d>=0.5:
        x=x+0.5
    else:
        x=x-0.5

    return x

def JD2Date(jd):
    seconds=(jd-R2P5(jd))*86400

    jd0=int(jd+0.5)

    if jd0<2299161:
        c=jd0
    else:
        b=int(((jd0-1867216)-0.25)/36524.25)
        c=jd0+b-int(b/4)+1

    c=c+1524

    d=int((c-122.1)/365.25)
    e=365*d+int(d/4)
    f=int((c-e)/30.6001)

    date=[0,0,0,0,0,0]
    date[1]=f-1-12*int(f/14)
    date[0]=d-4715-int((7 + date[1])/10)
    date[2]=int(c-e+0.5)-int(30.6001*f)

    date[3]=int(seconds/3600)
    seconds=seconds - 3600*date[3]
    date[4]=int(seconds/60)
    date[5]=seconds-60*date[4]

    if date[0]<=0:
        date[0]=datetime[0]-1

    return date


def Obtain_visible_satellites(observatory_code, alfa, mag_limit,
                              JD=None, TLE_filename=None):

    # check if there is a TLE_filename or not
    if TLE_filename is None:

        # delete all previous created files
        filelist=glob.glob('*_TLE.txt')
        for file in filelist:
            os.remove(file)

        URL='http://www.celestrak.com/NORAD/elements/geo.txt'

        # the name of the txt file to be created is the date and time
        date_time=time.strftime('%Y-%m-%d-%H-%M-%S')
        file_name=date_time+'_TLE.txt'

            # get the text from URL
            page=requests.get(URL)

            # write the text in the txt file defined above
            f=open(file_name,'w')
            f.write(page.text)
            f.close()

            # save the text in name, line1 and line2 arrays
            with open(file_name) as f:
                for i, l in enumerate(f):
                    pass

            # i+1 is the number of lines
            # create list of values (lines of name, line1 and line2)
            lines_name=arange(0,i+2,3).tolist()
            lines_1=arange(1,i+2,3).tolist()
            lines_2=arange(2,i+2,3).tolist()

            mylist=[]
            name=[]
            line1=[]
            line2=[]

            # read all the lines of the txt file
            with open(file_name) as f:
                mylist_temp = f.read().splitlines()

            f.close()

            # modify mylist_temp because has an empty line every 2 lines
            l=0
            while l*2<len(mylist_temp):
                mylist.append(mylist_temp[l*2])
                l+=1

            # record lines in lists
            for l in range(len(mylist)-1):
                if l in lines_name:
                    name.append(mylist[l])
                elif l in lines_1:
                    line1.append(mylist[l])
                elif l in lines_2:
                    line2.append(mylist[l])
        else:
            with open(TLE_filename) as f:
                for i, l in enumerate(f):
                    pass

            # i+1 is the number of lines
            lines_name=arange(0,i+2,3).tolist()
            lines_1=arange(1,i+2,3).tolist()
            lines_2=arange(2,i+2,3).tolist()

            mylist=[]
            name=[]
            line1=[]
            line2=[]

            with open(TLE_filename) as f:
                mylist = f.read().splitlines()

            f.close()

```

```python
        for l in range(len(mylist)-1):
            if l in lines_name:
                name.append(mylist[l])
            elif l in lines_1:
                line1.append(mylist[l])
            elif l in lines_2:
                line2.append(mylist[l])

    # get the position and velocity of each satellite
    satellite=[]
    for k in range(len(line1)-1):
        satellite.append(twoline2rv(line1[k], line2[k], wgs72))

    # compute observatory latitude, longitude and altitude
    filelist2=glob.glob('Observatories.txt')
    for file in filelist2:
        os.remove(file)

    URL='http://www.minorplanetcenter.net/iau/lists/ObsCodes.html'

    file_name='Observatories.txt'

    # take the list from the website of Minor Planet Center
    page2=requests.get(URL)

    # write the text in a txt file
    g=open(file_name,'w')
    g.write(page2.text)
    g.close()

    data=genfromtxt('Observatories.txt',delimiter=[5,8,8,8,50],
                    skip_header=2,skip_footer=1,
                    dtype=('S3',float,float,float,'S50'),
                    names=('obscode','lon','cos','sin','name'))

    obs_code=str(observatory_code)

    # adjust the code if with 1 or 2 digits
    if len(obs_code)==1:
        obs_code='00'+obs_code
    if len(obs_code)==2:
        obs_code='0'+obs_code

    # collect the information concerning the observatory
    obs=data[data['obscode']==obs_code.encode('utf-8')]

    # parallax constants
    cosine=obs[0][2]
    sine=obs[0][3]
    observatory_name=obs[0][4]
    long=obs[0][1]*pi/180

    # compute the latitude in radians
    lat=atan(sine/cosine)

    # altitude put equal to zero because it cannot be computed
    # from the website
    alt=0

    # calculation of the ECEF coordinates for the observatory
```

```
            position_obs_ECEF=spiceypy.spiceypy.georec(long,
                                                       lat,alt,
                                                       6378.135,
                                                       1/298.26)


        # this variable keeps track of the visible satellites
            visible_satellites=[]

            # check if a Julian Date was given
            if JD is None:
                date_time=time.strftime('%Y-%m-%d-%H-%M-%S')
                year=date_time[0:4]
                month=date_time[5:6]
                day=date_time[8:9]
                hour=date_time[11:12]
                minutes=date_time[14:15]
                seconds=date_time[17:18]
                date=[year,month,day,hour,minutes,seconds]
                JD=Date2JD(date)

            else:

                date=JD2Date(JD)
                year=date[0]
                month=date[1]
                day=date[2]
                hour=date[3]
                minutes=date[4]
                seconds=date[5]

                year=str(year)
                month=str(month)
                day=str(day)
                hour=str(hour)
                minutes=str(minutes)
                seconds=str(seconds)

                if month[0]==0:
                    month=month[1]
                if day[0]==0:
                    day=day[1]
                if hour[0]==0:
                    hour=hour[1]
                if minutes[0]==0:
                    minutes=minutes[1]
                if seconds[0]==0:
                    seconds=seconds[1]

            year=int(year)
            month=int(month)
            day=int(day)
            hour=int(hour)
            minutes=int(minutes)
            seconds=float(seconds)

            position=[]
            velocity=[]

            position_satellites_ECEF=[]
            A=array([0,0,0])
```

```python
    B=array(position_obs_ECEF)
    proj_point=[]
    F_phi=[]
    mv=[]
    i=0


# propagate the TLE
    for k in range(len(satellite)):
        pos,vel = satellite[k].propagate(year,month,day,
                                         hour,minutes,
                                         seconds)
        position.append(pos)
        velocity.append(vel)

        # change the coordinates from ECI TEME to ECEF
        twopi=2.0*pi
        d2r=pi/180.0

        tut1=(JD-2451545.0)/36525.0

        gmst=-6.2e-6*tut1*tut1*tut1+0.093104*tut1*tut1+
        (876600.0*3600.0+8640184.812866)*tut1+
        67310.54841

        gmst=remainder(gmst*d2r/240.0,twopi)

        if gmst<0.0:
            gmst=gmst+twopi

        # build the rotation matrix
        st=[(cos(gmst),sin(gmst),0.0),
            (-sin(gmst),cos(gmst),0.0),
            (0.0,0.0,1.0)]

        st=array(st)

        position_satellites_ECEF.append(dot(st,position[k]))

        P=array(position_satellites_ECEF[k])

        # check if the satellite is visible
        AP=P-A
        AB=B-A

        # satellite position projected on the axis from the observatory
        proj_point.append(A+dot(AP,AB)/dot(AB,AB)*AB)

        # calculate the distance between satellite position and the axis
        diff=proj_point[k]-P
        distance=sqrt(diff[0]**2+diff[1]**2+diff[2]**2)

        obs_norm=sqrt(B[0]**2+B[1]**2+B[2]**2)

        # compute the distance of the projected point from the center
        # of the Earth and from the Earth's surface
        proj_norm=sqrt(proj_point[k][0]**2+proj_point[k][1]**2+
                       proj_point[k][2]**2)
        distance_tip=proj_norm-obs_norm

```

```python
            # control that the projected point is above the observatory
            if distance_tip<0 or dot(proj_point[k],B)<0:
                pass

            # check if the satellite is inside the cone
            elif distance<distance_tip*tan(pi*alfa/180):
                i=i+1

                # check if the satellite is really visible
                rho=0.2
                A=pi*(4.8/2)**2
                dist=(sqrt(P[0]**2+P[1]**2+P[2]**2)-obs_norm)*1000
                n=JD-2451545
                L=remainder(280.460+0.9856474*n,360)
                g=remainder(357.528+0.9856003*n,360)
                lam=L+1.915*sin(g*pi/180)+0.020*sin(2*g*pi/180)
                obOfE=23.439-0.0000004*n
                sLam=sin(lam)
                sun_vector_ECI=[cos(lam),cos(obOfE)*sLam,sin(obOfE)*sLam]
                distance_sun=(1.00014-0.01671*cos(g*pi/180)-0.00014*
                             cos(2*g*pi/180))*149600000

                sun_vector=dot(st,sun_vector_ECI)
                sun_vector=array(sun_vector)
                sun_position=sun_vector*distance_sun
                sun=sun_position-P
                observatory=B-P

                cosine_angle=dot(sun,observatory)/(linalg.norm(sun)*
                                                   linalg.norm(observatory))
                phi=arccos(cosine_angle)

                F_phi.append(2/(3*(pi)**2)*((pi-phi)*cos(phi)+
                                            sin(phi)))
                mv.append(-26.7-2.5*log10(A*rho*F_phi[i-1])+5.0*
                          log10(dist))
                if mv[i-1]<mag_limit:
                    visible_satellites.append([name[k],
                                               position_satellites_ECEF[k]])

        return visible_satellites

    visible_satellites=Obtain_visible_satellites(observatory_code, alfa,
                                                 mag_limit, JD=None,
                                                 TLE_filename=None)
```

Bibliography

[1] Vallado, D. and Griesbach, J. D., "Simulating space surveillance networks", Advances in the Astronautical Sciences, Vol. 142, No. 11-580, 2011, pp. 2769-2787

[2] ESA. About SSA. Retrieved February 16, 2017. [Online]. Available: http://www.esa.int/Our˙Activities/Operations/Space˙Situational˙Awareness/About˙SSA

[3] Wikipedia. GPS satellite blocks. Retrieved February 16, 2017. [Online]. Available: https://en.wikipedia.org/wiki/GPS˙satellite˙blocks

[4] Linares, R. and Furfaro, R., "Dynamic Sensor Tasking for Space Situational Awareness via Reinforcement Learning", *AMOS Conference*, 2016, p. 1

[5] Universe Today. What is Low Earth Orbit? Retrieved March 2, 2017. [Online]. Available: http://www.universetoday.com/85322/what-is-low-earth-orbit/

[6] Paluszek, M. et al., "Spacecraft attitude and orbit control", 2nd edition, Princeton Satellite Systems, Inc., 2009, pp. 97-98

[7] ascending node, entry in "The Encyclopedia of Astrobiology, Astronomy and Spaceflight", David Darling. Retrieved February 16, 2017. [Online]. Available: http://www.daviddarling.info/encyclopedia/A/ascending˙node.html

[8] AMSAT. The Keplerian Elements Short Tutorial. Retrieved February 16, 2017. [Online]. Available: http://www.amsat.org/amsat-new/tools/keps˙tutorial.php

[9] ECEF coordinate system. Retrieved February 18, 2017. [Online]. Available: http://what-when-how.com/gps-with-high-rate-sensors/ecef-coordinate-systems-gps

[10] USNO. International Celestial Reference System. Retrieved February 18, 2017. [Online]. Available: http://aa.usno.navy.mil/faq/docs/ICRS˙doc.php

[11] "Flexible Image Transport System: a new standard file format for long-term preservation projects?", July 5, 2012. Retrieved February 18, 2017. [Online]. Available: https://www.vatlib.it/moduli/Allegrezza˙EWASS2012.pdf

[12] WCSTools. Retrieved February 18, 2017. [Online]. Available: http://tdc-www.harvard.edu/wcstools/index.html

[13] VizieR. Retrieved February 18, 2017. [Online]. Available: http://vizier.u-strasbg.fr/index.gml

[14] Wikipedia. Cygnus CRS OA-6. Retrieved February 20, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Cygnus˙CRS˙OA-6

[15] Vallado, David A. et al., "Revisiting Spacetrack Report #3", Astrodynamics Specialist Conference, August 2006

[16] Wikipedia. Simplified perturbations models. Retrieved February 21, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Simplified˙perturbations˙models

[17] Photometry Pipeline 0.9 documentation. Retrieved February 22, 2017. [Online]. Available: http://mommermi.github.io/pp/index.html

[18] Holwerda, B., "Source Extractor for Dummies", Space Telescope Science Institute

[19] Bertin E., 2006, Astronomical Data Analysis Software and Systems XV, ASP Conf. Series 351, 112

[20] Solar System Dynamics, HORIZONS System. Retrieved February 27, 2017. [Online]. Available: http://ssd.jpl.nasa.gov/?horizons

[21] Vallado, David A., "Fundamentals of Astrdynamics and Applications, 2013, Space Technology Library", 4th edition, pp. 184-188

[22] Vallado, David A., "Fundamentals of Astrdynamics and Applications, 2013, Space Technology Library", 4th edition, pp. 162,231-234

[23] Linares, R. and Crassidis, John L., "Space object classification using model and data driven methods", AAS 16-518, 2016

[24] Linares, R. and Furfaro, R., "Space object classification using deep Convolutional Neural Networks", 19th International Conference on information fusion, July 5-8, 2016

[25] Mitchell, T., "Machine Learning", McGraw Hill, 1997, p. 2

[26] Wikipedia. Machine learning. Retrieved March 3, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Machine˙learning

[27] Nielsen, M., "Neural Networks and Deep Learning", online book.

[28] OpenCV. Introduction to Support Vector Machines. Retrieved March 6, 2017. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction˙to˙svm/introduction˙to˙svm.html

[29] Winston, Patrick. 6.034 Artificial Intelligence. Fall 2010. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu

[30] Pang, S. and Yang, X., “Deep Convolutional Extreme Learning Machine and Its Application in Handwritten Digit Classification”, 2016

[31] Huang, G., Bai Z. and Kasun, Liyanaarachchi Lekamalage C., “Local Receptive Fields Extreme Learning Machine”, IEEE computational intelligence magazine, May 2015

[32] Cambria, E. and Huang, G., “Extreme Learning Machines”, IEEE intelligent systems, November/December 2013

[33] Huang, G., “Extreme Learning Machines – Learning Without Iterative Tuning”, IJCNN2012/WCCI2012, June 2012

[34] World Geodetic System 1972. The Department of Defense, May 1974. Retrieved March 12, 2017. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a110165.pdf

[35] Space surveillance: the visual brightness and size of space objects. Retrieved February 21, 2017. [Online]. Available: https://mostlymissiledefense.com/2012/08/21/space-surveillance-the-visual-brightenss-and-size-of-space-objects-august-21-2012/

[36] U.S. Naval Observatory, U.K. Hydrographic Office, H.M. Nautical Almanac Office, “The Astronomical Almanac for the Year 2010”

[37] Silver, David. Reinforcement Learning course. University College London:http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

[38] About Nature. Retrieved March 15, 2017. [Online]. Available: http://www.nature.com/nature/about

[39] Mnih, V. et al., “Human-level control through deep reinforcement learning”, Nature, 26 February 2015

[40] Research. DeepMind. Retrieved March 15, 2017.[Online]. Available: https://deepmind.com/research/dqn/