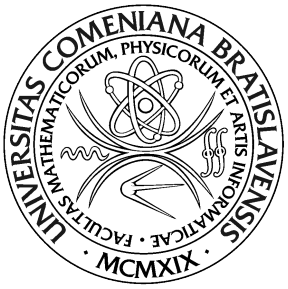
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



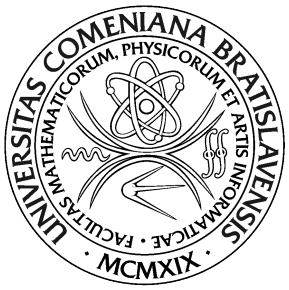
DEVELOPMENT OF AN EFFICIENT TRACKLET BUILDING ALGORITHM FOR SPACE DEBRIS OBJECTS

VÝVOJ ALGORITMU NA EFEKTÍVNE KONŠTRUOVANIE TRACKLETOV VESMÍRNEHO ODPADU

Diplomová práca

2018 Bc. Stanislav Krajčovič

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



DEVELOPMENT OF AN EFFICIENT TRACKLET BUILDING ALGORITHM FOR SPACE DEBRIS OBJECTS

VÝVOJ ALGORITMU NA EFEKTÍVNE KONŠTRUOVANIE TRACKLETOV VESMÍRNEHO ODPADU

Diplomová práca

Study programme:	Applied Informatics
Field of study:	2511 Applied Informatics
Department:	Department of Applied Informatics
Supervisor:	prof. RNDr. Roman Ďurikovič, PhD.
Consultant:	Mgr. Jiří Šilha, PhD.

Bratislava, 2018

Bc. Stanislav Krajčovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Stanislav Krajčovič
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Development of an efficient tracklet building algorithm for space debris objects.
Vývoj algoritmu na efektívne konštruovanie trackletov objektov vesmírneho odpadu.

Anotácia: V súčasnosti sa inštaloval nový 1-metrový ďalekohľad typu Newton na Astronomickom a geofyzikálnom observatóriu v Modre, FMFI UK. Hlavným pozorovacím programom nového systému budú astronomické pozorovania objektov vesmírneho odpadu ako nefunkčné satelity, nosné rakety, či úlomky satelitov. Celé spracovanie takýchto pozorovaní má bežne niekoľko krokov. Najprv sa musia na jednotlivých snímkach identifikovať všetky objekty ako hviezdy či vesmírny odpad. Tomuto kroku hovoríme segmentácia snímky. Pre každý takýto objekt sa vyčíta jeho poloha na CCD snímke (x,y), čas pozorovania a celková intenzita signálu, ktorý daný objekt obsahuje. Ďalší krok je tzv. astrometrická redukcia, kedy sa transformuje poloha objektu z koordinátov CCD snímky do astronomických equatoriálnych súradníc. Po tomto kroku sú hviezdy na snímke známe a možu byť odčítané zo súboru objektov. Následne získame skupinu objektov pre každú snímku, ktoré môžu, ale aj nemusia patriť skutočným objektom nakoľko na snímke sa možu nachádzať rôzne artefakty spôsobené šumom, alebo kozmickými časticami. V prípade, ak sme schopní prepojiť aspoň tri polohy toho istého objektu na troch rôznych snímkach, možeme pre daný objekt vytvoriť tzv. tracklet. Tracklet je séria po sebe nasledujúcich pozorovaní, ktoré patria tomu istému reálnemu objektu (asteroid alebo vesmírny odpad). Metódu, ktorá nám pomôže vytvoriť tracklet nazývame konštrukcia trackletu.
Úlohou študenta/-ky bude naštudovať si literatúru venujúcu sa konštrukcii trackletov z astronomických pozorovaní vesmírneho odpadu. Následne študent/-ka navrhne najvhodnejší, alebo aj vlastný algoritmus na konštrukciu trackletu, ktorý následne naprogramuje a otestuje. Testovanie algoritmu bude uskutočnené na syntetických dátach ako aj na reálnych snímkach na ktorých sa nachádza hviezdne pozadie a známy vesmírny odpad (aj viac objektov súčasne). Výsledky sa porovnajú s predpoveďami pozícii vesmírneho odpadu, ktoré budú študentovi dodané spolu s reálnymi snímkami získanými ďalekohľadmi na Astronomickom a geofyzikálnom observatóriu v Modre, FMFI UK.

Cieľ: Vývoj algoritmu na rýchle a efektívne konštruovanie trackletov objektov vesmírneho odpadu (nefunkčné družice, nosné rakety, atď.) a asteroidov (napr. blízkozemské asteroidy) získaných z astronomických pozorovaní.

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Literatúra: J. Kubica, L. Denneau, T. Grav, J. Heasley, R. Jedicke, J. Masiero, A. Milani, A. Moore, D. Tholen, and R. J.Wainscoat, Efficient intra- and inter-night linking of asteroid detections using kd-trees. Icarus, 189(1):151–168, 2007.
H. Oda, T. Yanagisawa, H. Kurosaki, M. Tagawa, Optical Observation, Image-processing, and Detection of Space Debris in Geosynchronous Earth Orbit, Proceedings of AMOS Conference, Maui, Hawaii, 2014.
T. Schildknecht, K. Schild, A. Vannanti, Streak Detection Algorithm for Space Debris Detection on Optical Images, Proceedings of AMOS Conference, Maui, Hawaii, 2015.

Vedúci: prof. RNDr. Roman Ďurikovič, PhD.
Konzultant: Mgr. Jiří Šilha, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 14.10.2016

Dátum schválenia: 14.10.2016 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I hereby certify that this thesis has been composed by me and is based on my work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged.

. . . . . .

Bratislava, 2018 Bc. Stanislav Krajčovič

Acknowledgement

I would like to thank my supervisor, prof. RNDr. Roman Ďurikovič, PhD. for all his invaluable and expert advice and guidance, which helped me successfully finish this thesis. I would also like to thank my consultant, Mgr. Jiří Šilha, PhD. for provided knowledge about astronomy and astrodynamics; for all his advice and support. I must also thank my colleagues from the YACGS seminary for their patience and input.

Abstract

Astronomical observations of a specific portion of the night sky produce images saved in Flexible Image Transport System (FITS) format. They contain basic information such as date, exposure time and other. However, these files are processed in a complex pipeline beforehand and data about the objects inside is gained. This document contains summary of prerequisite knowledge needed, such as object dynamics, already existing solutions and analysis of inputs. Then, we describe the design and creation of algorithms used to create tracklets, which are observations of the same object in time. We describe our reasoning behind using linear regression, Initial Orbit Determination and not using Hough transform. The implementation, including methods and class variables, follows. In conclusion, we review the effectiveness of our implementation and suggest possible improvements.

Keywords: space debris, observation, tracklet

Abstrakt

Astronomickými pozorovaniami časťami nočnej oblohy získavame obrázky uložené v tzv. Flexible Image Transport System (FITS) formáte. Obsahujú základné dáta, ako dátum, čas expozície a iné. Tieto súbory sú avšak spracované v komplexnom systéme a tým sú získané informácie o objektoch na snímkach. Tento dokument obsahuje zhrnutie potrebných vedomostí, ako napríklad dynamiku telies, existujúcich riešení a analýzu vstupných dát. V ďalšom kroku popisujeme dizajn a tvorbu algoritmov použitých na tvorbu trackletov, čo sú vlastne pozorvania toho istého objektu v čase. Zároveň odôvodňujeme dôvody na použitie lineárnej regresie, algoritmu Initial Orbit Determination a dôvod na vynechanie algoritmu Hough transform. Po tejto časti nasleduje implementácia, ktorá zahŕňa metódy a triedne premenné. Na záver zhrnieme efektivitu našej implementácie a navrhneme možné vylepšenia.

Kľúčové slová: vesmírny odpad, pozorovanie, tracklet

Contents

Chapter 1

Introduction

While we are able to put vehicles, both manned and unmanned, into space, in majority of cases it is not cost efficient to return them back to Earth. In some cases, such as from unpredictable events, mistakes in calculations or even from the effects of the Sun, orbiting satellites collide or break up. Therefore, the term space debris encompasses all artificial non-functional objects orbiting around Earth.

As is mentioned in this document, there are almost no space debris creation mitigation processes in place and those which are, are too insignificant to make a visible difference. This results in the rapidly increasing number of space debris. Even though the major agencies, for example European Space Agency (ESA) or National Aeronautics and Space Administration (NASA), already have plans for major space debris clean-ups in motion it is still vitally important to track, catalogue and observe existing debris, as it threatens missions - both future and present - and functioning satellites.

In this thesis we analyse and provide another solution to the problem of assigning observations of the same object in time together - tracklet building.

Chapter 2

Introduction to space debris and observations

Each space debris object has different physical properties, such as size and shape and is made of different materials which determine its behaviour and ease of tracking. Usually, in literature, space debris is categorized into five different groups by its type:

1. mission-related debris,
2. fragmentation debris,
3. non-functional spacecraft,
4. rocket bodies,
5. anomalous debris.

Mission-related debris, despite having many different sources and causes, is the most clearly divided between debris released intentionally and unintentionally. Examples of the former are launch adapters, lens covers, and many

other components associated with launch events and payload deployment. The latter category contains protective gloves, cooling liquids, or small particles released from material decay. Common feature of all these objects is that they are a result of a spacecraft's deployment, activation or operation – therefore the term mission-related debris (Klinkrad, 2006).

The polar opposite of mission-related debris is fragmentation debris. As the name indicates, it is composed from objects or particles created during destructive disassociation of a rocket body or an orbital payload and as a result of deterioration when smaller-than-the-parent-object fragments are created. Breakups may be intentional or accidental and are the largest source of catalogued space debris. Products of breakups are ejected into the surrounding area with various initial velocities and spread until they reach the limit of maximum inclination and altitudes of the debris (on the Peaceful Uses of Outer Space. Scientific and Subcommittee, 1999).

Another commonly described type of space debris in various literature is non-functional spacecraft. As of January 2017, there have been 5253 launches of human-made objects into space since the first deployed satellite Sputnik-1 on October 4, 1957 (ESA, 2017). However, a satellite does not have to break into smaller pieces to be considered debris. Functional spacecrafts that reach their end of life are either re-orbited or left in their former orbit. Historically, re-orbiting manoeuvres were performed only in geostationary orbit (GEO, approximately 36,000 kilometres above Earth's equator) and by spacecrafts carrying nuclear material in low Earth orbit (LEO, approximately 2,000 kilometres above Earth's equator), as well as for vehicles with crew and for reconnaissance. Nevertheless, according to the Mitigation Guidelines released in 2002 by Inter-Agency Space Debris Coordination Committee (IADC), spacecrafts in LEO should be allowed to fall into

the atmosphere and burn up within 25 years of mission end and spacecrafts in GEO should be re-orbited at least 300 kilometres above the GEO orbital ring and left in so-called *graveyard orbit*. Satellites in GEO are not lowered because it is not efficient to carry extra fuel specifically for this purpose.

The last category of space debris consists of rocket bodies. Spacecrafts mentioned in the previous section are launched on vehicles which are constructed for this single purpose. Deployment process consists of one or more phases that are represented by rocket bodies. The number of rocket bodies needed for ascent into desired altitude is proportional to the altitude – for example spacecrafts with missions in LEO only need one rocket body while those in GEO need may need up to three. First stages need to have enough thrust to lift the satellite despite gravity and air resistance and are generally bigger than other stages which are used to position the spacecraft in the final steps of deployment. As such, larger rocket bodies usually re-enter into the atmosphere and burn up or fall into the ocean, while the smaller stages are left at various altitudes. This kind of space debris poses danger especially because of its large dimensions and potentially leftover fuel that may cause explosions which produce new fragmentation debris.

2.0.1 Small space debris

The size of space debris varies from particles less than one millimetre small to objects more than one metre large. While the smallest particles are difficult to track, the amount of those that have size less than 1 centimetre is estimated to be more than 170 million (ESA, 2017). Other sources list the population of debris with diameters bigger than 1 mm as $3 * 10^8$ objects and $3 * 101^3$ objects with diameter bigger than 1 µm (Klinkrad, 2006).

A major source of sub-centimetre reproductive space debris is the effect

of harsh space environment on spacecraft surface materials. Intense UV radiation and atomic oxygen cause decay on spacecraft surfaces usually coated in paint for thermal purposes or other thermal protection materials. They are categorised as fragmentation or sometimes as anomalous debris.

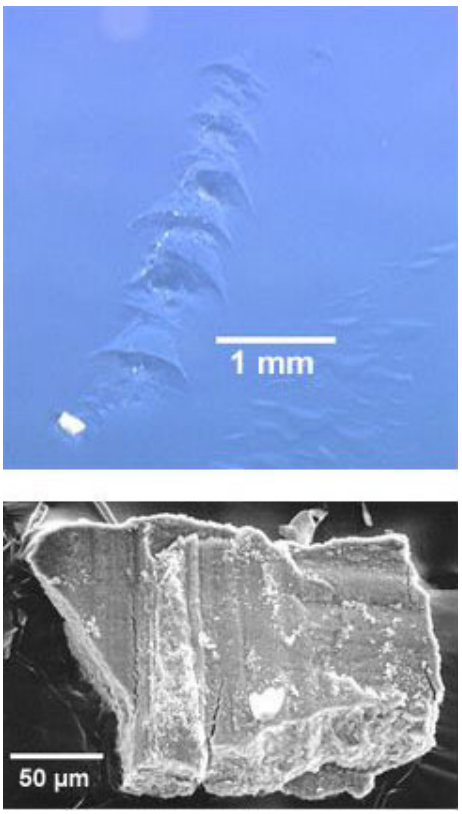


Figure 2.1: Paint flakes captured by Mir Environmental Effects Payload. Image downloaded from (NASA, 2018).

Hypervelocity collisions of small debris objects create particles upon impact which are called ejecta and are part of the fragmentation debris group. Figure 2.2 shows a hypervelocity impact which created a hole in a panel of the Solar Maximum Mission satellite.

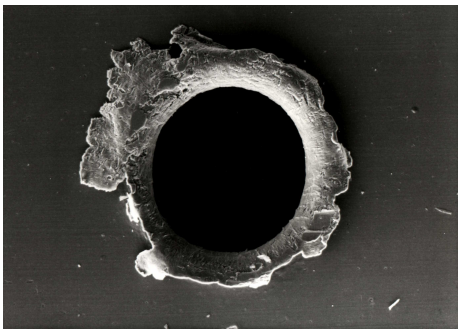


Figure 2.2: A hole made by debris in a solar panel. Image downloaded from (NASA, 2018).

While delivering payloads into orbit, burning process of solid rocket motors releases particles called SRM slag and SRM dust which are classified as non-fragmentation, unintentional debris and are composed mainly of aluminium oxide. Figure 2.3 shows a piece of SRM slag recovered from a test firing of a shuttle solid rocket booster.

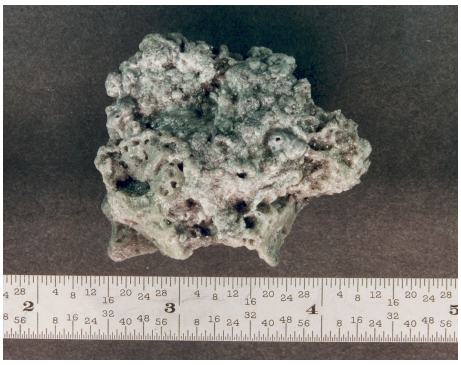


Figure 2.3: SRM slag. Image downloaded from (NASA, 2018).

Another similar but rare type is droplets of sodium-potassium alloy (NaK), a coolant in Russian Radar Ocean Reconnaissance Satellites (RORSAT). Since this kind of reactor is no longer used, NaK droplets along with Westford Needles (see next paragraph) are considered a historic, non-reproducing space debris.

A communications experiment between years 1961 and 1963 consisted of deploying copper wires around the Earth. These copper wires were 1.78 cm long and 25.4 µm in diameter - see Figure 2.4. However, the deployment failed and the estimated mass of Westford Needles is less than 60 kg in two clusters.

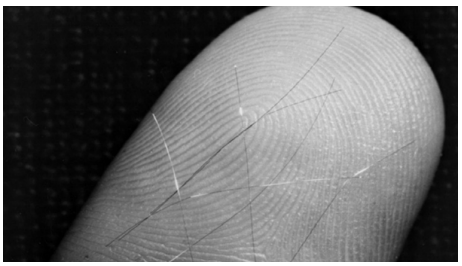


Figure 2.4: Westford needles. Image downloaded from (Massachusetts Institute of Technology, 2009).

While being the major contributor to the population of space debris, small objects are, as mentioned, virtually unobservable from the ground - only by radars and space-based telescopes positioned on LEO - and therefore are not the main focus of this thesis. For further information see (Klinkrad, 2006) or (Šilha, 2012).

2.0.2 Large space debris

The most extensive database of tracked objects in orbit is maintained by US Space Surveillance Network (USSSN). Only objects which exceed certain sizes at certain altitudes can be tracked with optical measurements or in some cases with ground-based radar. Due to this, USSSN incorporates only debris larger than 5 cm to 10 cm in LEO and 30 cm to 20 m in GEO. Currently, USSSN contains more than 42,000 tracked objects with more than half of them still in orbit. About 24% of them are satellites and 18% are spent upper stages and other mission-related objects (ESA, 2017). Out of 175

fragmentation events recorded since 1961 until January 2002, 48 have been categorized as deliberate explosions or collisions, 52 as propulsion system explosions, 10 have been caused by aerodynamic forces, 7 are believed to have been electrical system failures and 1 was an accidental collision (Klinkrad, 2006).

Deliberate collisions have been caused mainly as test scenarios for Strategic Defence Initiative (SDI) experiment. In the first case, an anti-satellite missile was fired and destroyed Solwind P78-1 satellite. The second collision was between the USA-19 spacecraft and an upper stage used to bring it into the orbit. Number of fragments for both of these events was 285 and 13 respectively but as a result of natural cleaning processes or deliberate planning, from H-10 event only 33 objects were on orbit by January 2002 and from Solwind P78-1 only 2. However, the largest fragmentation event occurred in year 2007 when China intentionally destroyed its non-functional weather satellite Fengyun-1C, generating more than 3,215 new fragments.

Even though unintentional collisions are rare and only 0.2% of all catalogued objects until February 2009 have been created in this way it is expected that they will be a major cause for debris in the future. On November 13, 1986 an Ariane-1 H-10 upper stage exploded, causing a fragment cloud containing 488 catalogue entries. The first unintentional in-orbit collision came to pass 10 years later between a French satellite and the mentioned H-10 upper stage explosion fragment. The Cerise satellite was still able to function afterwards and the event caused only one new piece of catalogued debris (Šilha, 2012).

A particularly interesting class of space debris is anomalous debris. It's a specific group that has small velocity and high area to mass ratio (A/M) and its formation process is unknown. The exemplary member of anomalous

debris is multilayer insulation (MLI), paint flakes and defunct solar panels. MLI can be divided into two types based on their purpose – as a cover/outer layer, and as a reflector/inner layer and is used as thermal protection to decrease thermal noise on antennae and satellite buses of spacecrafts.

The rapid creation of new debris of non-negligible sizes is a major concern due to the phenomenon called *Kessler syndrome*. The term was coined in 1978 and it means that each collision would produce several hundred objects large enough to catalogue, increasing the rate that future collision breakups would occur, resulting in an exponential growth in the collision rate and debris population (Kessler and Cour-Palais, 1978).

Spent upper stages and non-functional satellites are one of the largest and most compact space debris. Even though many of them re-enter the Earth's atmosphere on purpose, or are elevated on further orbits, and thus their life on orbit is relatively short, they still pose a threat to functional satellites or missions in progress. Figure 2.5 shows such spent upper stage, specifically Agena D, in the past the most launched American upper stage.

Figure 2.5: Agena D upper stage. Image downloaded from (NASA, 2018).

2.1 Observations and telescopes

Optical telescopes are usually placed at high altitudes, with minimal light pollution and good meteorological and atmospheric conditions. Telescopes used for satellite tracking must be operated at *astronomical night* (Sun being more than 18° under the horizon) while the tracked objects must still be illuminated by the Sun (Klinkrad, 2006).

Telescopes are divided into two groups: refractors and reflectors. The first type uses lens systems to observe objects while the second one uses mirror surfaces to focus incoming light. Reflectors are categorized into four subgroups:

1. Newton telescopes,
2. Cassegrain telescopes,
3. Coudé telescopes,
4. Ritchey-Chrétien telescopes.

The base upon which a telescope is placed is called a mount. As with telescopes, there are many types of mounts with each giving different advantages than the other. Mounts are able to rotate both vertically and horizontally (Klinkrad, 2006).

A telescope collects photons which are reflected or emitted by a space object. Usually, the origin of photons is the Sun and depending on the angle between the Sun, the reflection efficiency of the target is determined. The light is then translated into an image which can be used to generate an exposure on an Charge-Coupled device (CCD). CCDs are photosensitive and solid-state imaging sensors which convert incoming photons into electric

charges on an array of photodetectors. The time-tagged information coming from the photodetectors can be reconstructed into an image with a varying resolution. CCDs naturally heat up and the danger of thermal noise corruption is to a degree mitigated by using active coolants, such as liquid nitrogen (Klinkrad, 2006).

2.1.1 ESA OGS

European Space Agency (ESA) Optical Ground Station (OGS) is a Zeiss telescope built in the Teide Observatory, in Tenerife, Spain, 2,400 m above sea level. It's a one metre telescope with focal length 13.3 m originally used for tests with laser link, space debris observation and other astronomical night observations. The CCD camera has a field of $4000x4000$ pixels. Throughout years 2009 to 2013 the telescope has been credited by Minor Planet Center (MPC) with discovery of 38 minor planets (ESA Science, 2013).

Figure 2.6 shows the schematic of ESA OGS. One of the most important parts is the Cassegrain focus where the CCD camera is used for observation of asteroids and for long-term monitoring of comets (ESA Science, 2013).

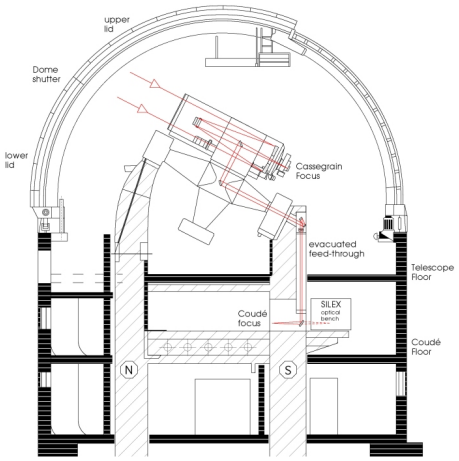


Figure 2.6: ESA OGS schematic. Image downloaded from (ESA Science, 2013).

Figure 2.7 is an image of the Zeiss telescope on an English mount.

Figure 2.7: ESA OGS telescope. Image downloaded from (ESA Science, 2013).

2.1.2 FMPI AGO

Astronomické a geofyzikálne observatórium (Astronomical and geophysical observatory, or AGO) is located in Modra, Slovakia and belongs to Fakulta matematiky, fyziky a informatiky (Faculty of mathematics, physics and informatics of Comenius University, or FMPI). The observatory has a main reflector telescope with a 0.7m Newton design Zeiss telescope with focal length

of 2.961m, field of view of $28.5x28.5$ arc-min on an Equatorial mount. The CCD camera has resolution of $1024x1024$ pixels.

Figure 2.8 shows the schematic of the main reflector telescope and its position at AGO observatory.

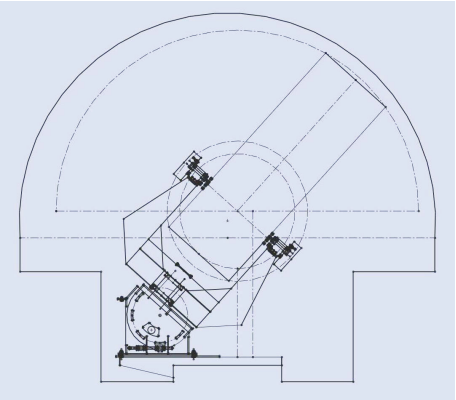


Figure 2.8: Telescope schematic. Image downloaded from (KAFZM FMFI UK, 2017).

Figure 2.9 shows the main telescope which is operated by a computer in the small AGO cupola.

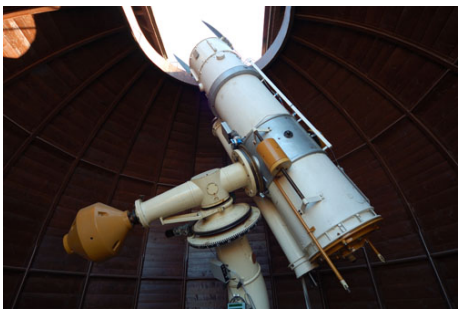


Figure 2.9: The main Zeiss telescope. Image downloaded from (KAFZM FMFI UK, 2017).

Chapter 3

Object dynamics

Contents of this chapter include an overview of physical laws and astronomical systems and algorithms describing movement of space debris objects which have the biggest impact on further contents of this thesis.

3.1 Kepler's laws of orbital motion

Kepler's law of orbital motion describes motion of a satellite in regards to gravitational centre of mass, e.g. Sun to the centre of the galaxy, Earth to Sun, etc. Generally, it states that such motion follows a trajectory which is always elliptically shaped.

In astronomy, an object's position in an inertial system, e.g. geocentric or heliocentric, at specific time (reference epoch t) can be defined from two different angles.

Either we define a state vector (three position vector components and three velocity vector components) or we define six orbital elements for an elliptical type of orbit:

- a - semi-major axis [m]

- e - eccentricity [-]
- i - orbital inclination [°]
- Ω - right ascension of ascending node (RAAN) [°]
- ω - argument of pericenter/perigee
- M_t - mean anomaly at reference epoch t [°]

Elements a and e define shape of the ellipse while i and Ω define orientation of the elliptical plane, ω defines the orientation of the ellipse in the elliptical plane and M_t defines position of an object on the ellipse (Montenbruck and Gill, 2005).

3.2 Equatorial coordinate system

To describe an object's position on celestial sphere, we use astronomical coordinate system called Equatorial coordinates defined through two parameters, right ascension (RA, $\langle 0°, 360° \rangle$) and Declination (Dec, $\langle -90°, 90° \rangle$). This coordinate reference system is fixed on Earth's equator and vernal equinox (Montenbruck and Gill, 2005). See Figure 3.1 for an illustration of RA/Dec with regards to Earth.

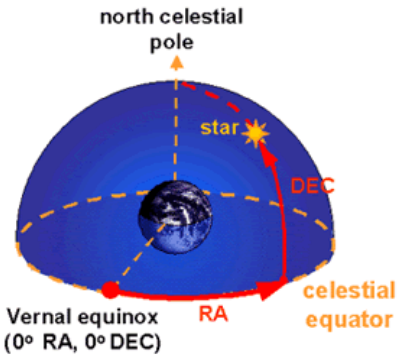


Figure 3.1: RA/Dec illustrated.

3.3 CCD and image reference frame

The first product of an observation is an object's position in the CCD reference frame (x, y). To transform from the x and y coordinates to RA/Dec a so-called *astrometric reduction* is employed which, in general, means linking the two systems together.

The primary goal is to determine the six orbital elements defined in Section 3.1 from RA/Dec in Section 3.2. To confirm that an object is on a conic section we need to find a solution to orbital elements which will yield an elliptical orbit. For more information, see reference (Montenbruck and Gill, 2005).

3.4 Initial orbit determination problem

In order to calculate the orbit of an object observed with an optical telescope the angle measurements are used (RA/Dec, Section 3.2). To fully determine all six orbital elements we must use at least six parameters - specifically

three pairs of angle measurements of the same object. Usually, three pairs of RA/Dec are required for three different observation epochs, e.g. few minute intervals for satellites.

Methods on how to perform Initial Orbit Determination (IOD) are well defined in the astronomical community. The output of this procedure are position vectors or position and velocity vectors for observation epochs. These vectors are then used to determine said orbital elements (Montenbruck and Gill, 2005).

In the case of observations not being of the same object, some orbital elements will be missing. This characteristic can be used as a filter, or to validate gathered data.

Chapter 4

Existing tracklet building solutions

4.1 k-d trees

k-dimensional trees (k-d trees) are a hierarchical data structure that recursively partitions both the set of data points and the space in which they reside into smaller subsets and subspaces. Each node in a k-d tree represents a partial region of the whole space and a set of points contained in the region (Bentley, 1975).

Complexity of k-d trees is similar, or identical in some cases, to other tree data structures. For example, adding a point into a balanced k-d tree takes $O(log\ n)$ time and removing a point from a balanced k-d tree takes $O(log\ n)$ time as well. There are several approaches to constructing a k-d tree, such as finding a median among points (with complexity $O(n)$), sorting all points (with complexity $O(log\ n)$) or sorting a fixed number of randomly selected points.

4.1.1 Efficient intra- and inter-night linking of asteroid detections using kd-trees

The main focus of the paper (see (Kubica et. al., 2007)) is the description of then under development Panoramic Survey Telescope and Rapid Response System (Pan-STARRS). The paper describes the capabilities of the system in areas such as identification of detection of moving objects in our solar system and linking of those detections within and between nights, attributing those detections to catalogued objects, calculating initial and differentially corrected orbits and orbit identification. Further, it illustrates k-d tree algorithms as suitable for linking of objects. The paper contains the description of their own pseudo-realistic simulation of the Pan-STARRS survey strategy and shows the results on both simulated and real data sets.

Pan-STARRS

The paper describes future goals of Pan-STARRS - obtaining two images per night of each Solar System survey field and using these images to distinguish between real and false detections and to separate stationary and moving transient near-Earth objects (NEOs). The object is obtained and verified across several consecutive nights in order to have high probability of being real and to be able to be submitted to the Minor Planet Center (MPC, see Chapter 8, Section 8.1.1)

Pan-STARRS MOPS

Images are firstly preprocessed - cosmic rays and other noise or irrelevant objects need to be removed, they are aligned and combined into a single image. The resulting image is called *master image* in the paper and is further

combined with other master images to create a static-sky image used to produce yet another image containing only transient sources and noise. It is then searched for asteroids and comets. The preprocessing phase ends with all the identified sources from both images, along with their metadata (time, trail length, axis orientation, flux, etc.), being passed to the later stages.

The first step - linking intra-night detections of spatially and temporally close objects, which, in addition, have fixed speed among multiple detections, into tracklets - is being done by comparing expected and detected trail length and orientation. Only the objects which pass through this filter are combined into tracklets. The second step is the inter-night linking of tracklets into collections called tracks. The tracks are verified by IOD (see Chapter 3, Section 3.4) and Orbit Determination (OD, see (Klinkrad, 2006)). The complexity of the linking increases like ρ^2, where ρ is the number of detections/deg^2. This can be solved by employing k-d trees to reduce the complexity that increases like $\rho \; log \; \rho$ (Kubica et. al., 2007).

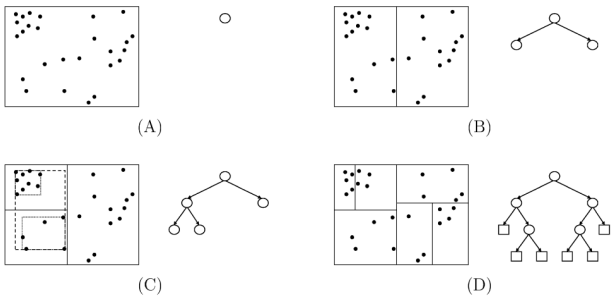


Figure 4.1: k-d tree schematic. Image from (Kubica et. al., 2007).

The k-d tree is created from top to bottom, having full image as the root node. At each level the data in form of points is used to create a bounding box which is then saved in the corresponding node. The points are then split into two disjoint sets at the widest dimension of the node and bounding

boxes are created and assigned again. The recursion is stopped when the currently created node owns less than pre-determined number of points and such node is marked as a leaf node. The construction process is illustrated in Figure 4.1.

Searching for spatially close nodes is done by descending the tree depth first and searching for points within a radius. If the algorithm finds a node that falls out of the radius it stops because it is guaranteed that each child of the node will fall out of the radius as well. If the algorithm reaches a leaf node, the points in it are tested for the distance from the point of query and eventually added to the results. The algorithm is extended by another constraint - temporal significance. In the paper, they consider each detection as the start of a potential tracklet and look at temporally subsequent detections to judge their relevance and add them to the tracklet. Time is added to the k-d tree as a third dimension. The result of having time as the third dimension is illustrated in Figure 4.2 as a cone which spread is controlled by the maximum allowed speed.

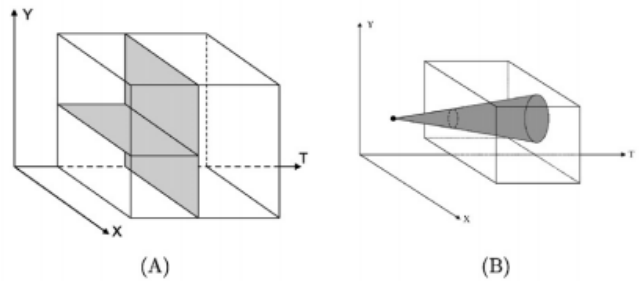


Figure 4.2: k-d tree temporal search. Image from (Kubica et. al., 2007).

Inter-night linking is out of scope of this thesis, similarly to creation of synthetic data and their description can be found in the paper - see (Kubica et. al., 2007).

4.2 Uniform linear motion detection

Linear motion is a one dimensional motion along a straight line and mathematically can be explained using one spatial dimension. One of the two types of linear motion is uniform linear motion, which has constant velocity. In the most cases space debris does not follow straight line nor its acceleration is zero. However, when observing under narrow enough field of view (FOV), only part of its trajectory is captured and the object moves according to uniform linear motion.

4.2.1 Optical observation, image-processing, and detection of space debris in GEO

The paper (see (Oda et. al., 2013)) describes an efficient detection of space debris on GEO orbit using a telescope with 3.17° FOV producing images with a resolution of $2048x2048$. The authors describe image processing and the successive application of the uniform linear motion algorithm to detect objects which are then either matched with the USSTRATCOM catalogue or classified as newly discovered debris.

Image processing

The image is preprocessed by subtracting a master dark bias frame from raw images to reduce noise, correcting the sensitivity of pixels and uneven illumination, masking bright pixels and subtracting the sky background frame. Due to the long exposure of 4.7 sec, the stars appear as streaks approximately 13 pixels long and are removed. Noise, such as cosmic rays, do not have diffusive shape and are removed too. The resulting image contains objects that have diffusive shape and are extracted by Gaussian fitting (Oda

et. al., 2013).

Detection algorithm

The detection algorithm is based on the facts mentioned in Section 4.2 - when we select small enough part of the whole trajectory of an object, the part adheres to the uniform linear motion. After selecting an appropriate candidate from an image (an object having sufficient intensity and correct shape), the next image is considered and only objects that show uniform linear motion and are the brightest are picked. This procedure is repeated until there are 9 positive matches out of 18 light frames and the object is marked as orbital object. See Figure 4.3 for illustration of the detection algorithm.

In conclusion, the algorithm has proven to be effective and efficient and lead to discovery of more than 200 unidentified objects in the span of five nights (Oda et. al., 2013).

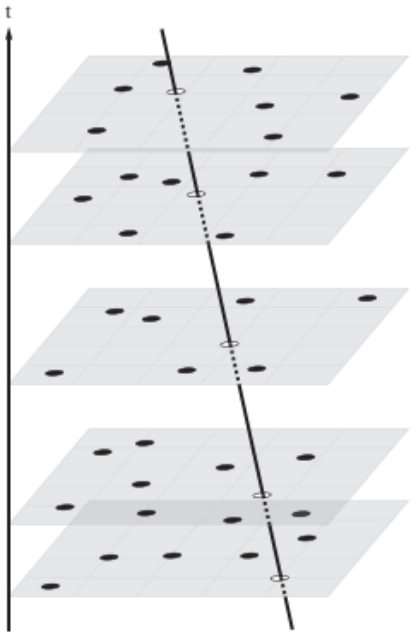


Figure 4.3: Uniform linear motion throughout successive images. Image downloaded from (Oda et. al., 2013).

Chapter 5

Technical and practical requirements

Our application is a part of a system, or a pipeline, and its inputs and outputs need to be precisely specified. Contents of this chapter consist of an overview of inputs coming from previous stages of the pipeline.

5.1 Input data

Information about potential unknown objects are passed to the application in the form of input files. There are two formats of files which contain relevant metadata, such as right ascension, declination, magnitude, and other, which are vital to the correct functioning of algorithms of tracklet building described in detail in Chapter 6.

The first is Flexible Image Transport System (FITS), a standardised and the most commonly used digital file format in astronomy. The second is an output from previous stage of the pipeline, a .cat file.

The formats are described in length in sections 5.1.1 and 5.1.2 respec-

tively.

5.1.1 Flexible Image Transport System files

FITS is the standard archival data format for astronomical data sets.

It was originally designed in year 1981 for transporting image data on magnetic tapes between research institutions. In year 1982, FITS was officially endorsed by International Astronomical Union (IAU) as the format for the interchange of astronomical data.

FITS files in this thesis are used as a source of data about objects and for image stacking (FIT, 2016).

File structure

FITS files consist of three main FITS Structures (structures):

- primary header and data unit (HDU, mandatory)
- conforming extensions (optional)
- other special records (optional).

In our case, all FITS Files contain only the primary HDU which is sometimes referred to as Basic FITS File or a Single Image FITS File.

If used, a structure contains a higher than zero number of FITS Blocks (blocks), each with size of 2880 bytes. Primary HDU always starts with the first block and each following structure begins with a block starting right after the end of the last block of previous structure. Furthermore, the primary HDU and each extension HDU must contain non-zero number of 2880-byte FITS Header (header) blocks, which are mandatory, and an optional array of associated 2880-byte data blocks (FIT, 2016).

Data arrays

If not empty, primary data array has a size of $1x999$ and is represented by continuous stream of bits. Each data value in the array consists of a fixed number of bits which are given in the keyword *BITPIX*. If length of data is shorter than length of final block, the difference is filled by setting all the trailing bits to zero (FIT, 2016).

An example of a partial FITS Data array is shown in Figure 5.1 below.

```
>>> data[30:40, 10:20]
array([[350, 349, 349, 348, 349, 348, 349, 347, 350, 348],
       [348, 348, 348, 349, 348, 349, 347, 348, 348, 349],
       [348, 348, 347, 349, 348, 348, 349, 349, 349, 349],
       [349, 348, 349, 349, 350, 349, 349, 347, 348, 348],
       [348, 348, 348, 348, 349, 348, 350, 349, 348, 349],
       [348, 347, 349, 349, 350, 348, 349, 348, 349, 347],
       [347, 348, 347, 348, 349, 349, 350, 349, 348, 348],
       [349, 349, 350, 348, 350, 347, 349, 349, 349, 348],
       [349, 348, 348, 348, 348, 348, 349, 347, 349, 348],
       [349, 349, 349, 348, 350, 349, 349, 350, 348, 350]], dtype=int16)
```

Figure 5.1: FITS data array represented as 2 byte integer. Image downloaded from (Astropy Developers, 2018).

Headers

Headers contain only limited set of text characters encoded in ASCII. Allowed characters are highlighted in Figure 5.2.

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F		127	0111 1111	7F	[DEL]

Figure 5.2: Allowed ASCII.

A header contains at least one header block, each consisting of maximum 80-character keyword sequence. In the 2880-byte block, this puts the maximum number of records at 36. The logical end of the block is marked by the END keyword and if there is space left, it is filled with ASCII character for space.

Keywords are composed of key-value pairs and an optional comment. The presence of the value indicator (*'= '*, ASCII character for equals followed by ASCII character for space) determines that the key has a value associated with it. The values field can hold any of the following types:

- character string (eg. '27/10/82')
- logical value (represented as T or F)
- integer number (eg. 12)
- real floating-point number (eg. -12.5)

- complex integer number (eg. $2 + 4i$)
- complex floating-point number (eg. $7.8 + 1.7i$).

Each FITS file (file) must contain several mandatory keywords (such as *SIMPLE* - describes whether FITS file is a Basic Fits File or not, and other) which are reserved and have fixed types of their values. FITS format also contains reserved keywords that are not mandatory (such as *DATE* - date on which the HDU was created, and other).

In our case, the most relevant keywords are *DATE-OBS* and *EXPTIME*, or *EXPOSURE* in files where the keyword *EXPTIME* is missing (FIT, 2016).

An example of FITS Header is shown in Figure 5.3.

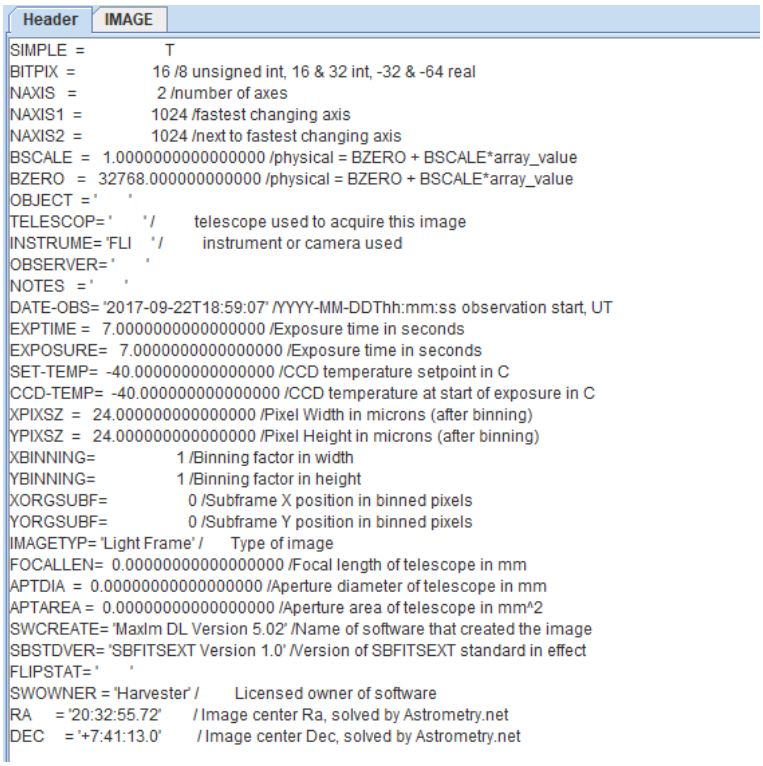


Figure 5.3: FITS Header.

5.1.2 .cat files

.cat files are text files generated by the Astrometrica tool accepting FITS files (see 5.1.1) as input, providing interactive graphical user interface to manipulate each FITS file and performs image processing, segmentation, and astrometrical reduction resulting in an output file with the extension .cat.

Detailed description of the contents of a .cat file is in the following subsection.

Contents of a .cat file

First four lines of a .cat file are a header - first line is empty, second is the name of the column, third contains units in which are values in specific column represented, and fourth is a separator, as shown in Figure 5.4.

```
Type    RA          dRA       Dec.       dDec     R      dR        x        y      Flux  FWHM   Peak    Fit  Designation
     h  m   s        "       °  '   "       "      mag    mag                       ADU    "     SNR    RMS
-------------------------------------------------------------------------------------------------------------------------
```

Figure 5.4: .cat header.

Meaning of each column is as follows:

1. Type - column outlining the type of a detected object; has four valid values:
   - R - a star which has been matched with the star catalogue; if the value is in parentheses, the deviation between the two is too large
   - H - a manually marked object
   - S - a star which has not been matched with the star catalogue
   - ? - an unknown object
2. RA - right ascension in hours/minutes/seconds (see Chapter 3, Section 3.2)

3. dRA - deviation of values of RA between centroid created by Astrometrica and the star catalogue
4. Dec. - declination in degrees/minutes/seconds (see Chapter 3, Section 3.2)
5. dDec - deviation of values of DEC between centroid created by Astrometrica and the star catalogue
6. R - brightness in magnitude
7. dR - deviation of values of brightness between centroid created by Astrometrica and the star catalogue
8. x - position of an object in FITS file on x axis
9. y - position of an object in FITS file on y axis
10. Flux - total intensity of all pixels in ADU (analog to digital unit)
11. FWHM - "*full width at half maximum*" describes a measurement of the width of a diffused object
12. Peak - signal to noise ratio; describes the difference between the pixel with the maximal value and background
13. Fit - the deviation of fitting of centroids in image to the star catalogue
14. Designation - an user given text

5.1.3 Processing server parameters

The server is a high CPU computer with Linux operating system capable of remote access for multiple users via console or x2goclient. The server is a

Supermicro SuperServer 7048R-TR with an Intel® Xeon® E5-2640 v4 with Intel® Turbo Boost Technology, hyper-threading and virtualization capabilities. It also contains 32GB of ECC DDR4 SDRAM 72-bit system memory, 400GB SSD system disk and 4TB SATA 3.0 HDD in RAID1 configuration for additional data storage. 920W power supply and free slots for an additional CPU, system memory sticks and GPUs provide for possible upgrade in the future.

Chapter 6

Proposed solutions

Motivated by previous solutions, mentioned in Chapter 4, Sections 4.1.1 and 4.2.1, we propose multiple approaches to solve the problem of tracklet building. The first one, described in Section 6.1 takes advantage of the facts mentioned in Section 4.2.1, specifically that picking sufficiently small part of the object's trajectory will make the object appear as if it moved in uniform linear motion. Section 6.2 describes an algorithm which filters and validates data gathered in Section 6.1. An experimental neural network is discussed in Section 6.3 and Section 6.4 acknowledges another possible solution which has not been implemented but has been considered nevertheless.

6.1 Use of linear regression

Linear regression is a well-known statistical concept modelling the linear relationship between a scalar dependent variable y and one or more independent variables, usually denoted as x. In this thesis, we will be using a single scalar predictor variable and a single scalar response variable - simple linear regression (SLR). We chose SLR because objects in our case appear

as if they followed the rules of uniform linear motion (for details on uniform linear motion, see Chapter 4, Section 4.2). For more information about linear regression as a statistical model, see (Freedman, 2005).

As discussed in Chapter 3, Section 3.3, objects' position on the image reference frame in each image in the set of provided inputs (see Chapter 5, Section 5.1) are represented by the values x and y. The first draft of linear regression was designed and realised using these values - x as an independent value and y as a dependent value.

To successfully create a tracklet, at least three confirmed observations out of several images are required. All objects are attempted to be classified beforehand and have several mandatory attributes assigned to them. However, some of them are unidentified or potentially misidentified (unknown) and can be the object we are looking for.

First, we pair each unknown point p_1 of an object o_1 from the first image with each unknown point p_2 of an object o_2 from the second image. Then we create a line l such that $p_1,\ p_2 \in l$. The final number of existing lines after this procedure is equal to $n_1 * n_2$ where n_1 is the number of unknown points in the first image and n_2 is the number of unknown points in the second image. It is important to note that this number might be relatively high depending on the number of unknown points and therefore on the quality of the pre-processing of each image.

The point of the initial procedure described in the above paragraph is that if we removed all irrelevant points we would be left with one line placed among all the relevant points from the rest of the images (see gray and orange coloured points in Figure 6.1).

However, there are several issues with this approach. First, fake objects might appear along a line in the same fashion as the real do. Second, the

real points might have non-negligible distance from a line. Third, the first or second point might be deviated too - this would cause the line to have incorrect slope. Fourth, the real points might appear as if their acceleration was non-zero.

The last three problems are mainly caused by atmospheric fluctuations and errors in the CCD camera processing and the first one is the result of cosmic rays impacting the CCD arrays or high temperatures of the device. We provide solutions to all of these complications in the next paragraphs.

To filter out points which have almost zero probability of being real objects we introduce a distance threshold T_l which determines a maximum distance a point can have from the currently considered line.

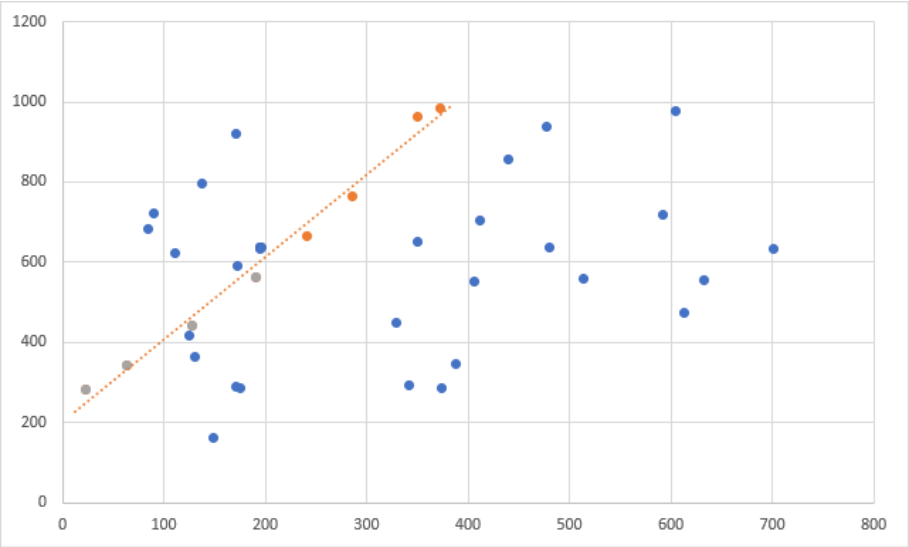


Figure 6.1: Points scattered along a trend line.

The distance of a point $p = (x, y)$ from the line $Am + Bn + C = 0$ is calculated by using formula

$$d_p = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$

We disregard a point if $d_p > T_l$, otherwise we process it in the next filter stage. The value of threshold T_l is set manually and can be changed from one batch of images to another. The higher the value the better the chance that a possibly highly deviated point will be included for further filters at the cost of considering irrelevant points as well. To mitigate this disadvantage and improve the accuracy of the process we extend our filter by calculating angle between the currently considered point and the last probable point.

Consider having two points represented by their positions $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$. We calculate angle θ between them as

$$\theta = |\arctan2(y_j - y_i, x_j - x_i)|$$

We then check whether the angle is within a manually set angle threshold T_a which provides yet another constraint on the placement of a point in the form of a theoretical two dimensional band. Introducing this restriction is crucial for removing objects which fall within the distance threshold T_l but are deviated too much. Therefore, it allows us to preserve points which conform to the line. On the other hand, setting the angle threshold T_a to a redundantly high value can have negative impact on the effectiveness of the filter.

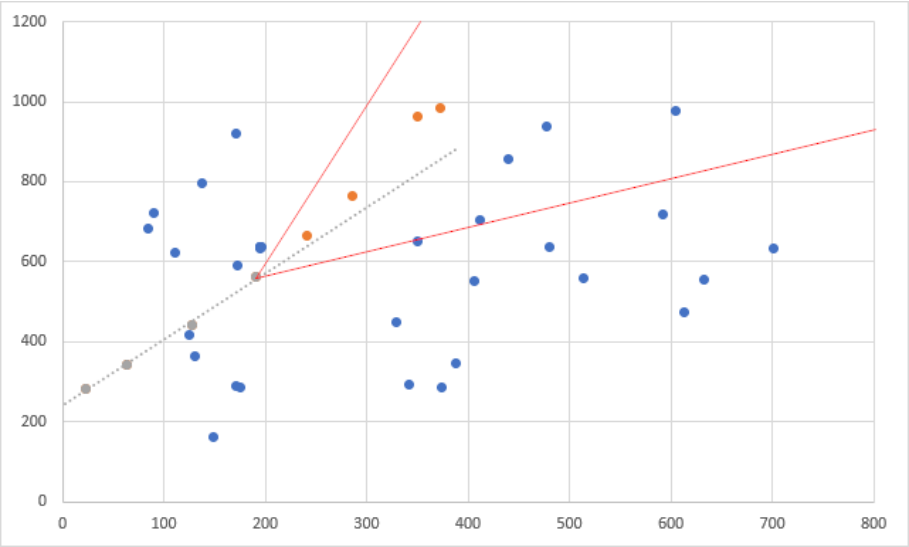


Figure 6.2: Heading of an object.

The last stage of the filter contains usage of speed s, which is calculated by using the formula

$$s = \frac{d}{\Delta t}$$

where $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ where (x_1, y_1) and (x_2, y_2) are two successive points in time and $\Delta t = |t_1 - t_2|$. We get the time $t_i,\ i \in \langle 1, N \rangle$ where N is the number of images. The purpose of the distance filter is to resolve an object's position perpendicular to its theoretical trajectory, and, to an extent, the purpose of the angle filter as well. The speed filter is mostly responsible to determine an object's position parallel to its trajectory. With regards to the previous filters we use speed threshold T_s which is different for each batch of images as it describes the approximate velocity of the object.

Only after each of these three filters has been satisfied by the currently chosen object o_i and its point p_i, we add it to the list of potential candidates.

However, there is yet another metric that helps determining the most probable object for a currently constructed tracklet. Filters only select out

points within user set boundaries - the candidates are then sorted by comparing the values assigned while processing filters. We treat first two objects o_1 and o_2 as a baseline - we calculate their speed and angle and then compare speed and angle of each successive object o_i with the baseline. The object with the closest value is then put at the front of the list and therefore implicitly marked as the most probable real object. SLR is then calculated again, containing the last added object as well. It is important to note that we add exactly one object from every image before moving to the next image. As opposed to adding every filtered object, this has proved to be a better solution.

The representation of each object o_i can be specified either by the standard coordinate system in the two-dimensional Euclidean plane or by a reference system, specifically RA/Dec (see Chapter 3, Section 3.2). The main difference and the reason why we swapped a coordinate system for a reference system is that the RA/Dec is more precise and mostly devoid of errors and deviations. The difference is best shown rather than explained - see Figure 6.3. On the left, there are shown 8 observations of the same object, represented in modified RA/Dec and on the right the same 8 observations, represented in the standard coordinate system. It is clearly visible that the deviation of each point in the right graph is more extreme than in the left graph. Acceleration of some of the points appears to be higher than zero - this is because the images were taken in different time intervals.

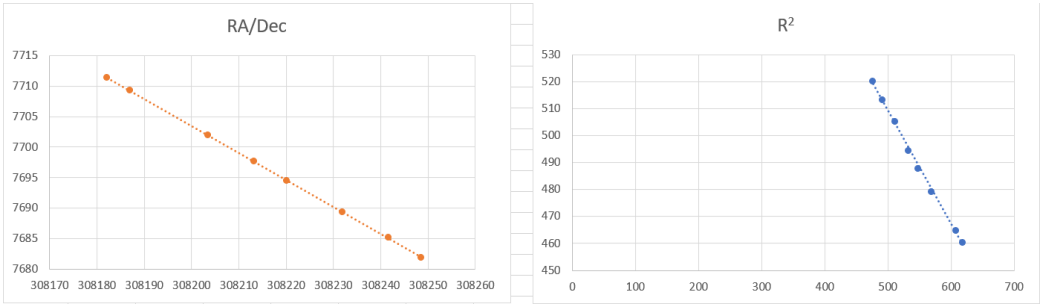


Figure 6.3: RA/Dec comparison.

We touched upon every problem mentioned before - we disregard points which are too far from the line, we correct the line by dynamically adding objects and re-calculating SLR, we filter out fake objects with nonsense speeds and angles. However, the situation when a fake object passes through all these filters and is marked as real is possible and is remedied, or rather verified, by using IOD. The final result is either one or several tracklets. As mentioned previously, tracklets have to contain at least three objects and thus we remove those which do not satisfy this condition.

6.2 Use of the Initial Orbit Determination algorithms

The theoretical details of IOD are described in Chapter 3, Section 3.4. We used a fully functional implementation of IOD from (Šilha, 2012).

The provided algorithm works as a standalone program but can be wrapped in our implementation and used for verifying data we provide to it.

In order to function correctly, the algorithm requires information about the observer and three separate objects. For the observer, the information we pass is about AGO (see Chapter 2, Section 2.1.2), specifically longitude and latitude in radians, and altitude. For the objects, we need to input RA in

radians, Dec in radians and time of the observation in Modified Julian Date (MJD). Then the orbital elements described in Chapter 3 are calculated and objects are evaluated as belonging to the same trajectory or not.

In the vast majority of cases, we are left with more than three objects from the previous phase (Section 6.1 in this chapter). In order to check them all, we need to form all possible combinations of three out of them.

To calculate all possible combinations of r objects out of n objects we use the provided implementation in the distribution of IOD, calculated from the formula

$$C(n, r) = \frac{n!}{r!(n-r)!}$$

where $r = 3$.

As an example - 11 objects in a tracklet puts the total number of combinations and runs of IOD at $C(11, 3) = 165$, which is a large number in this case.

The results of the IOD are then output into a file and we check the presence of every observation in every combination. Observations which have the highest number of appearances have the highest probability of being a real object - on the other hand, if an observation does not appear, it is considered a not real object and does not belong into a tracklet.

6.3 Use of neural network

A highly experimental part of this thesis is the use of a neural network. We used a higher-level API to TensorFlow, *tflearn*.

We built a deep neural network (DNN) and attempted to apply few activation functions (sigmoid, softmax) with multiple settings of layers and a regression layer at the end.

We attempted to train the network for various number of epochs with gradient descent algorithm but were ultimately unsuccessful. Nevertheless, we see great potential in application of neural networks in astronomical calculations.

6.4 Hough transform

Hough transform is an algorithm proposed in 1962 by Paul Hough and its main purpose is to detect shapes, such as ellipses or circles. The simplest form of Hough transform is however used to detect lines.

For each line defined as $r = x\cos\theta + y\sin\theta$ we associate a pair (r, θ) (also called a Hough space) where r is the distance of the origin O of the Euclidean space or plane to the closest point on the line and θ is the angle between the x axis and the line connecting the origin with the point. See Figure 6.4 for illustration of the Hough space.

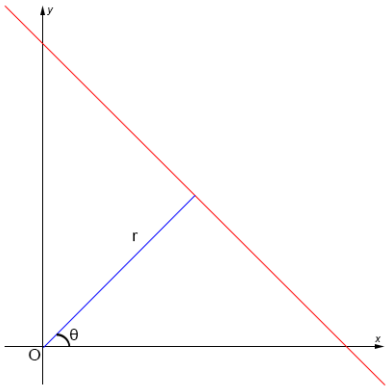


Figure 6.4: (r, θ), or Hough space visualization.

The principle of the transformation lies in finding unknown parameters - for a line, a two dimensional object, it's the mentioned pair (r, θ) - and

plotting them from points in Cartesian space into curves in the polar Hough parameter space. Lines, which are collinear in the Cartesian space produce curves in the polar Hough parameter space and intersect at a (r, θ) point.

By forming the Hough parameter space into accumulator bins, each point in Cartesian space is transformed into a curve and bins which hold corresponding values are incremented. At the end of this procedure, bins which have the highest values represent parameters of the most likely lines.

Hough transform has been not implemented because we consider it redundant, since the majority of information is provided in .cat files (see Chapter 5, Section 5.1.2) which are, in general, results of sooner image processing.

Chapter 7

Design and implementation

For implementation, we use two programming languages - Java and Python. The application runs in a console without a Graphical User Interface, on Linux (see Chapter 5, Section 5.1.3), even though due to the languages chosen it runs on MS Windows without problems as well.

We use several Java libraries and Python modules which allow for quicker and more efficient execution of many needed tasks and procedures.

Java libraries:

- Commons Math: The Apache Commons Mathematics Library - used for all calculations related to SLR and other mathematical requirements,
- eap.fits - NASA Java interface to FITS files, allowing for opening, reading and manipulating FITS files,

and Python modules:

- Astropy - module for performing astronomy and astrophysics related tasks in Python,

- numpy - module for scientific computations,
- matplotlib - 2D plotting module.

Throughout the document, you can see many classes having the prefix *SDT* - it stands for *space debris tracklet* and besides marking the classes as our own and distinguishing them from the rest, it means nothing else.

7.1 Classes

There more than 10 Java classes (along with the Main class) in our implementation.

Classes which are not covered in detail in this document are:

- Rectascension - class responsible for holding information about RA of an object and converting it to degrees
- Declination - class responsible for holding information about Dec of an object and converting it to degrees
- Type - enumeration of all possible types of an object
- Arguments - class for matching arguments in command line/terminal
- Combinations - class responsible for calculating all possible combinations of three objects out of n objects

All the other classes - their member variables and methods - are described in the next subsections.

7.1.1 algorithms.SDTLinearRegression

A class responsible for handling SLR. It contains all user-set values denoting thresholds needed for fine-tuning accuracy of the SLR described in Chapter 6, Section 6.1.

Class variable	**Description**
private static final double distanceThreshold	double representing distance threshold
private static final double angleThreshold	double representing angle threshold
private static final double speedThreshold	double representing speed threshold

Table 7.1: SDTLinearRegression class variables.

Method	**Description**
public static void perform	method responsible for performing all necessary actions, called from outside of the class
private static void findInitialRegressions	method responsible for constructing initial regressions from starting points
private static void fitPointsToRegressions	method responsible for handling all existing SLRs - modifying them, correcting them, etc.

Table 7.2: SDTLinearRegression methods.

7.1.2 algorithms.SDTOrbitDetermination

A class wrapping the already existing implementation of IOD. It handles translating information from previous steps and passing them to the IOD algorithms described in .

Class variable	**Description**
private static final double AGOLON	constant describing longitude of AGO
private static final double AGOLAT	constant describing latitude of AGO
private static final double AGOALT	constant describing altitude of AGO

Table 7.3: SDTOrbitDetermination class variables.

Method	**Description**
public static void perform	method responsible for performing all necessary actions, called from outside of the class
private static void transformObjectsIntoObservations	method responsible for translating inner objects into Observations (class in the existing implementation of IOD)

Table 7.4: SDTOrbitDetermination methods.

7.1.3 SDTObject

The SDTObject class is responsible for representing a single object in our implementation. SDTObjects are created while parsing and reading FITS and .cat files and contain several vitally important variables and methods for saving and calculating mandatory data.

Class variable	**Description**
private String fileName	String variable containing the name of the file in which the object resides (same for both FITS and .cat files)
private Type type	enumerator marking type of the object
private Rectascension rectascension	custom data structure holding information about RA of the object
private Declination declination	custom data structure holding information about Dec of the object
private double magnitude	double representing magnitude of the object
private double x	x value of the object on the reference frame
private double y	y value of the object on the reference frame
private double mjd	double containing date in the Modified Julian Date
private double xComponent	unified variable containing either RA value or x value
private double yComponent	unified variable containing either Dec value or y value
private String mpcDate	convenience String used for the MPC output

Table 7.5: SDTObject class variables.

Method	**Description**
public SDTObject	constructor for the SDTObject class
public double calculateDistanceToLine	method which takes a regression as a parameter and calculates distance of the object to the trend line
public boolean isWithinLineThreshold	method returning whether the object is in the pre-determined *distanceThreshold*
public double calculateDeltaTime	method subtracting time of an another object and returning delta time
public double calculateSpeed	method calculating speed
public boolean isWithinSpeedThreshold	method returning boolean value representing whether the object is in the *speedThreshold* variable
public double calculateHeading	method calculating heading
public boolean isWithinAngleThreshold	method deciding whether the object is in the *angleThreshold*
public int compareTo	overridden method from the Comparable<SDTObject> interface implemented by the class
public String shortString	convenience method constructing a short String containing only few information about the object
public String verboseString	convenience method constructing a long String containing significant number of information about the object
public String toString	convenience method returning either short or verbose String, depending on the internal switch
public boolean isUnidentified	method which determines whether an object is unknown or not (Types *H*, *S* and *U*
private double produceMjd	method converting the time read from a FITS file header into MJD (Šilha, 2012)
public void setTime	method setting the MJD time gotten from the *produceMJD* method and formatting the MPC time

Table 7.6: SDTObject methods.

Besides all the methods mentioned in the Table 7.6, the class also contains setter and getter for every private variable mentioned in the Table 7.5.

7.1.4 SDTBatch

A class used to hold and keep general information and data in the application, such as all existing SLRs, objects and tracklets and containing getters and setters for the mentioned data structures. Few convenience methods for better human readability are present as well.

Class variable	**Description**
private Set<SDTObject> fSet	first set of objects (from the first FITS file and the first .cat file)
private Set<SDTObject> sSet	second set of objects
private List<SDTObject> data	list containing the rest of the objects from the rest of the files
private Map<ArrayList<SDTObject>, SimpleRegression> regressions	the Commons Math library's implementation of SLR does not allow for retrieving input data so we implement this map to keep track of them
private List<SDTObject> regressionResults	list containing final tracklet
public static Map<String, Integer> objectsCount	helper map for keeping track of number of objects in each file
public static HashSet<SDTTracklet> tracklets	set containing all tracklets
public static final boolean DEBUG	switch to mark whether application should run in debug mode
public static final boolean RADEC	switch to resolve if we wish the application to use RA/Dec

Table 7.7: SDTBatch class variables.

Method	**Description**
public SDTBatch	constructor initializing empty *regressions*, *data*, *fSet* and *sSet*
public void mainDataInsert	method for inserting SDTObjects into *data*
public void firstSetInsert	method for inserting SDTObjects into *fSet*
public void secondSetInsert	method for inserting SDTObjects into *sSet*
public Set<SDTObject> getFirstSet	getter for the set containing first objects
public Set<SDTObject> getSecondSet	getter for the second set
public List<SDTObject> getMainData	getter for the *data* list
public Map<ArrayList<SDTObject, SimpleRegression>> getRegressions	getter for the *regressions* map
public void setRegressionResults	setter for inserting the final tracket
public List<SDTObject> getRegressionResults	getter returning the final tracklet
public void filterOutEmptyTracklets	method which removes tracklets which contain less than three objects and therefore does not conform to the general definition of tracklets
public String toString	convenience method returning *data* list in String format

Table 7.8: SDTBatch methods.

7.1.5 SDTTracklet

SDTTracklet is a class which holds objects in a complex structure designed specifically to allow for fine balance between convenience and performance while handling said objects and accommodating for the SLR algorithm performed in the class *SDTLinearRegression* described in Subsection 7.1.1.

The class contains two member variables, specifically
public ArrayList<ArrayList<Map<SDTObject, Double>>> objects
and public double valueToCompareWith.

objects is a two-dimensional list. The first dimension's length corresponds to the number of all images in the currently processed batch and therefore all possible number of objects a tracklet can contain. The second dimension can be imagined as *bins* which contain every possible object which passes through the filters. The objects in bins are represented as a map with *SDTObject* as the key and Double as the explicit value assigned while calculating a SLR. For more information on this process concerning the filters and the explicit value, see Chapter 6, Section 6.1. Figure 7.1 illustrates the *objects* list.

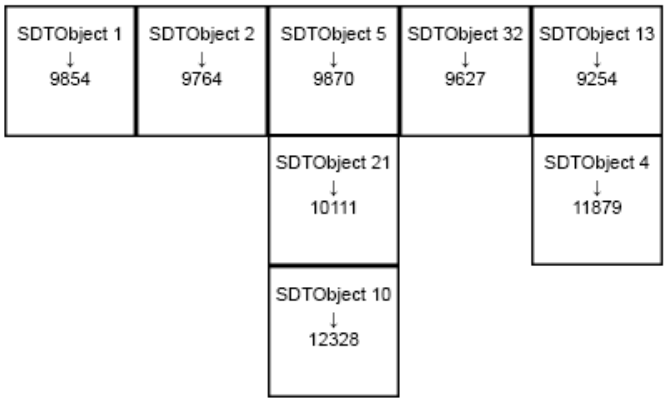


Figure 7.1: two-dimensional *objects* list.

Method	**Description**
pubic SDTTracklet	constructor initialising *objects* list
public SDTObject getLastPoint	method returning the last valid most probable point to insert to the SLR
public void sortLast	method sorting the last non-empty bin according to the Double value in *objects* list
private ArrayList<HashMap<SDTObject, Double>> sort	private method for sorting used by *sortLast*
public SDTObject getMostProbableObject	method returning the first object in the *bin*
public ArrayList<HashMap <SDTObject, Double>> getListOfObjectsOnIndex	method responsible for returning the right *bin*

Table 7.9: SDTTracklet methods.

7.1.6 SDTFileHandler

As the name indicates, this class is responsible for handling files, both FITS and .cat (see Chapter 5, Section 5.1). We use the mentioned library eap.fits for opening FITS files and reading headers.

Method	**Description**
static void readFiles	a method responsible for performing all necessary actions and using all utility methods in this class, called from outside of the class
private static void insertBatch	method adding objects to the correct data structure (either *fSet*, *sSet*, or *data*)
private static Map<File, File> mergeCATWithFITS	method internally merging corresponding .cat files with FITS files
private static List<File> getCatFiles	method collecting all the .cat files to be merged
private static FitsHeader getFitsHeader	method returning the header of a FITS file
public static void processEntry	the core method responsible for parsing .cat files, reading FITS headers and creating SDTObjects

Table 7.10: SDTFileHandler methods.

Public static methods from the class are used as the first step of the application. Data provided in the files on the disk are read and sent to the next steps.

7.1.7 SDTOutput

A class which handles output of the algorithms. The class reads output of the IOD (see Chapter 6, Section 6.2) and creates internal text file with the results, text file in Bern format (see Chapter 8, Section 8.2) and text file in MPC format (see Chapter 8, Section 8.1.1).

7.2 Python script

The included Python script is used to read results from the internal text file. The parsed information about objects is then used to create a line which begins at the first object and ends at the last object.

Moreover, FITS files are read by the *Astropy* module and stacked on top of each other via *numpy* module and the final image is created (see Chapter 8, Section 8.3).

The minimal (*vmin*) and maximal (*vmax*) luminance of the final image is left to be determined from one case to another. In the image referenced in the paragraph above *vmin* is at 10,000 and *vmax* at 25,000.

Chapter 8

Results

8.1 AIUB and MPC formats

This section provides a short insight over standardised output data formats used in this thesis and encouraged by multinational agencies and foreign universities.

8.1.1 MPC

The Minor Planet Center (MPC) is a worldwide organization for Astronomical purposes. The organization is in charge of collecting observational data for minor planets, comets and satellites of major planets and belongs under International Astronomical Union (Minor Planet Center, 2016).

8.1.2 MPC format

The officially MPC endorsed format is a text file with exactly prescribed form. The format is divided into columns where each column equals one character in a text file. The columns are explicitly set, therefore the format

does not contain any headers. There are four kinds of formats - for minor planets, for comets, for natural satellites and for minor planets, comets and natural satellites. In our case, we are using the last type (Minor Planet Center, 2016).

The designation for every column is as follows:

- columns 1-12 – designation of the observation
- column 14 – NOTE 1 - an alphabetical publishable note (for example *S* for "poor sky", or *K* for "stacked image"
- column 15 – NOTE 2 - indicates how observation has been made (for example *C* for "CCD")
- columns 16-32 – DATE OF OBSERVATIONS - contains the date and time in UTC of the mid-point of observation; the format of this column is "YYYY MM DD.dddddd"
- columns 33-44 – OBSERVED RA - contains observed right ascension; the format of this column is "HH MM SS.ddd"
- columns 45-56 – OBSERVED DECL - contains observed declination; the format of this column is "sDD MM SS.dd" ("s" for sign)
- columns 66-71 – OBSERVED MAGNITUDE AND BAND - the observed magnitude of an object and band in which the measurement was made
- columns 78-80 – OBSERVATORY CODE - the code of the observatory in which observation was made (AGO has code 118)

8.2 AIUB format

Astronomical Institute of University of Bern (AIUB) has its own format for tracking space debris objects.

The AIUB format is as follows:

- station name - name of the observatory; in our case "AGO_70"
- object name - name of the observed object, typically contained in the name of the file
- date - in the MJD format
- right ascension - in "hh.mmssssss" format
- declination - in "dd.mmssssss" format
- magnitude - in "RR.RR" format
- file name - name of the original .fits file

The list above describes a single line, while the file consists of many of these lines and therefore objects.

8.3 Visualization of results

The Python script described in Chapter 7, Section 7.2 creates an image displayed in Figure 8.1.

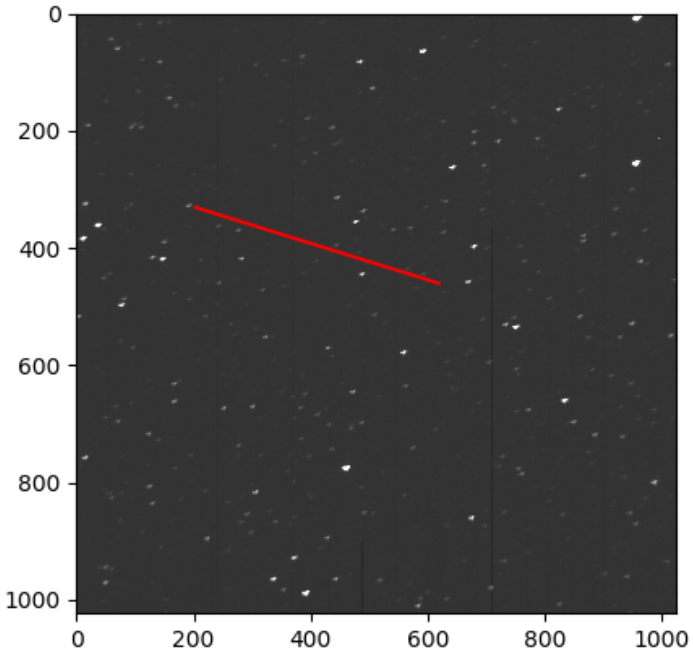


Figure 8.1: The resulting image.

In the figure, we can see a recently observed (year 2017) object *PR25*. The red line represents the calculated trajectory of the object. As for the objects we marked as real, see Figure 8.1.

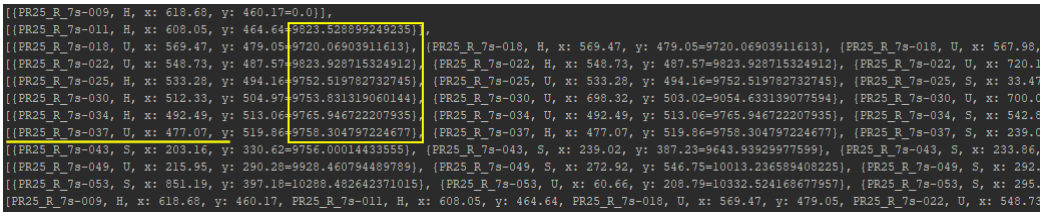


Figure 8.2: The final objects.

Objects above the yellow line are all confirmed as real objects. Some objects above the line are marked as unknown - in this case, this is irrelevant and is caused by incorrect input data. As is visible, the second object to the right is the next object in the *bin* mentioned in Chapter 7, Subsection 7.1.5, and is identical to the one marked as unknown and is marked as real. More to the right, there are other objects which are considered less likely to be a real object and therefore are further in the *bin*.

The image perfectly shows how powerful filters and thresholds are; values in the yellow rectangle represent the baseline values compared in the algorithm and are mentioned in Chapter 6, Section 6.1. Also, as is visible from the last three entries, some fake objects might pass through filters and thresholds and are left for IOD, mentioned in Chapter 6, Section 6.2, to remove.

The image further represents that now we have successfully created a tracklet from 8 objects out of 11 images, which is 72% rate of success and can be improved by fine tuning threshold values. We have similar, or even higher, success rate on other tested objects not included in this document.

Chapter 9

Conclusion

In this thesis we have presented the need for the construction of tracklets, base knowledge concerning space debris objects and their physical features and their use in today's astronomical scientific community. We have analysed currently professionally used algorithms for tracklet building employed in many world's respected establishments.

We have successfully designed and built our own relatively simple, yet powerful and elegant system for filtering out fake objects, creating tracklets and displaying results in a coherent, both visual and textual, way.

The system will be implemented on the server in a pipeline at the Astronomical and geophysical observatory in Modra where it will be further tested, improved and used to construct tracklets from the images of the night sky.

The room for improvement lies mainly in the machine learning part. While we were unsuccessful in training the neural network and directing it in such way that it would produce reasonable results, we have learned invaluable lessons which will be employed in next phases of the life-cycle of this system. Other improvements can be made in calculating thresholds and thus refining the SLR algorithm even more and producing cleaner results.

Bibliography

(2016). *Definition of the Flexible Image Transport System (FITS)*. International Astronomical Union, IAU. Version 4.0.

ASRC Federal Technical Services CCSDS (2017). CCSDS. `https://public.ccsds.org/about/default.aspx`, accessed at 2018.

Astropy Developers, T. (2018). Fits file handling. `http://docs.astropy.org/en/stable/io/fits/index.html`, accessed at 2018.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517.

ESA (2017). Space debris: the ESA approach. Brochure. `http://www.esa.int/About_Us/ESA_Publications/ESA_Publications_Brochures/ESA_BR-336_Space_Debris_The_ESA_Approach`, accessed at 2018.

ESA Science (2013). ESA optical ground station. `http://sci.esa.int/sci-fmi/36520-optical-ground-station/`, accessed at 2018.

Freedman, D. (2005). *Statistical Models: Theory and Practice*. Cambridge University Press.

KAFZM FMFI UK (2017). Department of astronomy, physics of the earth and meteorology. `http://www.daa.fmph.uniba.sk/`, accessed at 2018.

Kessler, D. J. and Cour-Palais, B. G. (1978). Collision frequency of artificial satellites: The creation of a debris belt. 83:2637–2646. Provided by the SAO/NASA Astrophysics Data System.

Klinkrad, H. (2006). *Space Debris: Models and Risk Analysis*. Astronautical Engineering. Springer.

Kubica et. al., J. (2007). Efficient intra- and inter-night linking of asteroid detections using kd-trees. paper. Icarus 189 (2007) 151–168.

Massachusetts Institute of Technology (2009). MIT lab notes. `https://www.ll.mit.edu/publications/labnotes/Looking_Back_3.pdf`, accessed at 2018.

Miller, F., Vandome, A., and McBrewster, J. (2009). *Kd-tree*. VDM Publishing.

Minor Planet Center (2016). MPC format. `https://minorplanetcenter.net/iau/info/OpticalObs.html`, accessed at 2018.

Montenbruck, O. and Gill, E. (2005). *Satellite Orbits: Models, Methods, Applications*. Springer.

NASA (2018). Nasa image gallery. `https://www.orbitaldebris.jsc.nasa.gov/photo-gallery.html`, accessed at 2018.

Oda et. al., H. (2013). Optical observation, image-processing, and detection of space debris in geosynchronous earth orbit. paper.

on the Peaceful Uses of Outer Space. Scientific, U. G. A. C. and Subcommittee, T. (1999). *Technical report on space debris: text of the report adopted by the Scientific and Technical Subcommittee of the United Nations Committee On the Peaceful uses of Outer Space*. United Nations.

Šilha, J. (2012). *Identification of the Artifical Objects in Close Vicinity of the Earth*. PhD thesis, Comenius University.