Chair of Astronautics
Prof. Prof. h.c. Dr. Dr. h.c.
Ulrich Walter

Master's Thesis

Determination of the Spatial Coverage of the AllSky7 Fireball Network

MA-2022/08

Author:

Yannic Heidegger

Supervisor:	Prof. Prof. h.c. Dr. Dr. h.c. Ulrich Walter Chair of Astronautics Technische Universität München

Erklärung

Ich erkläre, dass ich alle Einrichtungen, Anlagen, Geräte und Programme, die mir im Rahmen meiner Semester- oder Masterarbeit von der TU München bzw. vom Lehrstuhl für Raumfahrttechnik zur Verfügung gestellt werden, entsprechend dem vorgesehenen Zweck, den gültigen Richtlinien, Benutzerordnungen oder Gebrauchsanleitungen und soweit nötig erst nach erfolgter Einweisung und mit aller Sorgfalt benutze. Insbesondere werde ich Programme ohne besondere Anweisung durch den Betreuer weder kopieren noch für andere als für meine Tätigkeit am Lehrstuhl vorgesehene Zwecke verwenden.

Mir als vertraulich genannte Informationen, Unterlagen und Erkenntnisse werde ich weder während noch nach meiner Tätigkeit am Lehrstuhl an Dritte weitergeben.

Ich erkläre mich außerdem damit einverstanden, dass meine Master- oder Semesterarbeit vom Lehrstuhl auf Anfrage fachlich interessierten Personen, auch über eine Bibliothek, zugänglich gemacht wird, und dass darin enthaltene Ergebnisse sowie dabei entstandene Entwicklungen und Programme vom Lehrstuhl für Raumfahrttechnik uneingeschränkt genutzt werden dürfen. (Rechte an evtl. entstehenden Programmen und Erfindungen müssen im Vorfeld geklärt werden.)

Ich erkläre außerdem, dass ich diese Arbeit ohne fremde Hilfe angefertigt und nur die in dem Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Garching, den ______________

____
Unterschrift

Name: Yannic Heidegger
Matrikelnummer: 03721495

Acknowledgements

First of all, I would like to thank Michael Frühauf and Detlef Koschny for their excellent guidance and support during the work. I would also like to thank all the members of the AllSky7 network who made this work possible. A special thanks goes to Mike Hanky, who provided the code for the AllSky7 network and was always a great help with questions.

Zusammenfassung

Ziel dieser Studie war die Entwicklung einer Software zur Bestimmung der räumlichen Abdeckung des AllSky7-Fireball-Netzwerks. Dazu wurde ein Ansatz gefunden, um die Abdeckung der einzelnen Kameras zu bestimmen und dann ein Abdeckungsprofil für das gesamte Netzwerk zu erstellen. Um das Gebiet über Europa zu diskretisieren, wurde ein Raster erstellt, das in der Länge einen Bereich von −20° to 37° und in der Breite einen Bereich von 27° to 67° umfasst. Bei einer Beobachtungshöhe von 100 km deckt das Netz 25.23 % des verwendeten Rasters und 0.93 % der Welt unter idealen Bedingungen ab. Dann wurde ein Modell für die Helligkeitsabnahme aufgrund der Lichtauslöschung durch die Atmosphäre und die zunehmende Entfernung bei niedrigen Höhenwinkeln eingeführt. Die Abdeckung wurde durch einen Grenzwinkel für jede Kamera in Abhängigkeit von der Höhe der Station und einer bestimmten Helligkeit im Zenit angepasst. Die resultierende Abdeckung des Gitters beträgt 19.4 % und 0.72 % der Welt. Unter Berücksichtigung verschiedener Meteorebenen verringert sich die abgedeckte Fläche um 23.87 % bei einer Beobachtungshöhe von 80 km im Vergleich zur Referenzhöhe von 100 km. Bei einer Beobachtungshöhe von 120 km erhöht sich Abdeckung um 28.08 %. Die Ergebnisse zeigen, dass die räumliche Abdeckung über Mitteleuropa, insbesondere Deutschland, auf einer Höhe von 100 km sehr gut ist. Im Gegensatz dazu ist die Abdeckung im Norden und Osten Europas nicht ausreichend.

Abstract

This study aimed to develop software to determine the spatial coverage of the AllSky7 Fireball Network. Therefore, an approach was found to determine the coverage of individual cameras and then create a coverage profile for the entire network. To discretize the area over Europe a grid was created ranging from −20° to 37° in longitude and from 27° to 67° in latitude. At an observation altitude of 100 km, the network covers 25.23 % of the used grid and 0.93 % of the world under ideal conditions. Then a model for the brightness reduction due to light extinction by the atmosphere and the increasing distance at low elevation angles was introduced. The coverage was adjusted by a limiting angle for each camera depending on the station's altitude and a specified magnitude at the zenith. The resulting coverage of the grid is 19.4 % and 0.72 % of the world. Considering different meteor levels, the covered area decreases by 23.87 % at an observing altitude of 80 km in comparison to the reference altitude of 100 km. At an observing height of 120 km, the coverage area increases by 28.08 %. The results showed that the spatial coverage over central Europe, especially Germany, is quite good for an altitude of about 100 km. In contrast, the coverage in Europe's north and east is insufficient.

Contents

List of Figures

List of Tables

Symbols and Formulas

Symbol	Unit	Description	Symbol	Unit	Description
α	°	right ascension	δ	°	declination
γ	°	azimuth	ϵ	°	elevation
λ	°	longitude	ϕ	°	latitude
R	km	mean Earth radius	H	km	altitude
n_s	μm^{-2}	refraction index	X		air mass
m		magnitude	d	km	distance

1 Introduction

The threat of an impact by a massive asteroid on Earth is minor, but the aftermath of such an impact can be devastating. So far, the most likely theory for the extinction of the dinosaurs is such an impact by an asteroid about 10 km in size [1]. Every day micrometeorites fall on the Earth. Most of them are being evaporated in the atmosphere but around 10 % of them are reaching the Earth's surface [2]. Moreover, even some more enormous impacts of asteroids happened in recent history. The most recent impact was by an asteroid of about 20 m in diameter over the city of Chelyabinsk in 2013. Although it did not hit the ground, it exploded in 30 km height and created a shock wave that caused several injuries and damaged buildings. Another big event happened in 1908, when an object of approximately 30 m in diameter exploded in the sky over Tunguska in Russia, releasing energy equal to 1000 Hiroshima bombs. [3] If such an event happens over a big metropolis, the effects would be catastrophic. Space organizations like NASA and ESA are tracking such Near-Earth Objects (NEOs) and maintaining a risk list of all known objects with a higher probability than zero to hit the Earth [4]. However, not all objects can be seen from Earth due to their small size or position to the sun. Furthermore, it is hard to track the entire sky. Therefore, more information about the amount, size, and trajectories of asteroids and meteoroids is needed to calculate the risk of possible impacts. Amateur astronomers significantly contribute to this data.

1.1 The AllSky7 Fireball Network

The AllSky7 fireball Network was founded as a non-commercial organization to track the sky and record fireball events. It consists of owners of AllSky7 Fireball camera systems to support scientific analyses and spread information, data, and recordings of meteors and fireballs. The Network started in Germany and is now spread all over Europe, having stations in Austria, Belgium, Switzerland, Germany, Denmark, Spain, France, Hungary, Netherlands, Norway, Ireland, Italy, Poland, Portugal, Slovenia, Slovakia, United Kingdom, and the United States / Iowa [5]. To cover the entire sky above an observation site, the AllSky7 fireball camera System consists of seven highly sensitive NetSurveillance NVT cameras. Five cameras point at an elevation of 25° above the horizon, whereas two cameras point at an elevation of 70° in the northern and southern direction [5]. The camera system is shown in figure 1–1.

1.2 Determination of the Flux Density

To contribute to ESA's work on NEOs, the LRT is working on determining the flux density of meteoroids and asteroids in the size range below tens of meters. As part of the AllSky7 network, the LRT can analyze the footage to get the required parameters. These parameters are the spatial coverage, the time of clear sky, and the number of fireballs. Although, in theory, a camera system can observe the entire sky, there are often objects in the field of view. In addition, the observable sky might not be fully

(a) Housing (b) Cameras

Fig. 1–1: AllSky7 Camera System [5]

visible over the observation time due to cloud cover. Moreover, an observed fireball must be detected by at least two cameras to confirm its existence and to be able to compute the trajectory of the object. To calculate the object's size, it is also necessary to determine the actual magnitude of the fireball. Because the value can be inaccurate due to the characteristics of a camera when observing a moving object. [6]

1.3 Scope of this Thesis

As a first step, the mentioned problems have to be solved. This study aims to develop software to determine the spatial coverage of the AllSky7 Fireball network. For this purpose, an approach is found to determine the spatial coverage of each station by analyzing the calibration images. The obtained data is then used to calculate the spatial coverage of the entire network using the developed software. To account for brightness reduction at low elevation angles, a model is proposed to determine a minimum elevation angle for each station as a function of a certain magnitude at zenith and the station's elevation above sea level. The spatial coverage is then analyzed for the influence of brightness degradation and different observation levels. Finally, the results are discussed, and an outlook for further research is given.

2 Basics of meteor observation

To improve knowledge of the meteorite and asteroid population in the solar system, these objects must first be studied. Most is known about the larger objects in the hundreds of meters to kilometers range. Because of their size, they reflect enough light from the Sun to be observed from Earth. The smaller they are, the fainter they appear and the more difficult it becomes to detect them from Earth. For very small objects such as micrometer-sized planetary dust, the number can be extrapolated from a small detection range because they frequently strike the Earth. The population of meteoroids ranging in size from tens of meters to millimeters is not yet well known. They are usually perceived as meteors only when they strike the Earth's atmosphere. This chapter explains the basic definitions and describes the details of observation.

2.1 Asteroids, Meteoroids and Interplanetary Dust

The solar system consists mostly of the Sun and eight planets orbiting the Sun. The space in between is mostly vacuum. However, there are also some dwarf planets such as Pluto, Ceres, and Eris, as well as cosmic debris which is, in most cases, leftovers from the solar system's formation. To distinguish these terms, some definitions must be given here. A planet is defined by the International Astronomical Union as a celestial body that orbits the Sun, is large enough to form a round shape due to its mass, and is capable of clearing its orbit of cosmic debris [7]. Therefore, Pluto has lost its status as a planet because, despite its round shape, it is not large enough to clear its orbit. Cosmic debris can be divided into asteroids, meteoroids, and interplanetary dust. However, the classification is not always precise. Celestial objects significantly smaller than a planet but larger than a meteoroid are called asteroids. The size at which asteroids can still be detected from Earth has been proposed as a lower limit. This gives a rough size range of about 1000 km to a few meters in diameter. Most known asteroids are located in the asteroid belt between Mars and Jupiter. Most known asteroids are located in the asteroid belt between Mars and Jupiter. However, due to gravitational forces, they can break out of the belt and change their orbit, posing a threat to Earth. Objects that come closer to Earth than 1.3 au are called near-Earth objects. All particles smaller than 10 µm are called interplanetary dust. Consequently, a meteoroid is classified as an object with a diameter between 1 m and 10 µm. Meteoroids may be parts of asteroids separated by collisions or gravitational forces or remnants of planet formation. [8] For simplicity, the term meteoroid is used in the following sections to describe the events of an impact on Earth.

2.2 Meteors, Fireballs and Bolides

The following section is largely based on Ceplecha's description of meteor phenomena, and their phases [9]. A meteor is a luminous phenomenon caused by the heating of a meteoroid or an asteroid as it enters the Earth's atmosphere. A logarithmic scale, magnitude, is used to describe the brightness of celestial objects. The brightest stars in the sky are classified as having a magnitude of 1, and the faintest stars visible to the naked eye are classified as having a magnitude of 6. The scale ratio is 2.512, meaning that a star is 2.512 times brighter than a star in the next fainter category. Much brighter objects have a negative magnitude, for example, the full moon has a magnitude of −13, and the Sun shines with a magnitude of −27. The brightness of a meteor depends on the size and velocity of the incoming object. The limiting size for a meteoroid to produce a meteor is about 0.01 mm. If the meteor is brighter than −3 in magnitudes, it is also called a fireball. At brightnesses of about −17 magnitudes, when it can be seen by satellites in Earth orbit, it is called a super bolide. [6] Solar system meteoroids can have velocities between 11.2 $\mathrm{km\,s^{-1}}$ (escape velocity of Earth) and 72.8 $\mathrm{km\,s^{-1}}$ (velocity of meteoroid at Earth's perihelion: 42.5 $\mathrm{km\,s^{-1}}$ plus velocity of Earth at perihelion: 30.3 $\mathrm{km\,s^{-1}}$). Therefore, most meteor impacts on Earth are due to collisions rather than gravity. The direction from which a meteoroid comes is called a radiant. Most meteors come from meteor showers like the Perseids, where all meteors belong to the same stream and have the same radiant. Only a small part are so-called sporadic meteors, which do not belong to any meteor shower.

2.2.1 Meteor Phases

Ceplecha describes the phenomenon of a meteor in five phases: Orbital motion, preheating, ablation, dark flight, and impact [9].

2.2.1.1 Orbital Motion

The trajectory of a meteoroid is primarily influenced by the Sun's gravity. However, its trajectory can be disturbed by the gravity of larger bodies, such as planets or minor planets, as well as by collisions or the irradiation of cosmic rays.

2.2.1.2 Preheating

When the meteoroid enters the atmosphere, the surface is strongly heated by collisions with air molecules. Preheating begins at an altitude of about 300 km to 100 km. Except for very small bodies, the core of the body remains unheated. The preheating process usually lasts only a few seconds. At a temperature of 900 K, ablation begins. Due to the high-temperature gradients, the meteoroid may fall apart. The dominant heat transfer mechanism for small bodies is radiation, while for larger bodies, it is more likely to be conduction.

2.2.1.3 Ablation

Ablation begins with the fragmentation of the body at lower temperatures. When the surface temperature reaches 2200 K, the material begins to melt. At even higher temperatures of about 2500 K, vaporization occurs. The hot gases fill the air around the meteoroid, and light is emitted as the particles are de-excited by radiation. The temperature remains relatively constant at this point because most of the kinetic energy is lost through ablation. Smaller objects are little affected by deceleration because the body is consumed by ablation before it can be decelerated. Larger meteoroids can persist

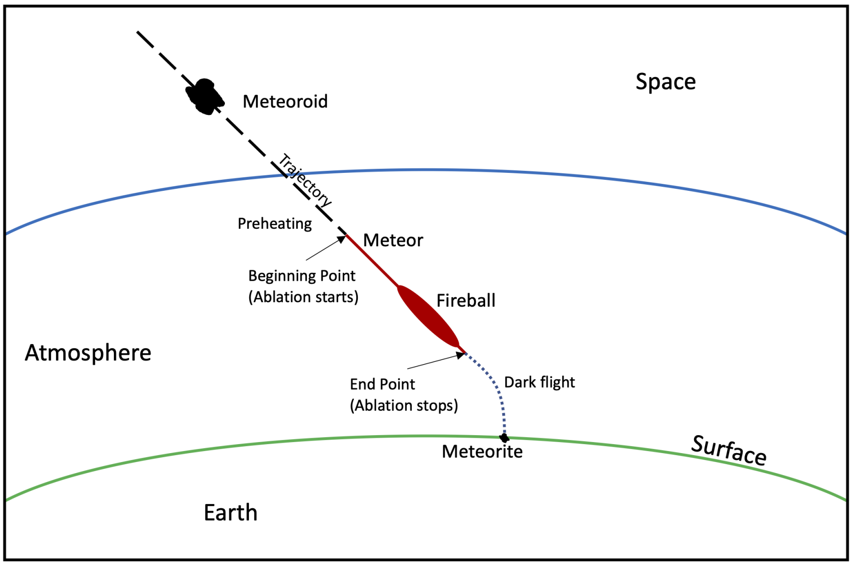


Fig. 2–1: Meteor Phases based on Ceplecha [9]

up to a velocity of about 3 km s^{-1}. The high temperature cannot be maintained at this speed, and the meteor enters the dark flight phase. The phase of a visible meteor starts usually at 110 km and ends at 80 km height [6].

2.2.1.4 Dark Flight

In this phase, the velocity of the meteoroid is too low to heat the surface above the melting temperature due to air friction. The surface of the meteoroid is now rapidly cooled and forms a crust. As the velocity decreases, the body goes into free fall, which can last for several minutes. Since the meteoroid is no longer visible during this phase, it can be complicated to calculate its trajectory.

2.2.1.5 Impact

The velocity at impact with Earth ranges from 10 m s^{-1} for smaller bodies of 10 g to 100 m s^{-1} for larger objects of 10 kg final mass. The impact of average meteoroids forms a pit about as large as itself. If the object is large enough that the ablation phase continues to the ground, a huge box may form due to the explosive release of kinetic energy. Figure 2–1 shows the different meteor phases.

2.2.2 Influence of the Velocity

Velocity greatly affects the mass loss of the meteoroid during the flight phase. This process is called ablation and is proportional to v_∞^{-6}, where v_∞ is the initial velocity before entering the atmosphere. Thus, the higher the velocity, the larger the meteoroid must be to reach the ground. When a meteoroid hits the Earth's surface, it is called a meteorite. An upper limit for the fall of a meteorite is approx. 30 km s^{-1}. If the meteoroid has a higher initial velocity, the body will most likely vaporize before it hits the ground.

2.2.3 Influence of the Size

There are four types of meteor phenomena, depending mainly on the object's mass. The following distinctions apply to a meteorite with an initial velocity of 15 km s^{-1} and a bulk density of 3500 kg m^{-3}.

2.2.3.1 Meteors

A typical meteor has a brightness between 6 and 2 mag. The size ranges from 0.05 mm to 20 cm. The size limit visible to the naked eye is about 0.01 mm. For bodies larger than 0.05 mm, only the surface down to tenths of a millimeter is heated by collisions with the air molecules. At a temperature of 2200 K, the surface layer begins to sublime, and the surrounding air fills with vapor particles. The excited atoms emit their energy through radiation, producing the visual effect of a meteor. After a few kilometers, the entire body mass has evaporated without losing much of its velocity, and the visible light fades.

2.2.3.2 Fireballs

Meteors with a brightness of –3 mag or higher are called fireballs [6]. Such bright phenomena are caused by objects larger than 20 cm. At this size, the body does not lose all its mass in the ablation phase. The remaining mass is decelerated to the critical velocity of 3 km s^{-1}, and the surface temperature drops below 2200 K. At this temperature, evaporation no longer occurs, and the meteor light goes out. The molten surface cools and forms a crust. The body then enters a dark flight phase and slows to free fall speed. The remnant falls to the ground as a meteorite.

2.2.3.3 Bolides and Superbolides

A fireball is classified as a bolide if the brightness exceeds −14 mag and as a superbolide, if it exceeds −17 mag. [8]. In this very rare case, a body of several meters in size collides with the Earth. Because of its enormous mass, it cannot be decelerated below the critical velocity before hitting the ground. Consequently, the light does not end in the flight phase, and the asteroid impacts the Earth's surface at several kilometers per second, forming a meteor crater. If the object is unstable, it can also explode in the air before hitting the ground. This happened, for example, in 2013 over Chelyabinsk, where a super bolide of about 20 m exploded at about 30 km altitude. The shock wave released energy equal to that of 30 Hiroshima bombs.

2.2.3.4 Meteoric Dust Particles

Small dust particles of a few hundredths of a millimeter decelerate quickly in high atmospheric layers. Therefore, the particles cannot reach the evaporation temperature, and no meteor phenomenon occurs. The dust settles unchanged on the surface of the earth.

2.3 Types of meteor observation

There are several ways to observe meteors, which are well described in the International Meteor Organization's Meteor Observing Handbook by Rendtel and Arlt [6]. The following section is primarily based on this manual.

2.3.1 Visual observation

The oldest method is visual observation with the naked eye. The advantage over the other methods is that almost no equipment is needed for observation. However, it is necessary to write down all essential parameters such as time, position, speed, and brightness by hand. Therefore, the accuracy of the recorded data can vary greatly. Especially the brightness is difficult to determine. The only reference for brightness is a star in the same region of the observed sky. The limiting magnitude is about +6 mag or +5 mag and depends on the capabilities of the observer's eye. However, in the case of heavy meteor activity, it may be challenging to capture all essential parameters in time.

2.3.2 Photographic observation

Another method is photographic observation. The most convincing argument is the accuracy of position determination required to accurately calculate the meteor's trajectory, velocity, mass, and spectrum. A significant disadvantage of the photographic method is the low limiting magnitude of about +1. Another limiting factor is the focal length f. The limiting magnitude is inversely proportional to the focal length f. The higher the focal length, the narrower the image. So the exposure time is shorter because the meteor moves faster across the pixel. Much space is mapped onto a few pixels at large angles, so the meteor path is relatively short.

2.3.3 Video observation

For automated observations, the video-based approach is the best method for meteor detection. It combines the advantages of photographic and visual observation. The determination of the essential parameters is sufficiently accurate, while the limiting magnitude is about the same as with the naked eye. Moreover, there are no physiological limitations, such as fatigue.

2.3.4 Radar observation

Radio observations achieve the highest detection strength. This method can be used to detect smaller objects and is independent of weather or time of day. However, it is more difficult to interpret the data.

2.4 Celestial Coordinates

Right ascension α and declination δ are geocentric coordinates. The reference point is the vernal equinox, i.e., the Sun's position at the beginning of spring. Right ascension

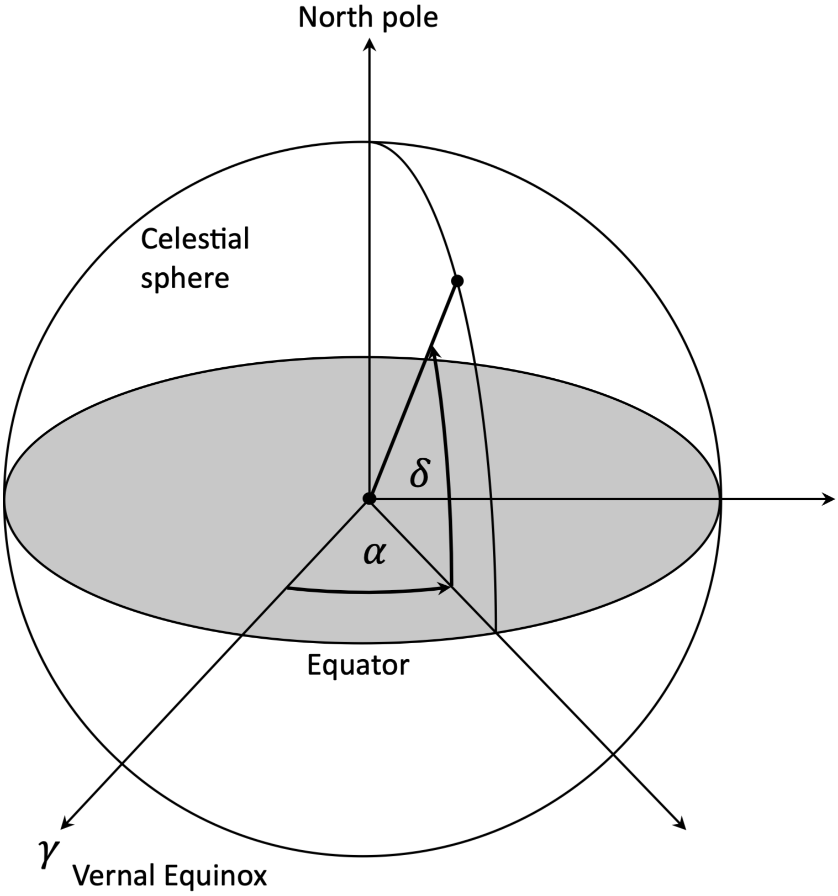


Fig. 2–2: Right Ascension and Declination

gives the angle between the vernal point and the observed object in the equatorial plane. The declination gives the angle from the equatorial plane to the observed object. A sketch of the coordinate system of right ascension and declination is shown in Figure 2–2.

2.5 Horizontal Coordinates

The horizontal coordinate system is centered at the observer. Azimuth and elevation define a point in the sky from the observer's position on Earth. Azimuth is the tangent to the Earth's surface and indicates the direction, while elevation indicates the height of the defined point in the sky. Both values are measured in degrees. Azimuth starts north and increases clockwise from 0 to 360 degrees. Elevation starts at the horizon and increases vertically from 0° to 90° to the zenith. The principle is shown in figure 2–3.

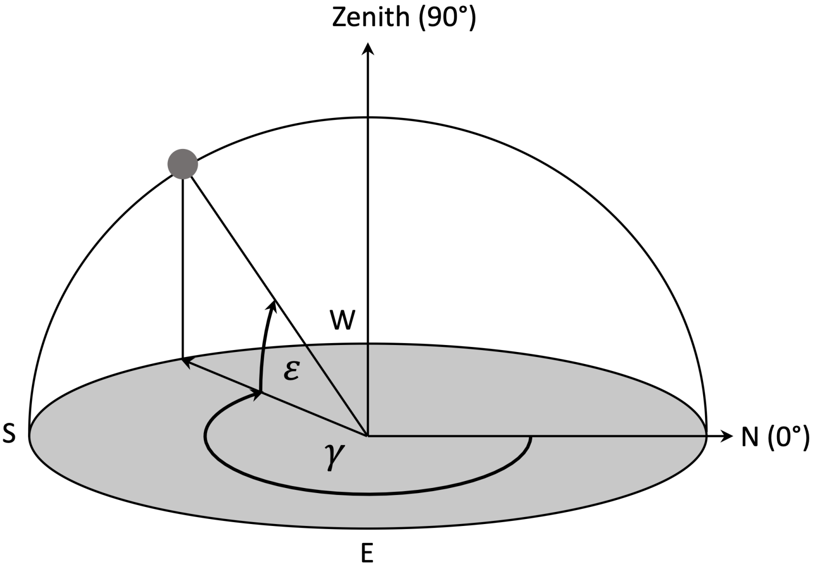


Fig. 2–3: Azimuth and Elevation

2.6 Coordinate Transformation

The transformation of right ascension and declination to azimuth and elevation is well described by Walraven [10]. First, the sidereal time must be calculated. A solar day is the time in which the celestial sphere revolves once around the Earth. Since the Earth moves around the Sun, the sidereal day is slightly shorter than a normal day. The sidereal time in Greenwich (ST) in hours can be calculated with the following formula:

$$ST = 6.720165 + 24*\left(\frac{d}{365.25} - (y - 1980)\right) + 0.000001411 \cdot d, \qquad (2\text{–}1)$$

where d is the days since the reference epoch J1980 plus the local time expressed in days and y is the current year. Afterward, the local standard time LST can be calculated as well:

$$LST = ST - \lambda, \qquad (2\text{–}2)$$

where λ is the local longitude in hours ($1\,\text{h} = 15°$). Finally, the local lateral time S can be calculated as follows:

$$S = LST + 1.0027379 \cdot (LST + Z - C), \qquad (2\text{–}3)$$

with the local time zone Z and C being either zero or one, depending on whether daylight saving time is in effect or not. With the local solar time, the hour angle HA can be calculated, which gives the distance from the zenith to the observed object in hours, minutes, and seconds:

$$HA = \alpha - S, \tag{2–4}$$

Finally, the azimuth γ and elevation ϵ can be determined using the following expressions:

$$z = arccos(sin(\phi)sin(\delta) + cos(\phi)cos(\delta)cos(HA)), \tag{2–5}$$

$$\epsilon = 90^\circ - z, \tag{2–6}$$

$$\gamma = arcsin\left(cos(\delta)\frac{HA}{sin(z)}\right), \tag{2–7}$$

where z is the angular distance from the zenith to the object and ϕ is the latitude of the observer.

2.7 Plate Solving

Plate Solving is a method of determining a camera's alignment position and lens distortion. It used to be done by hand by comparing the positions of stars to a star chart. With the advent of computers, plate solving is now performed by algorithms. Using a star catalog, the algorithms can detect star patterns and determine the right ascension and declination of the center of the image. By calculating the deviation of the star positions, a mathematical model can be derived that describes the lens distortion. Using this model, the coordinates of each pixel can be determined [11].

2.8 Previous works on meteor flux density determination

In 1990, Koschack and Rendtel described a method to calculate the flux density from visual observations [12]. Later, this method was further developed by Bellot-Rubio to apply it to photographic meteor observations [13].

Grün et al. developed a flux density model for small objects in the range of 10-21 kg to 10-3 kg [14]. For larger objects, for meteor diameters from 1 m to 9 m, Brown et al. provided a model for flux density as a function of energy [15]. Drolshagen et al. derived a combined meteor flux model from Green's model for the small size range and Brown's model for larger objects. For this purpose, they converted Brown's flux model into a function of mass. The missing middle size range, from 10-3 kg to 103 kg, was then interpolated. The resulting diagram is shown in Figure 2. [16]

With data from the CILBO (Canary Islands Long-Baseline Observatory) and alternative models, they derived a final model of flux density over the range of 10-21kg to 1012 kg. Figure 3 [16] shows the combined flux density models.

Other works on automated video-based observation to determine the flux density in the visual domain were carried out by Molau et al. and Blaauw et al. [17] [18]. Molau et al. evaluated data from the 2011 Draconid Meteor Viewer. The data came from the

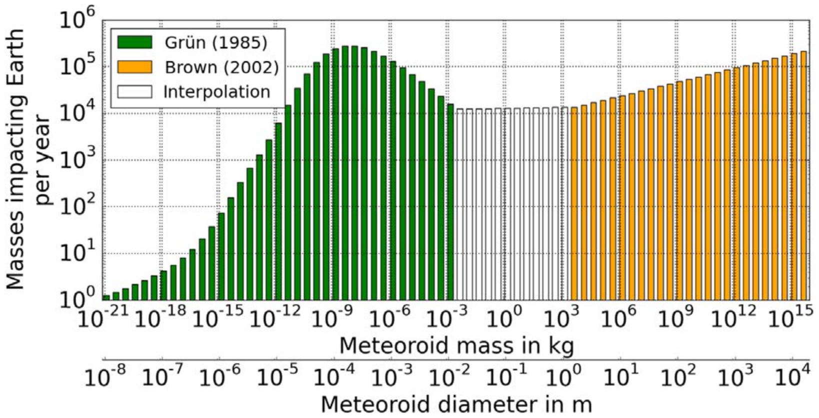


Fig. 2–4: Masses impacting Earth [16]

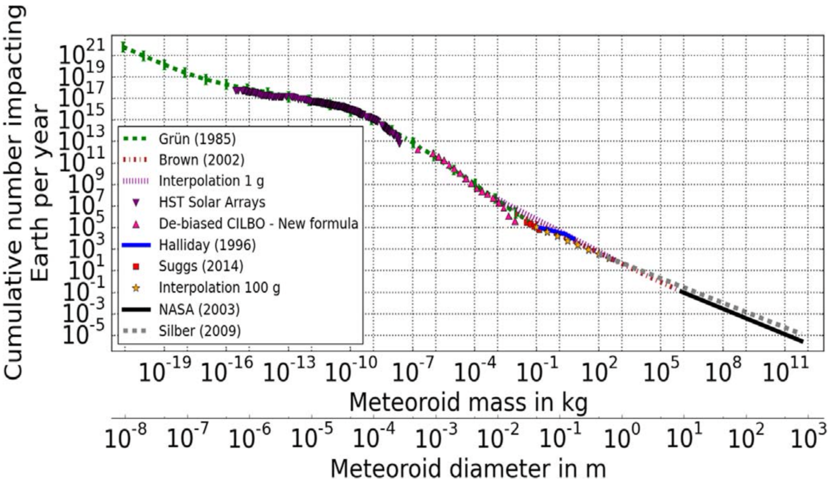


Fig. 2–5: Combined flux density model [16]

Video Meteor Network of the International Meteor Organization. The coverage area was set to a constant altitude of 100 kilometers. Therefore, the number of meteors was divided by the area at that altitude. Blaauw et al. determined flux density using a system of eight wide-angle meteor cameras. Their approach was to create a three-dimensional grid at the altitude of each meteor's brightest point, allowing a more accurate determination of the collection area. Another work by Koschny et al. used the CILBO mentioned above, with two cameras pointed at a position 100 km above the ground. By calculating the longitude and latitude, the overlap area could be determined [19].

3 Approach to Determine The Spatial Coverage of station AMS 80

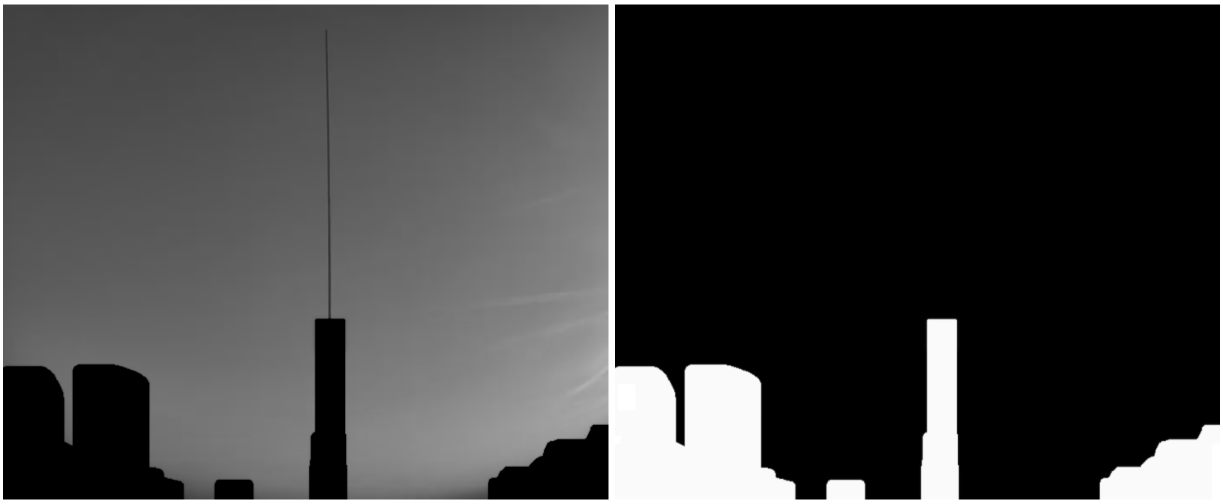
In this chapter, an approach is presented to how the spatial coverage of the AllSky7 Network can be determined. For that purpose, a program called "Horizon" was written with Python to determine the actual horizon data of a single camera. And another program called "Coverage" was written to determine the coverage of the entire network. In the first step, the pixel values of an image must be looked at to obtain the actual horizon. Then, the pixel values can be transformed into azimuth and elevation coordinates. With this information, a polar plot of the actual horizon of a camera system is generated. To determine the spatial coverage of the entire network, the coordinates are transformed into longitude and latitude. The coverage can be represented with a grid in a heat map.

3.1 Camera Systems

The AllSky7 Fireball Network consists of over 50 active camera systems stationed mainly in Europe. Each station has seven highly sensitive NetSurveillance NVT cameras with the SONY STARVIS IMX291 CMOS Sensor. The lens has a focal length of 4 mm and an aperture of f/1.0. The small focal length allows a wide field of view of about 45° × 85°. Due to the low aperture number, the camera can detect objects with a low brightness up to 4 mag. The resolution of a camera is 25°. The first five cameras are evenly spaced in a circle and oriented at about 25° above the horizon. The last two cameras each point north and south, respectively, at an elevation of about 70°. [5] With this setup, a camera system can, in theory, cover the whole sky over a station. In reality, there are often obstacles in front of the cameras, reducing the coverage area.

3.2 Determination of the Real Horizon

To compute the coverage area of a camera system, the real horizon must be determined. The AllSky7 software can generate mask images of the camera footage to identify obstacles in the field of view. An algorithm analyzes photographs taken early in the morning when the sun is10° to 5° below the horizon. In these photographs, the sky appears brighter than the objects in the field of view because the sunlight has not reached them yet. The dark areas below a certain threshold are then covered with a mask geometry. An example of a mask image of camera 4 of station AMS 80 is shown in figure 3–1. Figure 3–1a shows the mask applied to the photograph, while figure 3–1b shows the resulting mask image used to determine the real horizon. However, very thin objects are not detected. For example, the antenna in the photograph in 3–1a is not transferred to the mask image in 3–1b.



(a) Mask in photograph **(b) Resulting mask image**

Fig. 3–1: Mask of camera 4 of the station AMS80

3.2.1 Determination of Horizon Pixels

In the mask images, the sky is displayed in black, and the obstacles are in white. With the help of an image processing tool, the pixel values can be read out. The tool used for this work is the OpenCV package in Python. Reading an image results in a matrix of pixel values that indicate the pixel's color. Depending on the mode in which an image is read, the pixel will have either three channels in color mode or one in grayscale mode. Each channel can take values from 0 to 255. The value of the channel indicates the intensity of the color. For a grayscale image, the value (0) represents the color black, and (255) represents the color white. The values in between are different shades of gray that become lighter with higher channel values. In OpenCV, the channels for the color mode are blue, green, and red. Figure 3–2 shows some examples of pixel color. The blue color is represented by the full intensity of the blue channel and the zero intensity of the other two channels. With zero intensity in all channels, the resulting color is black, while full intensity in all channels gives the color white. A simplified matrix of a mask is shown in Figure 3–3. The position of the pixels is given by their (x, y)-coordinates. The origin is in the upper left corner, with the x-values running from left to right (column number) and the y-values from top to bottom (row number).

In this work, the lowest pixel with a clear view in a column is called horizon pixel. The Horizon algorithm iterates through each column from the top row to the bottom to obtain the horizon pixel. Since the sky is black, the pixel values in all channels are close to zero. The mask begins when the pixel color changes to white e. g. a channel value is bigger than zero. In this case, a threshold of 30 is set to determine the change to white. When a channel in a column exceeds the threshold, the pixel position of the previous row is saved. This pixel is the last pixel with a clear view of the sky. The red pixel in the 3–3 figure represents the horizon pixel of the first column. The threshold is exceeded at pixel (0, 8), so the saved pixel is the red pixel (0, 7). If a column reaches the bottom without detecting a mask, the last pixel of the column is saved. The result

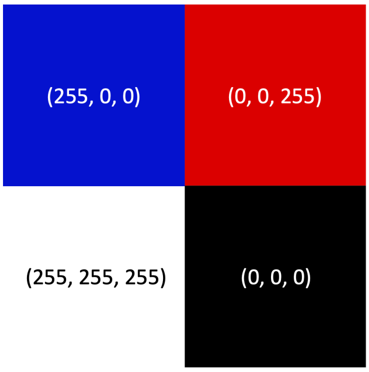


Fig. 3–2: Pixel values in colored mode have three channels (B, G, R) with the values giving the intensity of a color.

of the algorithm is a list with all horizon pixels of the mask image. A visualization of the horizon pixel data for camera 4 of the AMS 80 station is shown in figure 3–4.

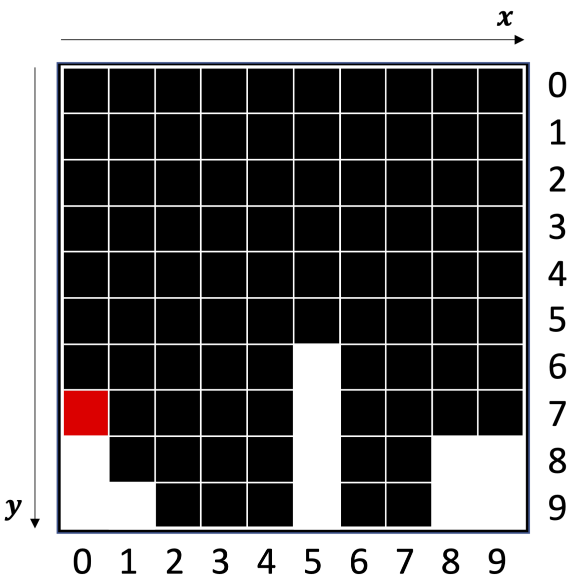


Fig. 3–3: Simplified Pixel Matrix with Horizon Pixel in Red

3.2.2 Transformation of Pixel Positions into Azimuth and Elevation

In order to plot the horizon data, the pixel position must be transformed into azimuth and elevation coordinates. The AllSky7 software uses plate solving to determine the right ascension and declination of the center and to generate a lens model describing the lens distortion. The deviation of the positions of the stars is transformed into a polynomial model. With this model, the coordinates of each pixel can be determined. The lens model for camera four of the AMS 80 station is shown in figure 3–5.

With the lens model, the AllSky7 software can transfer any pixel position into azimuth and elevation with an accuracy of about 0.1 °. In figure 3–6, the contour of the horizon for camera four is shown in the azimuth and elevation grid created by the lens model.

Fig. 3–4: Colored horizon of camera 4 mask of AMS 80

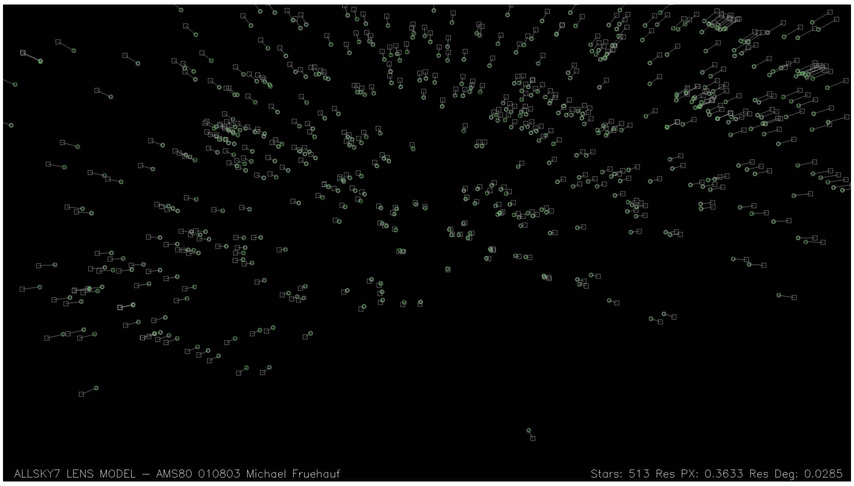
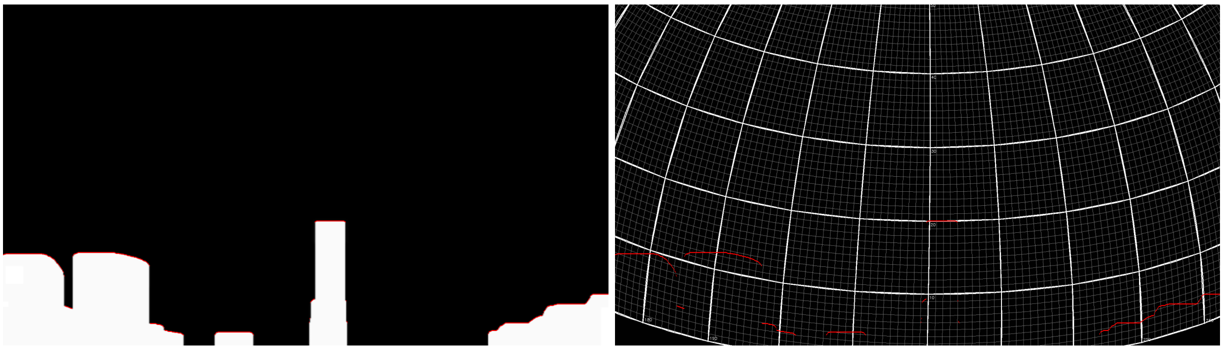


Fig. 3–5: Lens Model for Camera 4 of AMS 80



(a) Colored Horizon for Camera 4 in Mask Image of AMS 80

(b) Horizon Contour for Camera 4 in Grid Image of AMS 80

Fig. 3–6: Horizon for Camera 4 in Mask and Grid Image AMS 80

The Horizon code saves the pixel azimuth and elevation values. To reduce the data and to be able to compare the elevation, the azimuth values are rounded to 0.1 °, and the average elevation of the same azimuth values is taken. The result is a list of azimuth values in 0.1 ° steps and their elevation angles for a camera image.

3.2.3 Data Merging Algorithms for Lower Images

After running the code on all seven cameras of a camera system, the data had to be merged to create the horizon data of a full circle. Because the cameras overlap at the edges, the horizon data of two sequential cameras have the same azimuth values at the overlapping areas. However, the elevation angles are not exactly the same since the viewing angles of the cameras are different. In addition, the masks are created differently for every picture leading to different elevation angles. Therefore, a good merging algorithm had to be found. For the merging algorithm, three options are available: optimistic, pessimistic, and weighted.

3.2.3.1 Weighted Merge

The optimistic and pessimistic approaches create a hard cut-off at the end of an image. A weighting system was applied to smooth the transition. The principle of the algorithm is shown in Figure 3–7. A value is weighted more the deeper it is in an image. First, the overlapping values are counted. Then two weighting factors are introduced: i for the first image and j for the second image. For the example of 20 overlapping values, the first value of the first image is weighted 20 times to 1 for the value of the second image. Advancing into the second image, the weighting factor i of the first image decreases, and the weighting factor j of the second image increases. The sum is divided by the number of values to get the weighted value. In the case of 20 overlapping values, 21 values are received so the sum is divided by 21.

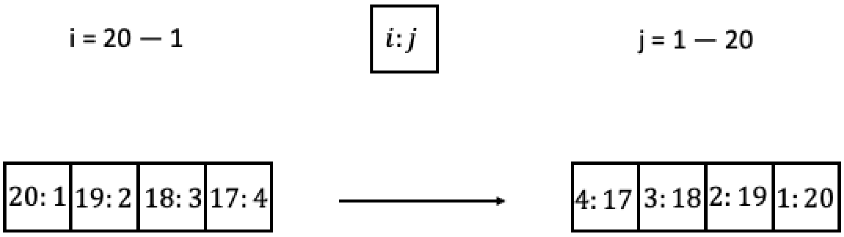


Fig. 3–7: Weighting Factors

3.2.3.2 Pessimistic Merge

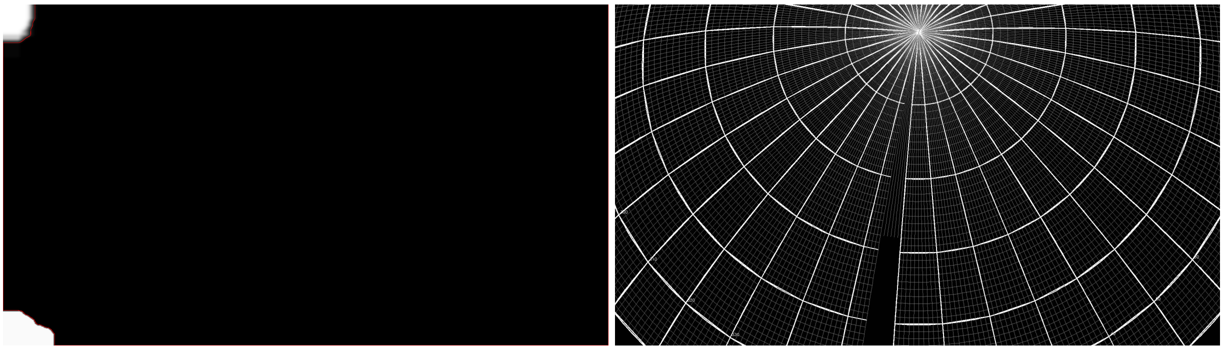
The pessimistic Merge assumes that the mask covers less area than in reality by obstacles. Hence, the algorithm takes the highest elevation angle as the correct value.

3.2.3.3 Optimistic Merge

For the optimistic approach, it is assumed that a mask covers more area than in reality. Therefore, when comparing the elevation angle of the same azimuth values, the lowest elevation is taken as the correct value. The data represented in this work is merged with the optimistic approach because it is assumed that the mask generation covers more area than needed.

3.2.4 Data Merging Algorithms for Upper Images

Although the lower five cameras cover a full circle, some cameras have obstacles in the field of view reaching the top of the image. For example, an upper image is shown in figure 3–8. Considering the lower images would result in a lower elevation angle than in reality due to a cutoff at the upper edge. Therefore, obstacles in the upper images have to be taken into account. However, the horizon pixel search algorithm must be adjusted because the azimuth angles can no longer be assumed constant in a column. Furthermore, the iteration of a column should not stop when reaching a bright pixel. To solve this issue, the algorithm saves the pixel when a color change is detected. To get the obstacles' vertical edges, the algorithm iterates from top to bottom in the first run and from left to right in the second run. The horizon pixels at the edges of the obstacles in the upper images override the horizon data for the same azimuth of the lower images.



(a) Colored Horizon for Camera 6 in Mask Image of AMS 60 **(b) Horizon Contour for Camera 4 in Grid Image of AMS 80**

Fig. 3–8: Horizon for Camera 6 in Mask and Grid Image AMS 60

3.2.5 Horizon Data of Camera AMS 80

The completed horizon data of a camera system consists of a list with azimuth angles from 0° to 359.9° in 0.1° steps and their elevation angles. An example of the resulting horizon for station AMS 80 is shown in figure 3–9. The elevation angle is plotted over the azimuth angle. The visible sky lies above the plotted line. All horizontal lines in the image are of inverse parabolic shape in the diagram because of the lens distortion. The large arcs represent a single camera, whereas the small arcs display jumps due to obstacles in the field of view. The first arc is displaying the horizon of camera 1 and goes from 320° to 42°, the second arc represents camera 2 and goes from 32° to 120°, camera 3 goes from 103° to 189°, camera 4 goes from 174° to 262° and camera 5 goes from 242° to 329°. Camera 1, 2 and 5 are only having few obstacles and therefore the arcs can be recognized very well. In this case, there are no obstacles in the upper images, so the elevation angles stay under 30°.

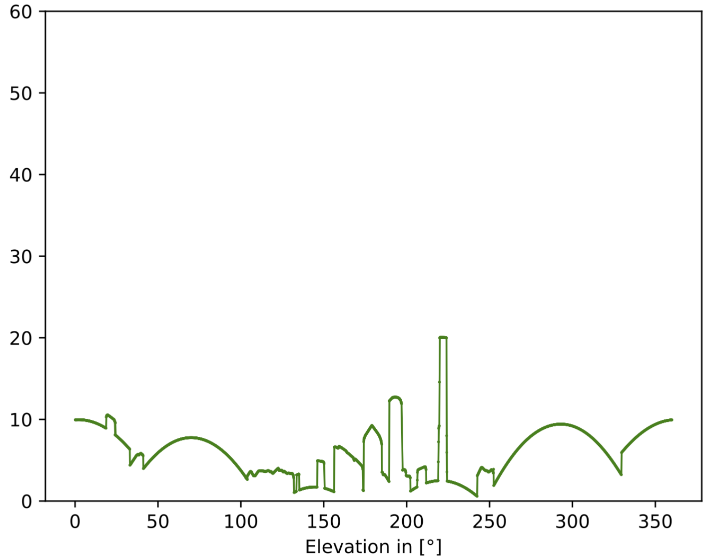


Fig. 3–9: Horizon Line of AMS 80

3.3 Determine the Observing Coverage of Station AMS 80

To determine the area that the AllSky7 cameras can observe, the actual horizon on the images had to be considered. Under perfect conditions, at zero height above the ground, the elevation of the horizon is 0°. However, in most cases, the perfect horizon is obscured by houses or trees in the line of sight of the camera system. Therefore, in the first step, an approach was found to determine the actual horizon of a camera system. The next step was calculating the observing distance in all directions and plotting the covered area. After determining the distance, the station's coverage area could be calculated.

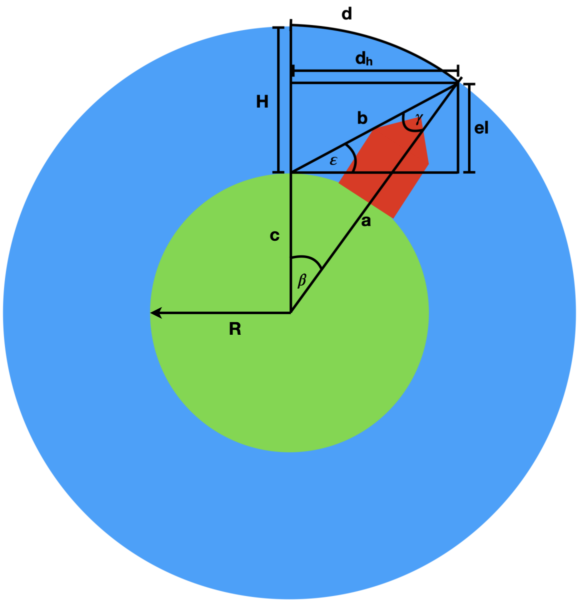


Fig. 3–10: Principal to Determine the Observing Distance

3.3.1 Calculation of the Observing Distance

Figure 3–10 shows the concept of an obstructed view, where φ is the elevation, and d_h is the horizontal distance we can see at a given height.

For the calculations, we consider the triangle with the sides a, b, c. First, we define the angle α, which is the elevation angle ϵ plus 90°. The length c is the average radius of the Earth R. Moreover, the length a is given by R plus the height H above sea level at which we expect the meteors. Therefore, we get the following expressions:

$$\alpha = 90^\circ + \epsilon, \tag{3–1}$$

$$c = R, \tag{3–2}$$

$$a = R + H. \tag{3–3}$$

To then obtain the observed distance, we use the following trigonometric considerations.

With the sine law

$$\frac{a}{sin(\alpha)} = \frac{c}{sin(\gamma)} \tag{3–4}$$

we get:

$$\begin{aligned} \gamma &= \arcsin\left(\frac{c \cdot \sin(\alpha)}{a}\right), \tag{3–5}\\ \beta &= 180^\circ - \alpha - \gamma, \tag{3–6}\\ \beta &= 90^\circ - \epsilon - \arcsin\left(\frac{R \cdot \sin(\alpha)}{R + H}\right). \tag{3–7} \end{aligned}$$

Finally, we get the curved distance d and the horizontal distance d_h, which we can see at the height of H:

$$\begin{aligned} d &= 2\pi r \cdot \frac{\beta}{360^\circ}, \tag{3–8}\\ d_h &= \sin(\beta) \cdot c. \tag{3–9} \end{aligned}$$

.

The observing distance of camera AMS 80 is plotted in figure 3–11 in polar coordinates, where 0° degrees represents the North.

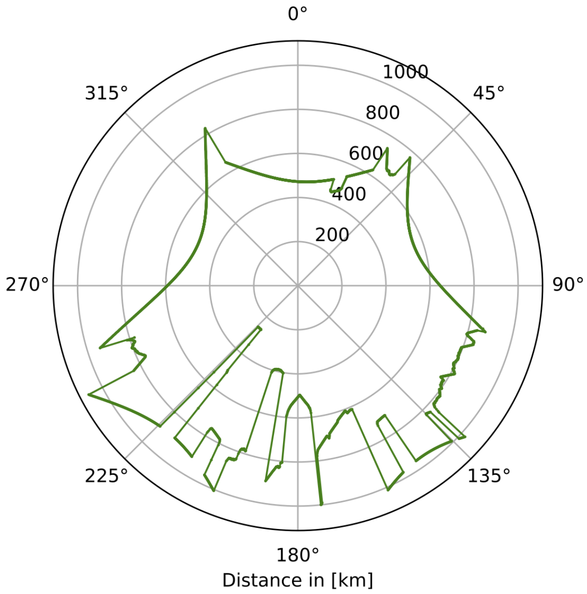


Fig. 3–11: Observing Distance for Azimuth angle of AMS 80

3.3.2 Determination of the Coverage Area of a Camera

As the last step, the coverage of a camera system had to be ascertained. In the following, three procedures of area determination are presented.

3.3.2.1 Calculation of the Coverage Area with Trapezoidal Rule

The first method to calculate the area is the trapezoidal rule. Figure 3–12 shows the principle of the trapezoidal approach. The area between the x-axis and the function is divided into n trapezoidal segments. Subsequently, the areas S_i of the trapezoids are summed up to approximate the integral of the function. The Python package "scipy" has a built-in function that uses the rule to calculate the area of a function. Since the distances are obtained in polar coordinates, the data points had to be transferred from polar into Cartesian coordinates.

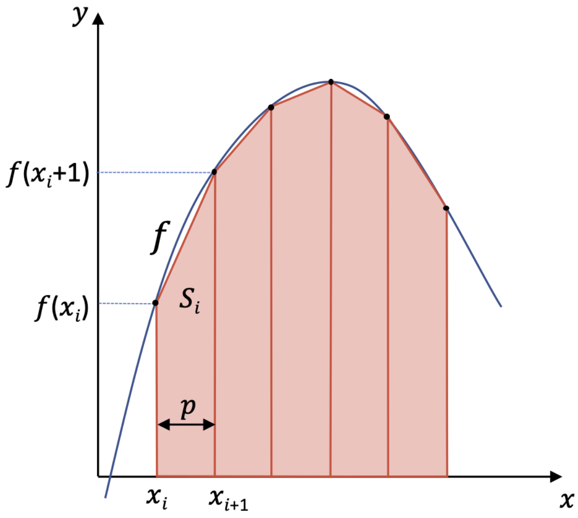


Fig. 3–12: Principal of the Trapezoidal Rule

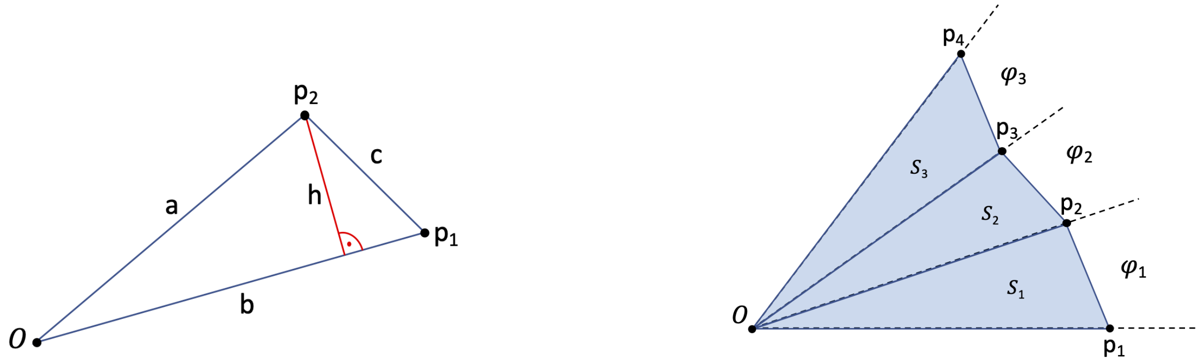
3.3.2.2 Calculation of the Coverage Area with a Triangle Approach

The second approach represents the area between two data points as triangles. The area of a triangle can then be calculated with the equation:

$$A = \frac{1}{2} \cdot b \cdot h \tag{3–10}$$

.

Then the areas S_i of the triangles can be summed to obtain the station's coverage area. The Concept is shown in figure 3–13. The advantage over the first approach is that the data points do not have to be converted into Cartesian coordinates.



(a) Area of a Triangle (b) Triangle Areas of Data Points

Fig. 3–13: Triangle Approach for Area Calculation

3.3.2.3 Calculation of the Coverage Area with a Spherical Integral

The last two methods are intuitive, but they can only calculate the area in a flat plane, whereas, in reality, the area is curved. Therefore a third approach was developed using Riemann sums with a midpoint rule seen in 3–14.

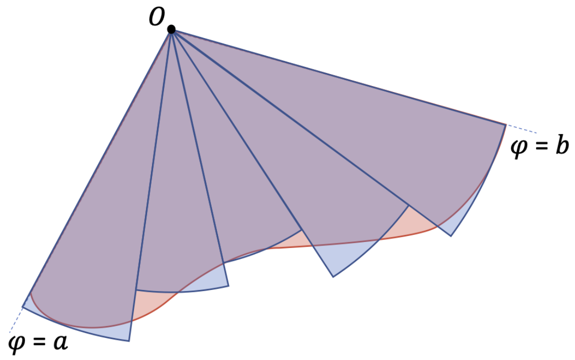


Fig. 3–14: Riemann Integral with Midpoint Rule

In order to deduce the curved area, a modified version of the general integral of a sphere was used. The spherical integral can be expressed as:

$$\mathrm{d}V = \int_0^{\Phi}\int_0^{R}\int_0^{\Theta} r^2 \sin(\theta)\mathrm{d}\theta\mathrm{d}r\mathrm{d}\varphi, \tag{3–11}$$

with the radius r, the angle θ in the y-z-plane, starting from the z-axis, and angle φ in the x-y-plane, starting from the x-axis. With a fixed radius R and the angle φ between two data points, the area $\mathrm{d}A$ can be computed with:

$$\mathrm{d}A = \varphi R^2 \int_0^{\Theta} \sin(\theta)\mathrm{d}\theta = \varphi R^2[1 - cos(\theta)] \tag{3–12}$$

In this case, the radius R is composed of the Earth's radius $R_{Earth} = 6371km$ plus the height of the meteor level $H = 100\,\text{km}$: $R = R_{Earth} + H = 6471\,\text{km}$. The azimuth angle between two data points equals the incremental angle φ: $\varphi = 0.1^\circ$ and the upper boundary theta equals 90° minus the elevation angle ϵ: $\Theta = 90^\circ - \epsilon$. The area $\mathrm{d}A$ for two data points is shown in figure 3–15, and the observable area can be deduced by summing the areas of all data points.

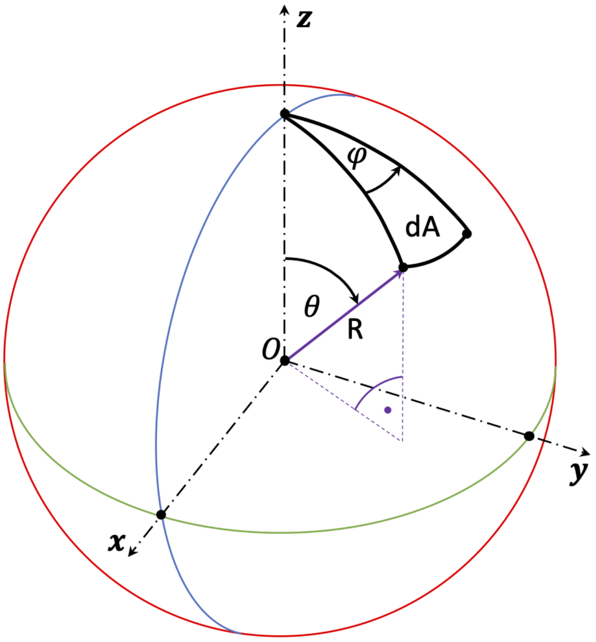


Fig. 3–15: Surface Integral dA of a Sphere with radius R

3.3.3 Comparison of Proposed Methods

Now the previously introduced methods for area determination are compared and evaluated. For that purpose, the areas for station AMS 80 at an altitude of 100 km will be used. In table 3–1, the calculated areas of the trapezoidal rule and the triangle approach are compared to the spherical integral approach. The difference between all three methods is minimal due to the large radius. Therefore the curved surface could be neglected. However, the spherical integral was chosen for the upcoming calculations since extending the model to calculate volumes is effortless.

Tab. 3–1: Calculated Area Comparison for AMS 80

	Spherical Integral	**Triangle Approach**	**Trapezoidal Rule**
Area	1 480 109 km^2	1 479 155 km^2	1 481 378 km^2
Difference	0 %	–0.6 %	0.09 %

3.3.4 Calculation of the Observable Volume

Because meteors are moving through the atmosphere, the meteor level of 100 km can only be used as a projection plane for meteor trails. Therefore it can be advantageous to calculate a coverage volume from 80 km to 120 km height since most meteors have their maximum brightness at this height level. For the volume calculation, the general spherical integral in equation 3–11 can be modified to:

$$\mathrm{d}V = \varphi \int_{R_1}^{R_2} \int_{0}^{\Theta} r^2 \sin(\theta) \mathrm{d}\theta \mathrm{d}r, \tag{3–13}$$

$$\mathrm{d}V = \varphi \int_{0}^{\Theta} \left[\frac{1}{3} r^3 \sin(\theta) \right]_{R_1}^{R_2} \mathrm{d}\theta, \tag{3–14}$$

$$\mathrm{d}V = \frac{1}{3} \varphi [R_1^3 (\cos(\theta) - 1) + R_2^3 (1 - \cos\theta)], \tag{3–15}$$

for radius boundaries from $R_1 = R_{Earth} + H_0$ and $R_2 = R_{Earth} + H_2$ with $H_1 = 80\,\text{km}$ and $H_2 = 120\,\text{km}$ height. For station AMS 80, the coverage volume can be calculated to 59 204 542 km^3.

4 Determination of the Spatial Coverage of the AllSky7 Fireball Network

In the previous chapter, the procedure for obtaining the horizontal data and calculating the coverage of a single camera was described. In this chapter, the determination of the spatial coverage of the entire network will be performed using the methods introduced in the previous chapter. In order to map the coverage of all cameras, the horizon data have to be transformed into longitude and latitude coordinates. Then a grid is generated, which counts how many cameras can see a specific grid point. A heat map then visualizes the coverage. After that, the magnitude reduction due to the atmosphere and the distance was considered, and a new spatial coverage profile was created.

4.1 Coordinate Transformation to Latitude and Longitude

So far, only individual cameras have been considered. For this purpose, an observer-centered coordinate system with azimuth and elevation was a good solution. However, an earth-centered coordinate system is needed to analyze the coverage by multiple cameras. Therefore, the data points are transferred to the geographic coordinate system with longitude and latitude. To convert the data points, consider a sphere of radius $R = R_{Earth} + H = 6471\,\text{km}$ with the mean radius of the Earth $R_{Earth} = 6371\,\text{km}$ and the meteor observation altitude $H = 100km$ shown in Figure 4–1. The north pole N , the station position P_1, and an arbitrary horizon point P_2 form a triangle on the spherical surface with the arcs of a great circle a, b, c and the angles A, B, C between the arcs.

With known longitude and latitude of the station (λ_1, ϕ_1), the longitude λ_2 and the latitude ϕ_2 of the second point can be calculated with the spherical sine and cosine laws:

$$B = arcsin\left(sin(b) \cdot \frac{A}{sin(a)}\right), \tag{4–1}$$

$$a = arccos(cos(b)cos(c) + sin(c)sin(b)cos(A)). \tag{4–2}$$

With the calculated angle B, which is the difference angle of λ_2 and $\lambda 1$, and a, which is 90 ° minus the latitude of the second point, the longitude λ_2 and the latitude ϕ_2 of the horizon point can be calculated with:

$$b = distance/R, \tag{4–3}$$

$$c = 90^\circ - \phi_1, \tag{4–4}$$

$$A = azimuth. \tag{4–5}$$

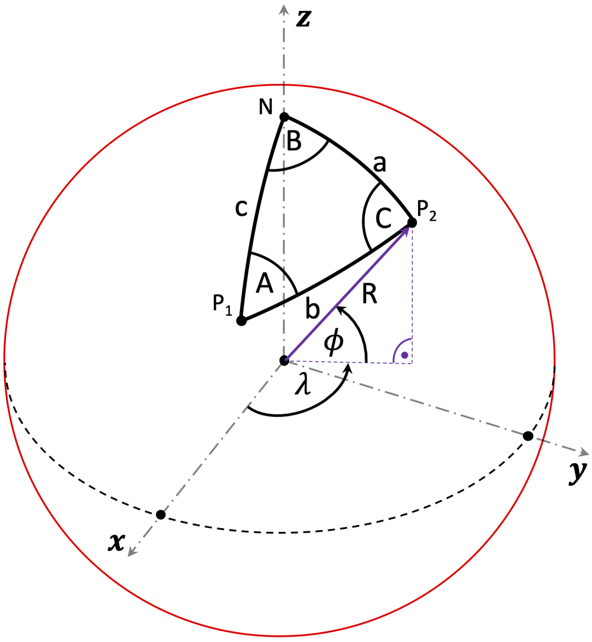


Fig. 4–1: **Spherical Sine and Cosine Law**

With the calculated angle B, which is the difference angle of λ_2 and $\lambda 1$, and a, which is 90 ° minus the latitude of point two, the longitude λ_2 and latitude ϕ_2 of the horizon point can be computed with:

$$\lambda_2 = B + \lambda_1, \tag{4–6}$$

$$\phi_2 = 90^\circ - a. \tag{4–7}$$

After converting the AMS 80 station horizon data to longitude and latitude, the result is the plot of the coverage area shown in Figure 4–2 at an observation altitude of 100 km.

4.2 Determination of the Spatial Coverage

The first step in determining the network coverage is to discretize the area of Europe. In this case, the grid created ranges from –20 ° to 37 ° in longitude and from 27 ° to 67 ° in latitude. The grid points contain the latitude, longitude, and a counter that counts the cameras that can see the grid point at the observation height. For each station, the algorithm loops through the grid file and checks which grid points are visible to the camera. To check if a point is within camera coverage, the distances at the same azimuth angles of the horizon point and the grid points are compared. If the distance from the meteor level station position to the grid point is less than the distance to the horizon point, the grid point is covered by the camera. The distances to the horizon points have already been calculated in section 3.3.1. The azimuth and distance from the station must first be determined for the grid points. For the distance, the spherical cosine law applies for b:

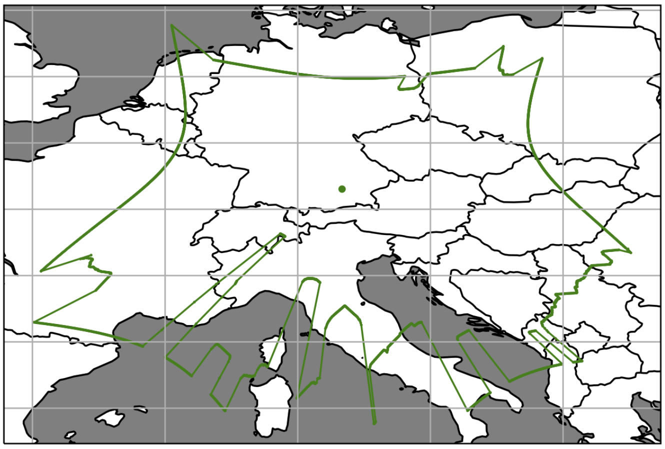


Fig. 4–2: Coverage of station AMS 80

$$b = arccos(cos(a)cos(c) + sin(a)sin(c)cos(B)). \tag{4–8}$$

The distance can then be calculated with the expression 4–4:

$$distance = b \cdot R. \tag{4–9}$$

To obtain the azimuth angle to the grid point, the spherical cosine law for a can be solved for $A = azimuth$:

$$azimuth = \frac{arccos(cos(a) - cos(b)cos(c))}{sin(c)sin(b)}. \tag{4–10}$$

4.2.1 Determination of the Coverage Area

In order to calculate the coverage area, an area must be defined to represent the one-dimensional grid points. Here, the area spanned by going half the distance to the next grid points was used. The principle is shown in figure 4–4.

Now, a modified integral of the general spherical volume can be used to calculate the area dA of a grid point, as shown in Figure 4–4. With a constant radius $r = R + H$, the angle phi, which is the angle between two grid points, and the initial and final angles of theta, the modified integral can be derived from equation 4–12 as follows:

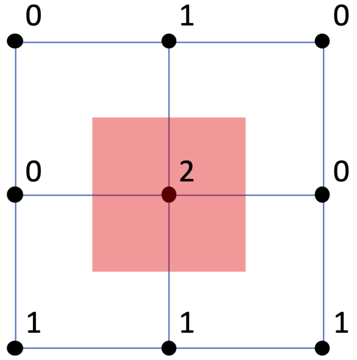


Fig. 4–3: **Area dA of a Grid Point**

$$\mathrm{d}V = \int_0^R \int_0^\varphi \int_0^\theta r \cdot sin(\theta)\ \mathrm{d}r\mathrm{d}\varphi\mathrm{d}\theta, \qquad (4\text{–}11)$$

$$\mathrm{d}A = r^2\varphi \int_0^\theta sin(\theta)\mathrm{d}\theta = r^2\varphi[cos(\theta_1) - cos(\theta_2)]. \qquad (4\text{–}12)$$

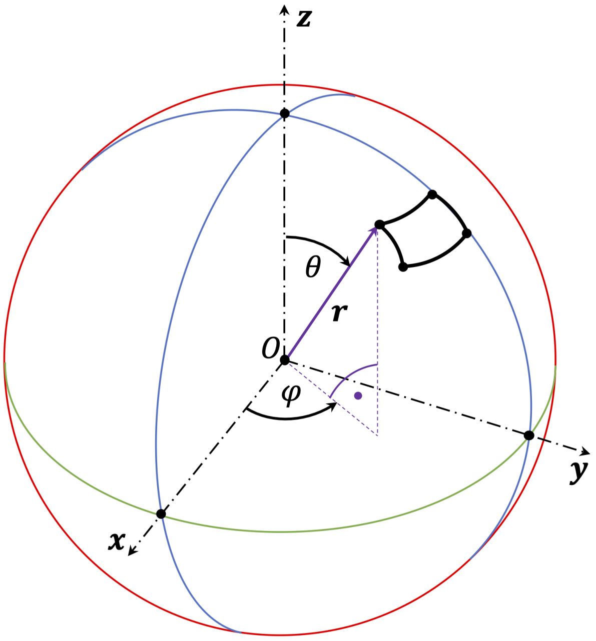


Fig. 4–4: **Area dA of a Grid Point**

4.2.2 Determination of the Coverage Volume

To calculate the volume, the constant radius r for the area calculation is now extended to a variable radius. The expanded volume is shown in figure 4–5 , where the radius ranges from r_1 to r_2. The modified integral for the volume dV of a grid point is given in equation 4–13.

$$\mathrm{d}V = 1/3\varphi[r_1^3(cos(\theta_2) - cos(\theta_1)) + r_2^3(cos(\theta_1) - cos(\theta_2))]. \tag{4–13}$$

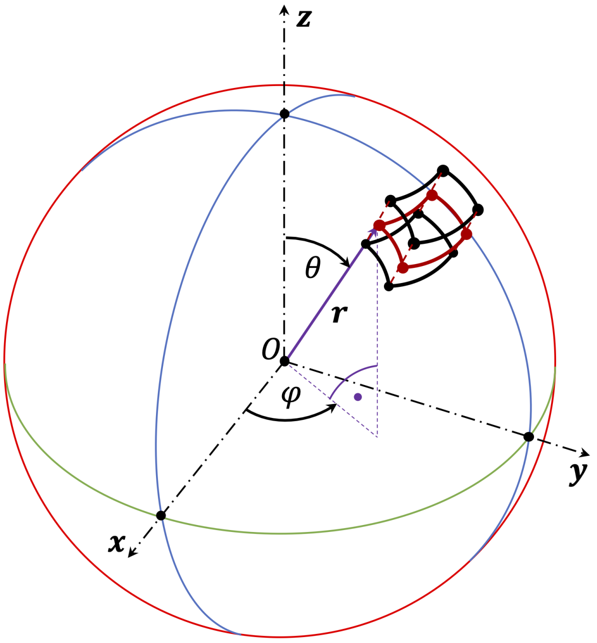


Fig. 4–5: Volume dV of a Grid Point

4.2.3 Choosing the Right Step Size

A fine grid provides the most accurate values, but the required time to calculate the values quadruples if the step size is doubled in the 2D case. Therefore, a good compromise for the step size of the grid had to be found. The step size was defined by $step = 1/n$, starting with $n = 1$, then n was doubled until $n = 8$. The difference in the area to the previous step size was then compared. The resulting coverage areas, their differences, and the percentages are shown in the table 4–1.

The first reduction of the step size from 1 ° to 0.5° resulted in a difference of 0.77 %. Further halving the step size to 0.25° resulted in a difference of 0.13 %. With a step size of 1.25°, the difference was only 0.09 %. For the upcoming calculations, the step size was set to $step = 1/4$ since this was the best ratio of accuracy to computation time. The resulting coverage of the network at a meteor level of 100 km is visualized in a heat map in Figure 4–6. The heat map was obtained by plotting the grid with matplotlib as pcolormesh.

Tab. 4–1: Influence of the Step Size on the Coverage Area

n	Coverage Area	Difference	Percentage
1	4 947 703 km^2		
2	4 909 707 km^2	37 996 km^2	0.77 %
4	4 903 549 km^2	6157 km^2	0.13 %
8	4 898 987 km^2	4562 km^2	0.09 %

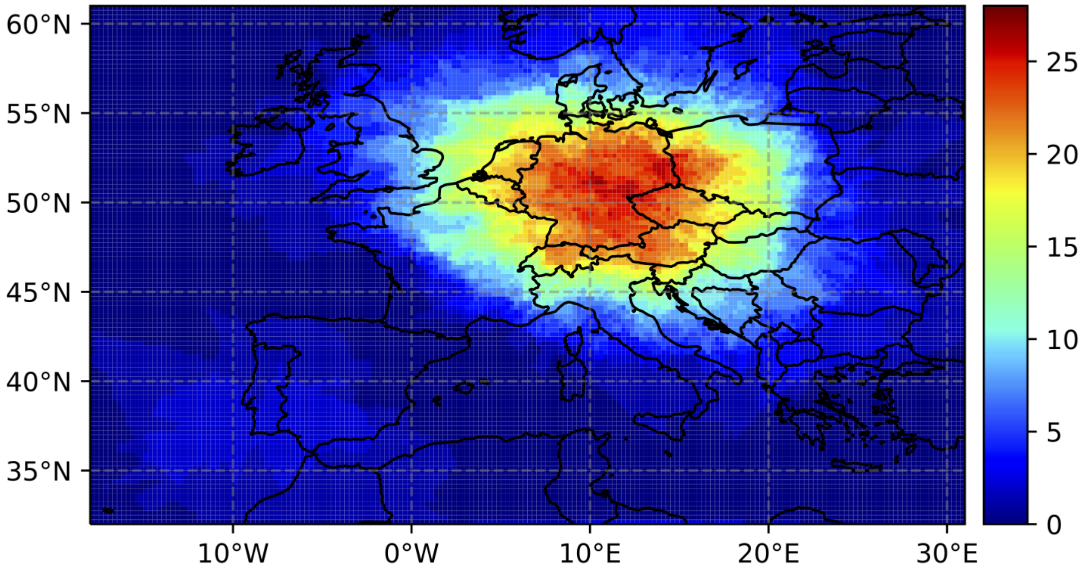


Fig. 4–6: Heat Map of the Coverage of AllSky7 Network

4.3 Consideration of the Brightness Reduction at Low Elevation Angles

Until now, only the visible sky was considered for the detection profile to detect a meteor. However, in reality, the brightness is reduced by the atmosphere and the distance between the light source and the observer. In this section, a model is proposed to predict the brightness reduction and limit the camera's elevation angle.

4.3.1 Magnitude Reduction due to the Atmosphere

The extinction of light by the atmosphere is influenced mainly by three factors: Rayleigh scattering, aerosol scattering, and molecular absorption. Each factor depends on wavelength as well as time and altitude variations. [20]

4.3.1.1 Rayleigh Scattering

Rayleigh scattering is caused by air molecules whose size is smaller than the wavelength of the scattered light. The extinction caused by this can be modeled under standard conditions by the following expression [20]:

$$A_{RaySTP} = 0.0094977 \cdot \lambda^{-4} \cdot n_s^2 \cdot e^{-\frac{h}{7.996}}, \tag{4–14}$$

where λ is the wavelength in µm, h is the height of the observer above sea level, and n_s is the refractive index, which can be described as follows:

$$n_s = 0.23465 + \frac{107.6}{146 - \lambda^{-2}} + \frac{0.93161}{41 - \lambda^{-2}}. \tag{4–15}$$

A scaling factor can be used to account for pressure and temperature deviations from the standard state [21]:

$$A_{Ray} = A_{RaySTP} \cdot \left(\frac{T_{STP}}{T} \frac{P}{P_{STP}} \right), \tag{4–16}$$

where $T_{STP} = 273.15K$, $P_{STP} = 760mmHg$, and T and P are the actual temperatures and pressures during the observations.

4.3.1.2 Aerosol Scattering

Scattering by aerosols such as water, pollen, dust, or soot strongly depends on the concentration of these particles and varies not only from place to place but also in time. The effect of aerosols can be divided into wavelength-independent neutral aerosol extinction A_n and selective aerosol extinction A_S [22]. The smaller the particles, the stronger the wavelength dependence. The extinction can be modeled as [20]:

$$A_{aer} = A_0 \cdot \lambda^{-\alpha_0} \cdot e^{\frac{-h}{H}}. \tag{4–17}$$

The values A_0, α_0, and H are difficult to determine without direct measurements. However, since the extinction can be calculated accurately due to Rayleigh scattering and the minimal influence of ozone, the values can be adjusted to correspond to typical extinction values. Green suggests as values $A_0 = 0.05$, $\alpha_0 = 1.3$, and $H = 1.5\,\text{km}$ [23].

4.3.1.3 Molecular Absorption

The effect of molecular absorption on light extinction is quite small compared to the other two effects. Knowing the total thickness T in $\text{mm}\,\text{cm}^{-1}$ of the ozone layer, the extinction can be calculated according to [22]:

$$A_{oz} = 1.09 \cdot T \cdot k(\lambda), \tag{4–18}$$

where $k(\lambda)$ is the absorption coefficient of ozone. Without knowing the ozone concentration at the observer's location, Green suggests a value for ozone absorption of $A_{oz} = 0.016$ [23].

4.3.1.4 Air Mass

The factors presented for light extinction by the atmosphere apply to observations with a zenith angle $z = 0$. The zenith angle refers to the angular distance from the zenith. To estimate extinction at smaller angles, the air mass is used to describe the amount of air between the observer and the object. A rough approximation of the air mass can be given by $1/cos(z)$. At a zenith angle of zero, the air mass is equal to 1; as angle z increases, the air mass increases slowly at first but rapidly at high zenith angles. However, this approximation fails for observations near the horizon. A more accurate estimate is given by Rozenberg, who is mentioned by Green [23]:

$$X = cos(z) + 0.025 \cdot e^{-11cos(z)}. \tag{4–19}$$

The final reduction in brightness Δm_{atm} due to light extinction by the atmosphere can then be calculated with the expressions:

$$A = A_{Ray} + A_{aer} + A_{oz}, \tag{4–20}$$

$$\Delta m_{atm} = X \cdot A. \tag{4–21}$$

This model assumes a linear dependence between light extinction and air mass and is known as the Bouguer method. However, light shifts to a redder spectrum as air mass increases, which changes the wavelength-dependent extinction coefficients. Red light is scattered less than blue light, resulting in nonlinear behavior [22]. Nevertheless, a linear dependence was assumed for this work. Also, the coefficients calculated here are values for the reduction through the entire atmosphere. Since the meteors are visible at altitudes of about 120 km to 80 km, the values may not be accurate. However, the air density above this level is very low, so most of the light extinction occurs below this meteor level. Therefore, the values should still be a reasonable estimate.

4.3.2 Magnitude Reduction due to the Distance

In addition to the decrease in brightness due to the atmosphere, the light loses intensity with increasing distance. The intensity is inversely proportional to the square of the distance to the observer. The ratio of a light source at a distance d_1 to a light source at a distance d_2 can be expressed as follows:

$$\frac{I_1}{I_2} = \left(\frac{d_2}{d_1}\right)^2. \tag{4–22}$$

Therefore, the reduction of the magnitude due to the distance can be calculated with:

$$m_2 - m_1 = 2.5log\left(\frac{I_1}{I_2}\right), \tag{4–23}$$

$$m_2 - m_1 = 2.5lo\left(\frac{d_2}{d_1}\right)^2, \tag{4–24}$$

$$\Delta m_{dist} = 5log\left(\frac{d_2}{d_1}\right). \tag{4–25}$$

4.4 Determination of the Limiting Elevation Angle

To adjust the spatial coverage to the brightness reduction, both effects were considered. With the two models, the combined brightness reduction is given by:

$$\Delta m = \Delta m_{dist} + \Delta m_{atm}, \tag{4–26}$$

$$= 5log\left(\frac{d_2}{d_1}\right) + X \cdot A. \tag{4–27}$$

Since there is no analytical solution for the combined model for the elevation angle ϵ, the problem was solved numerically. Standard conditions were assumed, and the wavelength was set to 0.51 µm. The values for A_0, α_0, and H were set to 0.05, 1.3, and 1.5 km, respectively. A value of $A_{oz} = 0.016$ was assumed for the molecular absorption. To calculate the distance-induced brightness decrease, the reference distance d_1 at the meteor level was set as $d_1 = 100$ km. The distance d_2 is the direct distance at which the camera can no longer detect the object and depends on the elevation angle ϵ. It holds:

$$\beta = arccos\left(\frac{R \cdot cos(\epsilon)}{R + H}\right) - \epsilon, \tag{4–28}$$

$$d_2 = (R + H) \cdot \frac{sin(\beta)}{cos(\epsilon)}. \tag{4–29}$$

The problem is solved numerically in the code for a specified magnitude m_z at zenith and the limiting magnitude $m_{lim} = 4$ mag of the cameras so that it holds:

$$m_z - \Delta m < m_{lim} = 4\,\text{mag}. \tag{4–30}$$

The limiting angle is increased from 0° in 1° steps until the resulting magnitude $m_r = m_z - \Delta m$ is smaller than 4 mag. The table 4–2 shows the limiting angles for the magnitudes –3, –2, –1 and 0. Each camera's minimum angle is calculated based on its height H above sea level. The elevation angles in the data that are less than the minimum angle are then set to the limiting angle.

Tab. 4–2: Limiting Elevation Angle for Different Magnitudes

Magnitude	Light Extinction	Distance	Magnitude Reduction	Resulting Magnitude	Limiting Angle
-3	2.463	4.246	6.709	3.709	5°
-2	1.856	3.881	5.736	3.736	7°
-1	1.344	3.394	4.738	3.738	10°
0	978	2.851	3.829	3.828	14°

In figure 4–7 the heatmap of the reduced coverage for a magnitude at zenith $m_z = -3$ mag is shown.

Finally, it should be noted that the light is distributed over several pixels when observing moving objects with a camera. Therefore, there is also a decrease in brightness due to the velocity of the meteor. This effect is being researched by another student and has been neglected in this paper.

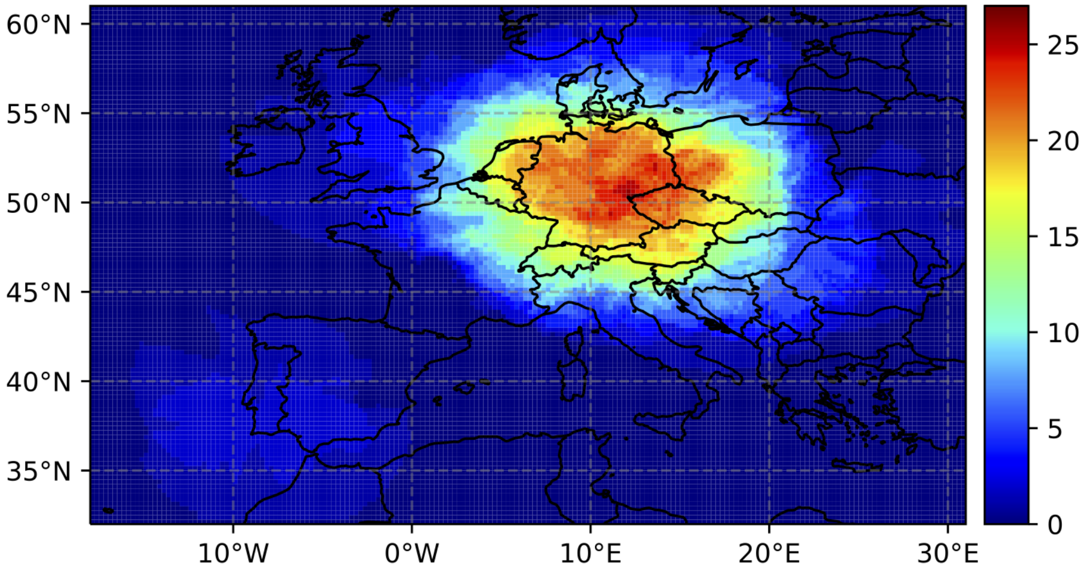


Fig. 4–7: Heat Map of Reduced Coverage of the AllSky7 Network

5 Results and Analysis

In this chapter, the results of the area calculations are discussed. First, the calculated area is compared with the grid's and the world's area. Then the effect of brightness reduction on the area is determined. Finally, the effect of observation heights is examined.

5.1 Spatial Coverage of the AllSky7 Network Compared to the Grid and World

Figure 5–1 shows the coverage of the AllSky7 network for an ideal case without corrections for brightness degradation at an altitude of 100 km. For clarity, the camera counters have been grouped into seven categories. All grid points with a camera counter of less than two were set to zero, resulting in the yellow area. The next areas are grid points, which can be observed up to 5, 10, 15, 20, 25 and 27 cameras, respectively. Since at least two cameras are needed to calculate the trajectories of the meteors, the orange area is the area of interest. In Table 5–1, the orange area and volume are compared to the displayed grid's area and the entire world's area. The volume was calculated from a height of 80 km to 120 km since most meteors are visible at this altitude. The percentage indicates the ratio of coverage to the grid or world. The grid covers 25.23 % of the grid area and 0.93 % of the world area. The same percentages apply to the volume. The sky over Germany is covered by at least 20 cameras almost everywhere, even up to 27 since most cameras are stationed in Germany. Smaller neighboring countries, such as the Netherlands, Belgium, Switzerland, Austria, the Czech Republic, and even Poland, also have good coverage due to their proximity to Germany. However, there are no AllSky7 cameras in France, as France has its own network called FRIPON. The Nordic and Eastern countries, as well as Spain, also have poor coverage.

Tab. 5–1: Ideal Coverage Area of the AllSky7 Network at a Height of 100 km

	Coverage	Grid (Europe)	World
Area	$4.95 \times 10^6\,\mathrm{km}^2$	$19.43 \times 10^6\,\mathrm{km}^2$	$526.2 \times 10^6\,\mathrm{km}^2$
Volume	$1.96 \times 10^8\,\mathrm{km}^3$	$7.77 \times 10^8\,\mathrm{km}^3$	$210.48 \times 10^8\,\mathrm{km}^3$
Percentage	0 %	25.23 %	0.93 %

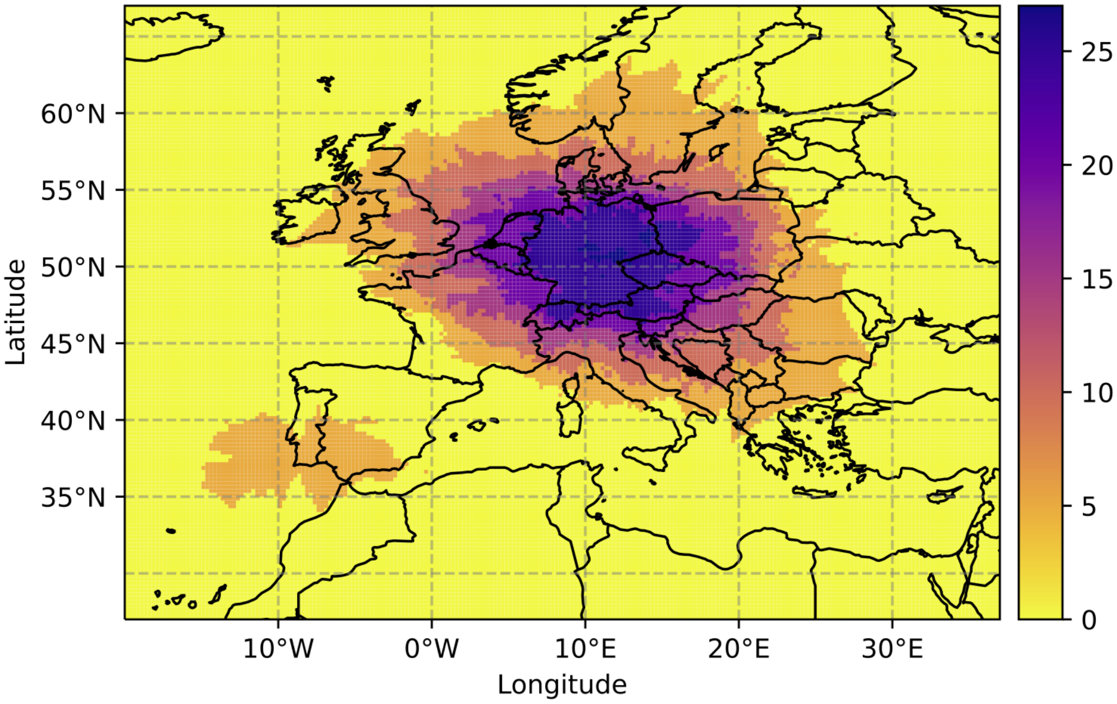


Fig. 5–1: Ideal Coverage of the AllSky7 Network at a Height of 100 km

5.2 Reduced Spatial Coverage due to Brightness Reduction at Low Elevation Angles

As mentioned in section 4.3, the limiting angle for a given brightness at the zenith can be calculated for each camera. The resulting cutoff angle depends mainly on the altitude of the stations. For most stations, the minimum elevation angle was calculated to be 5° or 6°. Increasing the elevation angle means decreasing the visible distance. The distance a camera can see a point in the azimuth direction was then adjusted. The resulting coverage is shown in Figure 5–2. It can be seen that the edges were smoothed by truncating low-elevation angles. Coverage in the center is still strong, although not as dense as in the ideal coverage profile. The data are summarized in Table 5–2. The reduced area covers only about 76.89 % of the ideal coverage area. In terms of the grid, the covered area drops from 25.23 % to 19.4 % and in terms of the world, from 0.93 % to 0.72 %.

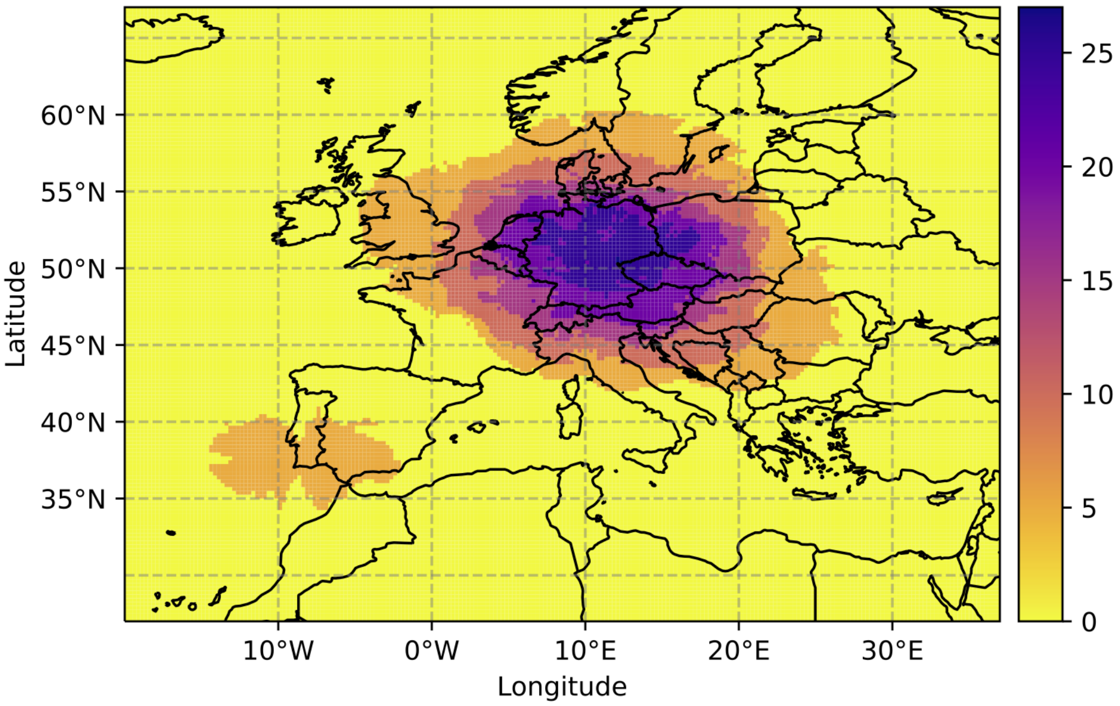


Fig. 5–2: **Adjusted Coverage due to Brightness Reduction at Low Elevation Angles**

Tab. 5–2: **Reduced Coverage due to Brightness Reduction at Low Elevation Angles at a Height of 100 km**

	Reduced Coverage	**Ideal Coverage**	**Grid (Europe)**	**World**
Area	3.77×10^6 km^2	4.95×10^6 km^2	19.43×10^6 km^2	526.2×10^6 km^2
Volume	1.51×10^8 km^3	1.96×10^8 km^3	7.77×10^8 km^3	210.48×10^8 km^3
Percentage	0 %	76.89 %	19.4 %	0.72 %

5.3 Influence of the Observation Height on the Spatial Coverage

Lastly, the Influence of the Observation Height on the Spatial Coverage was examined. For this the, coverage values of 80 km and 120 km height were compared with the reference area at 100 km. All areas in this section are adjusted to the limiting elevation angle.

5.3.1 Comparison of Coverage Area at 80 km to 100 km Height

For observations at lower altitudes, the field of view is narrower, so the area covered should be reduced at the edges. Figure 5–3 shows the comparison of the area at 80 km and 100 km altitude. As expected, coverage shrinks at the edges. Furthermore, the total number of cameras decreases to 20 to 15 per grid point over Germany. The maximum camera density also decreases from 27 to 25. Table 5–3 compares the values of 80 km height with the values of 100 km, the grid, and the world. The coverage area decreases to 76.13 % of the reference area. The covered area of the grid decreases by 4.63 % to 14.77 %, and the covered area of the world decreases from 0.72 % to 0.55 %.

Tab. 5–3: Comparison of the Coverage Area at 80 km and 100 km Height

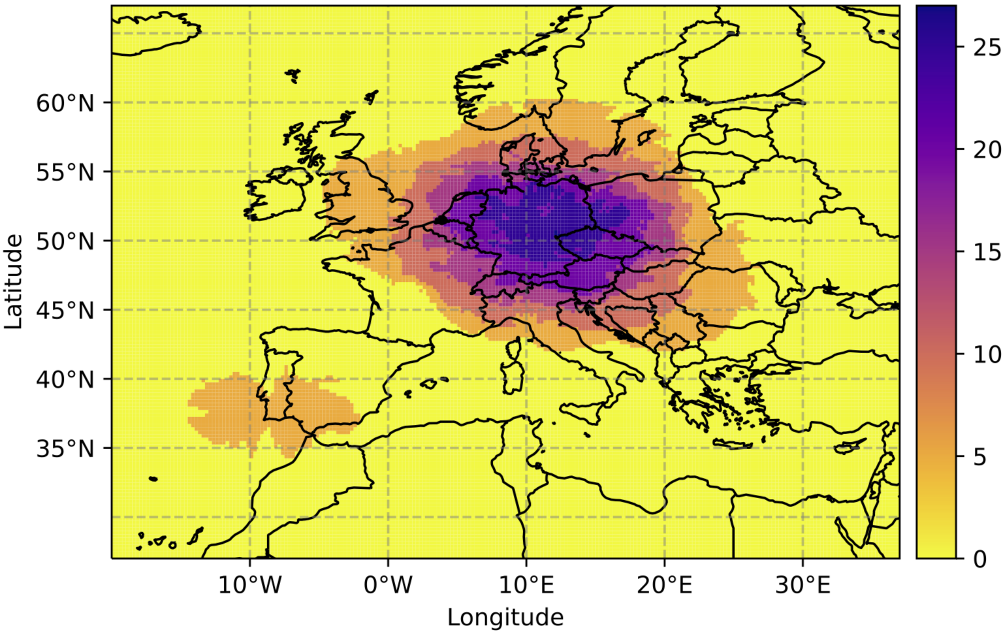
	Coverage 80 km	**Coverage 100 km**	**Grid (Europe)**	**World**
Area	2.87×10^6 km^2	3.77×10^6 km^2	19.43×10^6 km^2	526.2×10^6 km^2
Volume	1.15×10^8 km^3	1.51×10^8 km^3	7.77×10^8 km^3	210.48×10^8 km^3
Percentage	0 %	76.13 %	14.77 %	0.55 %

5.3.2 Comparison of Coverage Area at 120 km to 100 km Height

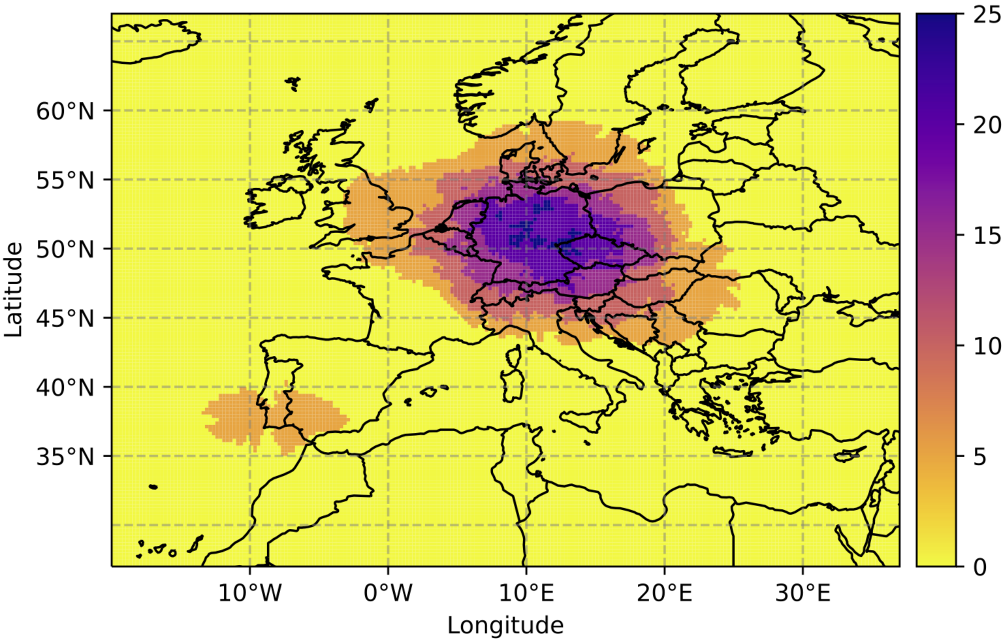
Figure 5–4 shows the coverage area in 120 km and 100 km height. The area covered by up to 27 cameras has greatly increased, and the minimum coverage of at least two cameras has also expanded. The coverage area has grown by 28.08 % compared to the reference height at 100 km. The covered area of the grid has increased from 19.4 % to almost 25 %, and the covered area of the world has increased from 0.72 % to 0.92 %.

Tab. 5–4: Comparison of the Coverage Area at 120 km and100 km Height

	Coverage 120 km	**Coverage 100 km**	**Grid (Europe)**	**World**
Area	4.83×10^6 km^2	3.77×10^6 km^2	19.43×10^6 km^2	526.2×10^6 km^2
Volume	1.93×10^8 km^3	1.51×10^8 km^3	7.77×10^8 km^3	210.48×10^8 km^3
Percentage	0 %	128.08 %	24.85 %	0.92 %

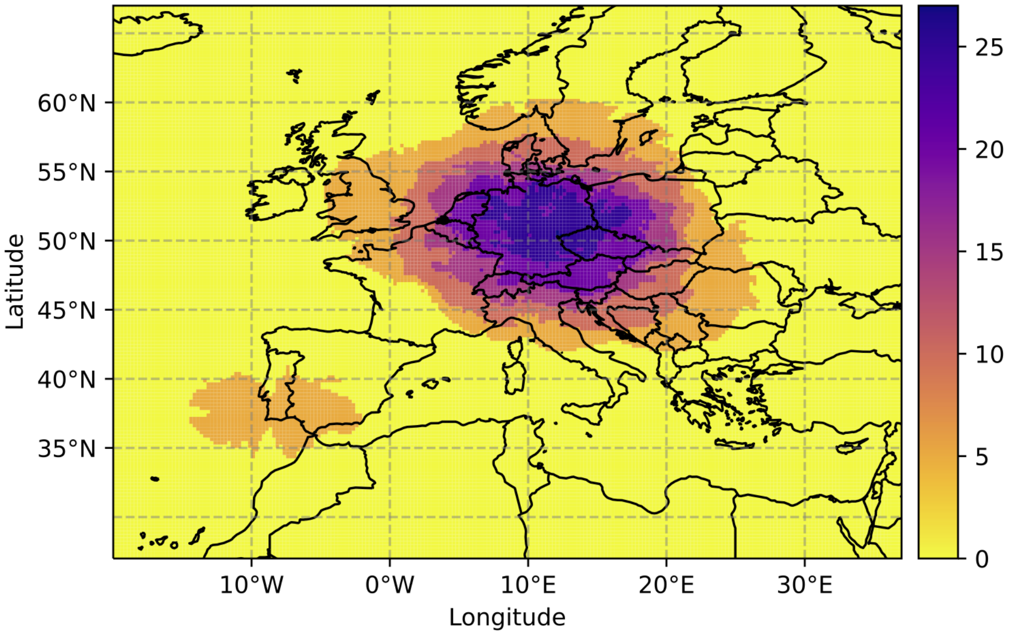


(a) Coverage Area at 100 km Height

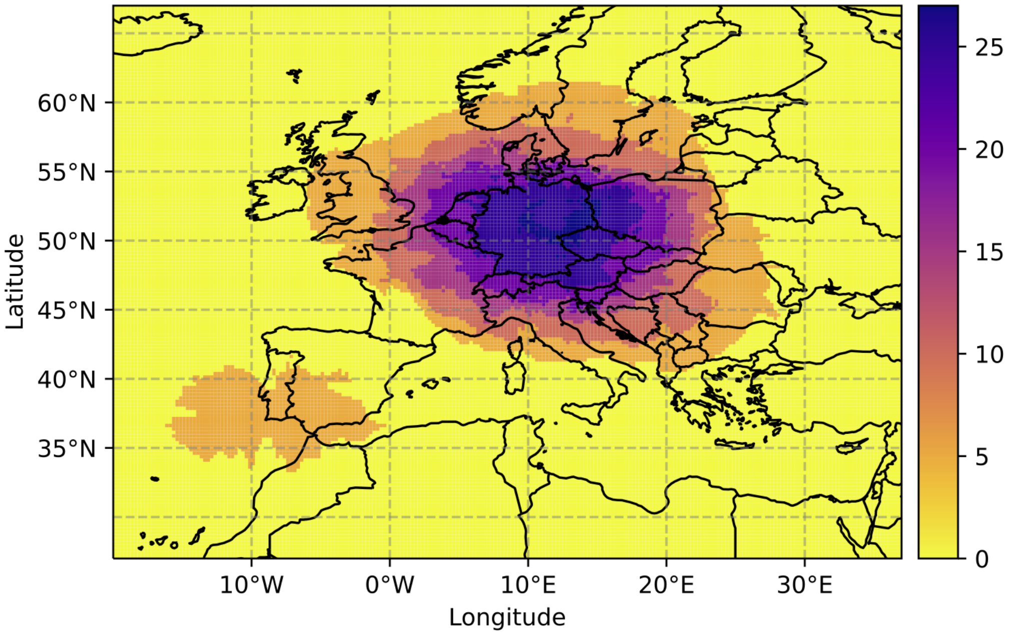


(b) Coverage Area at 80 km Height

Fig. 5–3: Comparison of the Coverage Area at 100 km and 80 km Height



(a) Coverage Area at 100 km Height



(b) Coverage Area at 120 km Height

Fig. 5–4: Comparison of the Coverage Area at 120 km and 100 km Height

The results showed that the spatial coverage over central Europe, especially Germany, is quite good for an altitude of about 100 km. In contrast, the coverage in the north and east of Europe is relatively poor. However, not all active cameras could be processed due to missing files in the archive. The cameras used to create the coverage profiles are listed in the appendix section A.1. Nevertheless, a coverage of almost 0.7 % of the world and 20 % of the grid is a good starting point.

It should be mentioned that the coverage presented here is not always visible due to clouds, so the coverage model needs to be extended to include considerations of precise sky time and cloud cover. In addition, brightness reduction due to light extinction in urban areas with heavy light pollution may be an essential factor. Furthermore, a decrease in brightness due to meteor motion must also be considered.

6 Conclusion

In this study, software was developed to determine the coverage of individual cameras. With the obtained data, a coverage profile for the entire network was created. A correction for the minimum elevation angle was performed to consider the influence of brightness reduction due to light extinction and the atmosphere. Subsequently, the results were analyzed on the influence of the brightness reduction and the observation altitude. Then the results were discussed. This chapter summarizes the work of the thesis and gives an outlook on further research.

6.1 Summary

A short introduction to the topic was given in chapter 1. The importance of meteor observation was explained, and the AllSky7 Fireball Network was introduced. Then a short outlook on the scope of this thesis was given.

Chapter 2 described the basics of meteor observations. The celestial and geographical coordinate system was explained, and a method to transform the coordinates was given. The last section dealt with the literature for determining the flux density of meteoroids and the work's significance.

Chapter 3 presented an approach to determine the coverage of a camera system. For the determination, mask images of the cameras are processed to determine horizon pixels above obstacles. The pixels are then converted to azimuth and elevation coordinates. With this data, the viewable distance is calculated for each azimuth angle. The resulting distance values were then visualized in a polar plot.

In chapter 4, the obtained data were transformed into geographical coordinates. The coverage data could then be plotted on a map. With the help of a grid, the area over Europe was discretized to determine network coverage. The grid ranged from −20° to 37° in longitude and from 27° to 67° in latitude. The cameras with coverage of the point were counted for each grid point. The resulting coverage grid was then plotted on a heat map. The coverage area was then adjusted to account for the brightness reduction at low elevation angles with a limiting angle.

Chapter 5 presented the results of the spatial coverage determination and analyzed the influence of the brightness reduction and differing altitudes. Under ideal conditions and an altitude of 100 km, the network covers 25.23 % of the grid and 0.93 % of the world. The coverage was then adjusted to brightness reductions with a limiting angle. The resulting reduced coverage covers 19.4 % of the grid and 0.72 % of the world. The altitude of observation was then analyzed by comparing the coverage at altitudes of 80 km and 120 km to a reference area at 100 km. At an altitude of 80 km the area decreases by 23.87 % and increases by 28.08 % at an altitude of 120 km.

6.2 Further Research

To calculate the flux density, the observed area, the observation time, and the number of meteors are needed. The presented results give information about the area but are only valid to clear sky conditions. Therefore an important research topic is the cloud cover above the cameras. The model should consider the visible area and the time frame at which the sky is visible. Furthermore, a more precise magnitude reduction model must be developed. The decrease in brightness due to light extinction can be modeled at the meteor level to yield more accurate values. Also, the light pollution in urban areas has to be considered, which is decreasing the magnitude too. Furthermore, since the meteors are moving, the light gets distributed on several pixels, reducing the overall magnitude. Another student researched to account for this problem. A decent counting method should be used to determine the meteor numbers.

Bibliography

[1] A. A. Chiarenza, A. Farnsworth, P. D. Mannion, D. J. Lunt, P. J. Valdes, J. V. Morgan, and P. A. Allison, “Asteroid impact, not volcanism, caused the end-Cretaceous dinosaur extinction,” *Proceedings of the National Academy of Sciences*, vol. 117, no. 29, pp. 17 084–17 093, Jul. 2020. [Online]. Available: https://pnas.org/doi/full/10.1073/pnas.2006087117

[2] S. Taylor, J. H. Lever, and R. P. Harvey, “Accretion rate of cosmic spherules measured at the South Pole,” *Nature*, vol. 392, no. 6679, pp. 899–903, Apr. 1998. [Online]. Available: http://www.nature.com/articles/31894

[3] “Notable Asteroid Impacts in Earth’s History.” [Online]. Available: https://www.planetary.org/notable-asteroid-impacts-in-earths-history

[4] “Risk List - NEO.” [Online]. Available: https://neo.ssa.esa.int/risk-list

[5] S. Molau, A. Knöfel, J. Strunk, and M. Kempf, “AllSky7.net.” [Online]. Available: https://www.allsky7.net/

[6] J. Rendtel, “Handbook for meteor observers 2014 edition, reprinted with minor changes in 2015, 2017,” p. 138, 2017.

[7] “International Astronomical Union | IAU.” [Online]. Available: https://www.iau.org/

[8] J. Powell, *Cosmic Debris*, ser. Astronomers’ Universe. Cham: Springer International Publishing, 2017. [Online]. Available: http://link.springer.com/10.1007/978-3-319-51016-3

[9] Z. Ceplecha, J. Borovicka, W. Elford, D. Revelle, and R. Hawkes, “Meteor Phenomena and Bodies,” *Space Science Reviews*, vol. 84, p. 145, Jan. 1998.

[10] R. Walraven, “Calculating the position of the sun,” *Solar Energy*, vol. 20, no. 5, pp. 393–397, 1978. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0038092X7890155X

[11] D. D. Koschny and D. E. Igenbergs, “A threat for Earth? – Or: NEOs for engineers and physicists,” p. 62, 2019.

[12] R. Koschack and J. Rendtel, “Determination of Spatial Number Density and Mass Index from Visual Meteor Observsations (II),” *WGN, the Journal of the International Meteor Organization*, vol. 18, pp. 119–140, 1990.

[13] L. R. Bellot Rubio, “Spatial Number Densities and Errors from Photographic Meteor Observations under Very High Activity,” *WGN, the Journal of the International Meteor Organization*, vol. 22, pp. 118–130, Aug. 1994.

[14] E. Grün, H. Zook, H. Fechtig, and R. Giese, “Collisional balance of the meteoritic complex,” *Icarus*, vol. 62, no. 2, pp. 244–272, May 1985. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0019103585901216

[15] P. Brown, R. E. Spalding, D. O. ReVelle, E. Tagliaferri, and S. P. Worden, “The flux of small near-Earth objects colliding with the Earth,”

Nature, vol. 420, no. 6913, pp. 294–296, Nov. 2002. [Online]. Available: http://www.nature.com/articles/nature01238

[16] G. Drolshagen, D. Koschny, S. Drolshagen, J. Kretschmer, and B. Poppe, “Mass accumulation of earth from interplanetary dust, meteoroids, asteroids and comets,” *Planetary and Space Science*, vol. 143, pp. 21–27, Sep. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0032063316302434

[17] S. Molau and G. Barentsen, “Real-Time Flux Density Measurements of the 2011 Draconid Meteor Outburst,” *Earth, Moon, and Planets*, vol. 112, no. 1-4, pp. 1–5, Aug. 2014. [Online]. Available: http://link.springer.com/10.1007/s11038-013-9425-3

[18] R. C. Blaauw, M. Campbell-Brown, and A. Kingery, “Optical meteor fluxes and application to the 2015 Perseids,” *Monthly Notices of the Royal Astronomical Society*, vol. 463, no. 1, pp. 441–448, Nov. 2016. [Online]. Available: https://academic.oup.com/mnras/article-lookup/doi/10.1093/mnras/stw1979

[19] D. Koschny, E. Drolshagen, S. Drolshagen, J. Kretschmer, T. Ott, G. Drolshagen, and B. Poppe, “Flux densities of meteoroids derived from optical double-station observations,” *Planetary and Space Science*, vol. 143, pp. 230–237, Sep. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0032063316302380

[20] D. S. Hayes and D. W. Latham, “A rediscussion of the atmospheric extinction and the absolute spectral-energy distribution of VEGA,” *The Astrophysical Journal*, vol. 197, p. 593, May 1975. [Online]. Available: http://adsabs.harvard.edu/doi/10.1086/153548

[21] M. C. Forbes, “A Detailed Investigation of Atmospheric Extinction via Vilnius Photometry,” *Baltic Astronomy*, vol. 5, pp. 281–295, 1996.

[22] E. Pakštienė and J. E. Solheim, “Atmospheric Extinction Corrections for WET Observations,” *Open Astronomy*, vol. 12, no. 2, Jan. 2003. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/astro-2017-0046/html

[23] D. W. E. Green, “Correcting for Atmospheric Extinction,” *International Comet Quarterly*, no. 14, pp. 55–59, 1992. [Online]. Available: http://www.icq.eps.harvard.edu/ICQExtinct.html

A First Appendix

A.1 Processed Stations

The stations used for the determination of the coverage are:

AMS 16, AMS 18, AMS 21, AMS 22, AMS 30, AMS 31, AMS 32, AMS 33, AMS 34, AMS 35, AMS 36, AMS 50, AMS 54, AMS 56, AMS 57, AMS 58, AMS 59, AMS 65, AMS 67, AMS 70, AMS 71, AMS 72, AMS 74, AMS 75, AMS 80, AMS 86, AMS 87, AMS 88, AMS 90, AMS 94, AMS 96, AMS 97, AMS 100.

Some active stations had missing calibration files and were not considered in this work as well as systems stationed in the USA.

A.2 Code

The data of this work was processed and visualized with the two python scripts shown below. The Horizon.py script was used for the data acquisition in chapter 3 and the Coverage.py script was used to compute and display the coverage of the network in chapter 4 and 5.

Horizon.py

```
import cv2
import os
import numpy as np
import math
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from scipy.integrate import trapz
from lib.PipeUtil import load_json_file
from lib.PipeAutoCal import XYtoRADec, AzEltoRADec
from datetime import date

# Round to the nearest number with the specified precision, and break ties by rounding up
def round_half_up(n, decimals=0):
    multiplier = 10 ** decimals
    return math.floor(n*multiplier + 0.5) / multiplier

# Go through all cal_files and selects the cal_file with the smallest residual
# pixel error
def get_best_cal_file(cal_files):
    best_cp = {}
    best_cf = None
    best_cp['total_res_px'] = 9999
```

```
    for data in cal_files:
        cf, _ = data
        cal_file = load_json_file(cf)

        if float(cal_file['total_res_px']) < float(best_cp['total_res_px']):
            print('RESET␣BEST␣CAL!', cal_file['total_res_px'])
            best_cp = cal_file
            best_cf = cf

    return(best_cf, best_cp)

# Calculate the distance and horizontal distance at a specific height
def calc_distance(data, H, min_elevation):
    Distance = []
    R = 6371
    r = R + H

    for point in data:
        if point[1] < min_elevation:
            point[1] = min_elevation
        phi = point[1]
        alpha = math.radians(90 + phi)
        beta = 90 - phi - math.degrees(math.asin(R * math.sin(alpha) / (R+H)))
        distance = 2 * math.pi * r * beta/360
        distance_h = math.sin(math.radians(beta))*r
        distance_g = 2 * math.pi * R * beta/360
        az = point[0]
        distance = float('{:1.3f}'.format(distance))
        distance_h = float('{:1.3f}'.format(distance_h))
        distance_g = float('{:1.3f}'.format(distance_g))
        beta = float('{:1.3f}'.format(beta))
        Distance.append([az, distance, distance_h, distance_g, beta])
    return(Distance)

# Merge the data points with equal az values
def reduce_data_points_low(data):
    v = 0
    red_data = []
    num = 1
    for element in data:
        if v == 0:
            v = element[0]
            sum = element[1]
        else:
```

```
            if v == element[0]:
                num += 1
                sum = sum + element[1]
            else:
                average = float('{:1.3f}'.format(sum / num))
                red_data.append([v, average])
                v = element[0]
                sum = element[1]
                num = 1
    average = float('{:1.3f}'.format(sum / num))
    red_data.append([v, average])
    return(red_data)

# Sum all points with same azimuth and calculate mean value
def reduce_data_points_up(data):
    data.sort()
    v = 0
    red_data = []
    num = 1
    sum = 0
    for element in data:
        if v == 0:
            v = element[0]
            sum = element[1]
        else:
            if v == element[0]:
                num += 1
                sum = sum + element[1]
            else:
                average = float('{:1.3f}'.format(sum / num))
                red_data.append([v, average])
                v = element[0]
                sum = element[1]
                num = 1
    average = float('{:1.3f}'.format(sum / num))
    red_data.append([v, average])
    return(red_data)

# Write a list of lists into a txt file
def write_data_to_txt(data, name, var, station_id):
    f = open(r'/home/ams/amscams/pipeline/Horizon/' + station_id
            + '/' + name + '.txt', 'w')
    for name in var:
        f.write(name + '␣␣')
    f.write('\n')
    for point in data:
        for element in point:
            f.write(str(element) + '␣')
```

```
        f.write(’\n’)
    f.close()

# Update the poly line variables in the cal_params file with the poly line
# variables of the multi_poly-AMSID-ID.info file
def update_poly_lines(station_id, id, cal_params):
    mpf = ’/mnt/ams2/cal/multi_poly-’ + station_id + ’-’ + id + ’.info’
    try:
        multi_poly = load_json_file(mpf)
    except FileNotFoundError:
        print(’multi-poly␣file␣not␣found’)
    else:
        x_poly = multi_poly[’x_poly’]
        y_poly = multi_poly[’y_poly’]
        x_poly_fwd = multi_poly[’x_poly_fwd’]
        y_poly_fwd = multi_poly[’y_poly_fwd’]

        cal_params[’x_poly’] = x_poly
        cal_params[’y_poly’] = y_poly
        cal_params[’x_poly_fwd’] = x_poly_fwd
        cal_params[’y_poly_fwd’] = y_poly_fwd
    return(cal_params)


# Merges the input data with the existing ’HorDat’ data
# If HorDat is None it appends the whole list
# The transition between the lists is realized by weighting the values
# depending on how deep they are in a picture
def merge_lower_horizon_weighted(data, HorDat):
    if HorDat == None:
        HorDat = data
    else:
        i = 0
        j = 0
        k = 1
        last_value = HorDat[-1][0] # Last value of existing List
        first_value = HorDat[0][0] # First value of existing List
        if first_value < 300:
            first_value = 360
        print(’fv:␣’ + str(first_value))
        print(’lv:␣’ + str(last_value))
        # Counts Overlapping data at start of list
        while data[i][0] <= last_value:
            i += 1
        # Counts Overlapping data at end of list
```

```python
        while data[-k][0] > first_value: # Only '>' because k starts with 1
            k += 1

        # If overlapping at start of list
        s_weight = i
        s_weight_l = s_weight
        s_weight_r = 1

        # If overlapping at end of list
        e_weight = k
        e_weight_l = e_weight
        e_weight_r = 1

        # Iterate through data considering all cases:
        # If element == last_value: calculate weighted elevation
        # If element between last_value and first_value: add element to list
        # If element >= first_value: calculate weighted elevation
        for element in data:
            if element[0] <= last_value:
                HorDat[-i][1] = float('{:1.3f}'.format((s_weight_r*element[1]
                                      + s_weight_l*HorDat[-i][1])
                                      / (s_weight+1)))
                s_weight_l -= 1
                s_weight_r += 1
                i -= 1
            if element[0] > last_value and element[0] < first_value:
                HorDat.append(element)
                last_value = HorDat[-1][0]
            if element[0] >= first_value:
                HorDat[j][1] = float('{:1.3f}'.format((e_weight_l*element[1]
                                     + e_weight_r*HorDat[j][1])
                                     / (e_weight+1)))
                e_weight_l -= 1
                e_weight_r += 1
                j += 1
    return(HorDat)

# If azimuth value already exists, the lower elevation value is taken
# If value does no exist, the value is added to HorDat
def merge_lower_horizon_optimist(data, HorDat):
    if HorDat == None:
        HorDat = data
```

```
    else:
        for element in data:
            exist = False
            for el in HorDat:
                if element[0] == el[0]:
                    if element[1] < el[1]:
                        el[1] = element[1]
                    exist = True
            if exist == False:
                HorDat.append(element)
    HorDat.sort()

    return(HorDat)

# If azimuth value already exists, the higher elevation value is
    taken
# If value does no exist, the value is added to HorDat
def merge_lower_horizon_pessimist(data, HorDat):
    if HorDat == None:
        HorDat = data
    else:
        for element in data:
            exist = False
            for el in HorDat:
                if element[0] == el[0]:
                    if element[1] > el[1]:
                        el[1] = element[1]
                    exist = True
            if exist == False:
                HorDat.append(element)
    HorDat.sort()

    return(HorDat)

# If azimuth value already exists, the upper value is taken
# If value does not exist, the value is added to HorDat
def merge_upper_horizon(data, HorDat):
    if HorDat == None:
        HorDat = []
        HorDat.append(data)
    else:
        for element in data:
            exist = False
            for el in HorDat:
                if element[0] == el[0]:
                    el[1] = element[1]
                    exist = True
            if exist == False:
```

```
                HorDat.append(element)
    HorDat.sort()
    return(HorDat)

# Iterate trough each column and go from top row down, search for bright
# pixel (k>=30)
# If bright pixel is found set hor variable to False, get az and el with
# XYtoRADec and append to a list
# If no bright pixel is found in a column the hor variable is still True,
# so the last pixel in the coulumn is taken
def get_lower_image_hor(mask_img, cal_file, cal_params, json_conf,
                        station_id, id):
    rows,cols,_ = mask_img.shape
    AzAl = []
    #AzEl = []

    for x in range(cols):
        hor = True
        for y in range(rows):
            k = mask_img[y,x]
            if np.any(k >= 30):
                hor = False
                _,_,ra,dec,raz,al = XYtoRADec(x,y-1,cal_file,
                                              cal_params,json_conf)
                AzAl.append([float('{:1.1f}'.format(raz)),
                             float('{:1.3f}'.format(al))])
                mask_img[y-1,x] = (0,0,255)
                break

        if hor:
            _,_,ra,dec,raz,al = XYtoRADec(x,y,cal_file,cal_params,json_conf)
            AzAl.append([float('{:1.1f}'.format(raz)),
                         float('{:1.3f}'.format(al))])
            mask_img[y,x] = (0,0,255)
    path = "/home/ams/amscams/pipeline/Horizon/" + station_id + "/masks/"
    cv2.imwrite(os.path.join(path , id + '_mask_new.png'), mask_img)
    return(AzAl)

# Get the horizon coordinates of an upper image (Cam6, Cam7)
def get_upper_image_hor(mask_img, cal_file, cal_params, json_conf,
                        station_id, id):
    mask_img, AzAl = get_upper_hor_tp(mask_img, cal_file,
```

```
        cal_params,
                                      json_conf)
    mask_img, AzAl = get_upper_hor_lr(mask_img, cal_file, cal_params,
                                      json_conf)
    AzAl.sort()
    path = "/home/ams/amscams/pipeline/Horizon/" + station_id + "/masks/"
    cv2.imwrite(os.path.join(path , id + '_mask_new.png'), mask_img)
    return(AzAl)

# Get the horizon coordinates of an upper image (iterate from top to bot)
def get_upper_hor_tp(mask_img, cal_file, cal_params, json_conf):
    AzAl = []
    rows,cols,_ = mask_img.shape
    for x in range(cols):
        wb = False
        for y in range(rows):
            k = mask_img[y,x]
            if wb == False:
                if np.any(k >= 30):
                    if y == 0:
                        wb = True
                        continue
                    else:
                        _,_,ra,dec,raz,al = XYtoRADec(x,y-1,cal_file,
                                                      cal_params,json_conf)
                        AzAl.append([float('{:1.1f}'.format(raz)),
                                    float('{:1.3f}'.format(al))])
                        mask_img[y-1,x] = (0,0,255)
                        wb = True
            if wb:
                if np.all(k < 30):
                    _,_,ra,dec,raz,al = XYtoRADec(x,y,cal_file,
                                                  cal_params,json_conf)
                    AzAl.append([float('{:1.1f}'.format(raz)),
                                float('{:1.3f}'.format(al))])
                    mask_img[y,x] = (0,0,255)
                    wb = False
    return(mask_img, AzAl)

# Get the horizon coordinates of an upper image (iterate from left to right)
def get_upper_hor_lr(mask_img, cal_file, cal_params, json_conf):
    AzAl = []
```

```
    rows,cols,_ = mask_img.shape
    for y in range(rows):
        wb = False
        for x in range(cols):
            k = mask_img[y,x]
            if wb == False:
                if np.any(k >= 30):
                    if k[0] == 0 and k[1] == 0 and k[2] == 255:
                        continue
                    elif y == 0:
                        wb = True
                        continue
                    else:
                        _,_,ra,dec,raz,al =
                           XYtoRADec(x,y-1,cal_file,
                                                     cal_params,json_conf)
                        AzAl.append([float('{:1.1f}'.format(raz)),
                                    float('{:1.3f}'.format(al))])
                        mask_img[y-1,x] = (0,0,255)
                        wb = True
            if wb:
                if np.all(k < 30):
                    _,_,ra,dec,raz,al = XYtoRADec(x,y,cal_file,
                                                 cal_params,json_conf)
                    AzAl.append([float('{:1.1f}'.format(raz)),
                                float('{:1.3f}'.format(al))])
                    mask_img[y,x] = (0,0,255)
                    wb = False
    return(mask_img, AzAl)

# Get the horizon coordinates of the edge of an upper image
def get_upper_edge(mask_img, cal_file, cal_params, json_conf):
    AzAl_edge = []
    rows,cols,_ = mask_img.shape
    # Search for horizon and color it red
    for x in range(cols):
        for y in range(rows):
            k = mask_img[y,x]
            if x == 0 or x == (cols -1) or y == (rows - 1):
                if np.all(k <= 30):
                    _,_,ra,dec,raz,al = XYtoRADec(x,y,cal_file,
                                                   cal_params,json_conf)
                    AzAl_edge.append([float('{:1.1f}'.format(raz)),
                                float('{:1.3f}'.format(al))])
    return(AzAl_edge)

# Ask for the height of Calculations
def get_height():
```

```
    while True:
        try:
            height = int(input('Please␣enter␣the␣height␣in␣kilometers:␣'))
        except ValueError:
            print('Incorrect␣input,␣please␣enter␣a␣positive␣number.')
            continue

        if height < 0:
            print('Please␣enter␣a␣positive␣number.')

        else:
            print('Height␣set␣to␣' + str(height) + '␣kilometers')
            break

    return(height)

# Ask for input of mode to merge data
def get_mode():
    while True:
        try:
            mode = int(input('1␣=␣optimist␣\n2␣=␣pessimist␣\n3␣=␣weighted␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\nPlease␣enter␣the␣mode␣to␣merge␣the␣horizon␣data:␣'))
        except ValueError:
            print('Invalid␣input.')
            continue

        if mode == 1:
            mode = 'optimist'
            break

        if mode == 2:
            mode = 'pessimist'
            break

        if mode == 3:
            mode = 'weighted'
            break

        else:
            print('Mode␣does␣not␣exist,␣please␣enter␣1,␣2␣or␣3.')
            continue

    return(mode)

```

```
# Transform polar coordinates into cartesian coordinates
def pol2cart(rho, phi):
    x = rho * np.cos(phi)
    y = rho * np.sin(phi)
    return(x, y)

# Calculate area of polar horizon with trapz function
def calc_area(data):
    new_data = []
    for el in data:
        new_data.append(pol2cart(el[1], math.radians(el[0])))

    x = []
    y = []
    for el in new_data:
        x.append(el[0])
        y.append(el[1])

    area = trapz(x, y) #switched x, y so integral gets positive
    return(area)

# Calculate area of polar horizon by adding triangle areas of 2 data points
def int_pol(data):
    A = 0
    l = len(data)
    i = 0
    while i < (l-1):
        g = data[i][1]
        g2 = data[i+1][1]
        phi = data[i+1][0] - data[i][0]
        if g < g2:
            v = g
            g = g2
            g2 = v
        h = g2 * math.radians(np.sin(phi))
        dA = 1/2 * g * h
        A = A + dA
        i += 1
    g = data[-1][1]
    g2 = data[0][1]
    phi = (360.0 - data[-1][0]) + (data[0][0])
    if g < g2:
        v = g
        g = g2
        g2 = v
    h = g2 * math.radians(np.sin(phi))
    dA = 1/2 * g * h
```

```
    A = A + dA
    return(A)

# Calculate area of polar horizon by integration of sphere volume
def int_pol_sphere(data, H, H0, H1):
    A = 0
    V = 0
    l = len(data)
    i = 0
    R = 6371
    r = R + H
    r0 = R + H0
    r1 = R + H1
    while i < (l-1):
        phi = math.radians(data[i+1][0] - data[i][0])
        theta = math.radians(data[i][4])
        dA = r*r * phi * (-np.cos(theta) + 1)
        A = A + dA
        dV = 1/3 * phi * (r0*r0*r0 * (math.cos(theta) - 1)
             + r1*r1*r1 * (1 - math.cos(theta)))
        V = V + dV
        i += 1
    phi = math.radians((360.0 - data[-1][0]) + (data[0][0]))
    theta = math.radians(data[-1][4])
    dA = r*r * phi * (-np.cos(theta) + 1)
    A = A + dA
    dV = 1/3 * phi * (r0*r0*r0 * (math.cos(theta) - 1)
         + r1*r1*r1 * (1 - math.cos(theta)))
    V = V + dV
    return(A, V)

# Get cal data from allsky7 archive
def get_cal_data(station_id, id, json_conf):
    #station_id = ’AMS74’
    print(id)

    # Load cal_range file and select first element with the same
       cam id
    cal_range = ’/mnt/archive.allsky.tv/’ + station_id + ’/CAL/’ +
       station_id\
                + ’_cal_range.json’
    try:
        cr = load_json_file(cal_range)
    except FileNotFoundError:
        print(’cal_range␣file␣not␣found’)
        error.write(station_id + ’_cal_range.json␣not␣found’ +
           ’\n’)
        return(None)
```

```
    else:
        cal_param = {}
        cp = None
        for element in cr:
            if element[0] == id:
                cp = element
                break

    # Load cal_params from cal_range.json file into a dictionary
    if cp is not None:
        cal_param['center_az'] = cp[3]
        cal_param['center_el'] = cp[4]
        cal_param['position_angle'] = cp[5]
        cal_param['pixscale'] = cp[6]
        cal_param['total_res_px'] = cp[7]
    else:
        print('cal_params␣do␣not␣exist')
        error.write(station_id + '_' + id + '␣cal_params␣do␣not␣exist' + '\n')
        return(None)

    # Load cal_history.json file to find a cal_file for the cam id
    cal_files = '/mnt/archive.allsky.tv/' + station_id + '/CAL/' + station_id\
              + '_cal_history.json'
    try:
        cf = load_json_file(cal_files)
    except FileNotFoundError:
        print('cal_history␣file␣not␣found')
        error.write(station_id + '_cal_history.json␣not␣found' + '\n')
        return(None)
    else:
        cal_file = cf[id]['cal_files']
        if len(cal_file) > 0:
            cal_file = cal_file[-1]
        else:
            print('cal_files␣do␣not␣exist')
            error.write(station_id + '_' + id + '␣cal_files␣do␣not␣exist' + '\n')
            return(None)

    # Calculate ra_center and dec_center with AzEltoRADec function
    # from cal_file
    ra, dec = AzEltoRADec(cal_param['center_az'], cal_param['center_el'],
                          cal_file, cal_param, json_conf)

```

```
    cal_param['ra_center'] = math.degrees(ra)
    cal_param['dec_center'] = math.degrees(dec)

    # Load missing cal_params from LENS_MODEL.json into dictionary
    mpf = '/mnt/archive.allsky.tv/' + station_id + '/CAL/' \
                + station_id + '_' + id + '_LENS_MODEL.json'
    try:
        cal_params = load_json_file(mpf)
    except FileNotFoundError:
        print('lens_model␣file␣not␣found')
        error.write(station_id + '_' + id + '_LENS_MODEL.json␣not␣found' + '\n')
        return(None)
    else:
        cal_params.update(cal_param)

    return(cal_params, cal_file)

# Calculates the longitude and latitude with given camera position,
# azimuth and distance
def AzEltoLonLat(data, station_lon, station_lat, H):
    LonLat = []
    R = 6371 + H

    for element in data:
        az_deg = element[0]
        dist = element[1]
        b = dist/R
        c = math.radians(90 - station_lat)
        az = math.radians(az_deg)
        a = math.acos(math.cos(b)*math.cos(c) + math.sin(c)*\
            math.sin(b)*math.cos(az))
        B = math.asin(math.sin(b)*math.sin(az)/math.sin(a))
        Lat = 90 - math.degrees(a)
        Lon = math.degrees(B) + station_lon
        LonLat.append([az_deg, float('{:1.3f}'.format(Lon)),
                        float('{:1.3f}'.format(Lat))])

    return(LonLat)

# Plot the horizontal distance data in polar coordinates
def plot_distance(data, station_id):
    rad = []
    r = []

    for point in data:
        rad.append(math.radians(point[0]))
        r.append(point[2])
```

```

    plt.figure()
    ax = plt.subplot(1, 1, 1, projection='polar')
    ax.plot(rad,r,'g.-', markersize=1, linewidth=1)
    ax.set_theta_zero_location("N")  # theta=0 at the top
    ax.set_theta_direction(-1)
    ax.set_xlabel('Distance␣in␣[km]')
    plt.savefig('/home/ams/amscams/pipeline/Horizon/' + station_id
                + '/horizon.pdf')
    #plt.show()

# Plot the horizontal distance in cartesian coordinates
def plot_hor_line(data, station_id):
    x = []
    y = []
    for point in data:
        x.append(point[0])
        y.append(point[1])

    plt.figure()
    ax = plt.subplot(1, 1, 1, projection = None)
    ax.plot(x, y, 'g.-', linewidth=1, markersize=1)
    ax.set_ylim([0, 60])
    ax.set_xlabel('Azimuth␣in␣°[]')
    ax.set_xlabel('Elevation␣in␣°[]')
    plt.savefig('/home/ams/amscams/pipeline/Horizon/' + station_id
                + '/horizon_line.pdf')
    #plt.show()

# Plot longitude and latitude on scatter plot
def plot_LonLat(data, station_id):
    lon = []
    lat = []
    for point in data:
        lon.append(point[1])
        lat.append(point[2])

    plt.figure()
    ax = plt.subplot(1, 1, 1, projection = None)
    ax.plot(lon, lat, 'g.-', linewidth=1, markersize=1)
    ax.set_xlabel('Longitude␣in␣°[]')
    ax.set_xlabel('Latitude␣in␣°[]')
    plt.savefig('/home/ams/amscams/pipeline/Horizon/' + station_id
                + '/Coverage.pdf')
    #plt.show()

# Plot longitude and latitude on scatter plot with cartopy
def plot_LonLat_cart(data, station_id, station_lon, station_lat):
```

```
    lon = []
    lat = []

    for point in data:
        lon.append(point[1])
        lat.append(point[2])

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
    ax.plot(lon, lat, ’g.-’, linewidth=1, markersize=1)
    ax.plot(station_lon, station_lat, ’go’, markersize=3)
    ax.gridlines()
    ax.add_feature(cfeature.BORDERS)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.OCEAN, facecolor=(0.5,0.5,0.5))
    plt.savefig(’/home/ams/amscams/pipeline/Horizon/’ + station_id
                + ’/Coverage_cart.pdf’)
    #plt.show()

# Plot longitude and latitude on scatter plot
def plot_LonLat_stations(data):
    plt.title(’AllSky7␣Coverage’)
    plt.xlabel(’Longitude’)
    plt.ylabel(’Latitude’)
    markers = [’g.-’, ’r.-’, ’b.-’, ’y.-’]

    plt.figure()
    for element in data:
        lon = []
        lat = []
        for point in data:
            lon.append(point[1])
            lat.append(point[2])

        plt.plot(lon, lat, ’g.-’, linewidth=1, markersize=1)
        plt.grid()

    #plt.savefig(’/home/ams/amscams/pipeline/Horizon/Coverage.pdf’)
    #plt.show()

# Get list with station_ids in archive
def get_station_ids(path):
    archive_dirs = os.listdir(path)
    station_ids = []
    for element in archive_dirs:
        if ’AMS’ in element:
            station_ids.append(element)
    return(station_ids)
```

```python

# Load config file, extract cam ids, get device position
def get_station_information(station_id):
    json_file = '/mnt/archive.allsky.tv/' + station_id + '/CAL/as6.json'
    if os.path.exists(json_file):
        if os.stat(json_file).st_size > 0:
            json_conf = load_json_file(json_file)
            cams = json_conf['cameras']
            cams_id = []
            lat = float(json_conf['site']['device_lat'])
            lon = float(json_conf['site']['device_lng'])
            alt = float(json_conf['site']['device_alt'])
        else:
            print('conf␣file␣empty')
            error.write(station_id + '␣as6.json␣is␣empty' + '\n')
            return(None)
    else:
        print('conf␣file␣not␣found')
        error.write(station_id + '␣as6.json␣not␣found' + '\n')
        return(None)


    # Create list of cam ids
    for cam in cams:
        cam_id = json_conf['cameras'][cam]['cams_id']
        cams_id.append(cam_id)

    id0 = cams_id[0] # first cam id
    return(lat, lon, alt, cams_id, id0, json_conf)

# Load config file, extract cam ids, get device position
def get_system_health(station_id):
    json_file = '/mnt/archive.allsky.tv/' + station_id \
                + '/' + station_id + '_system_health.json'
    if os.path.exists(json_file):
        if os.stat(json_file).st_size > 0:
            json_conf = load_json_file(json_file)
            last_update = json_conf['last_update']
            date_time = last_update.split("_")
            year = int(date_time[0])
            month = int(date_time[1])
            day = int(date_time[2])
        else:
            print('system_health␣file␣empty')
            #error.write(station_id + ' systen_health.json is empty' + '\n')
            return(None)
```

```
    else:
        print('system_health␣file␣not␣found')
        #error.write(station_id + ' as6.json not found' + '\n')
        return(None)
    today = date.today()
    d0 = date(year, month, day)
    d1 = today
    delta = d1 - d0
    return(delta.days)

# Open and resize mask images
def get_mask(station_id, id):
    mask_path = ('/mnt/archive.allsky.tv/' + station_id + '/CAL/MASKS/'
                 + id + '_mask.png')

    mask_img = cv2.imread(mask_path)
    if mask_img is not None:
        mask_img = cv2.resize(mask_img, (1920,1080))
    else:
        print('mask␣file␣not␣found')
        error.write(station_id + '_' + id + '_mask.png␣not␣found' + '\n')
        return(None)
    return(mask_img)

def get_horizon_data(cams_id, station_id, json_conf, id0, mode):
    HorDat = None
    for id in cams_id:
        print(id)

        cal_data = get_cal_data(station_id, id, json_conf)
        if cal_data is not None:
            cal_params, cal_file = cal_data
        else:
            return(None)

        mask_img = get_mask(station_id, id)

        if mask_img is not None:
            if int(id) - int(id0) < 5:
                AzAl = get_lower_image_hor(mask_img, cal_file, cal_params,
                                           json_conf, station_id, id)
            else:
                AzAl = get_upper_image_hor(mask_img, cal_file, cal_params,
```

```
                                           json_conf, station_id, id)
        else:
            return(None)

        # Saving AzAl data to text file
        name = 'AzAl' + id
        write_data_to_txt(AzAl, name, ['az','al'], station_id)
        print('cam ' + id + ' data generated')

        # Reduce data points
        if int(id) - int(id0) < 5:
            red_AzAl = reduce_data_points_low(AzAl)
            name = 'red_AzAl' + id
            red_var = ['az', 'al']
            write_data_to_txt(red_AzAl, name, red_var, station_id)
        else:
            if len(AzAl) > 0:
                red_AzAl = reduce_data_points_up(AzAl)
                name = 'red_AzAl' + id
                red_var = ['az', 'al']
                write_data_to_txt(red_AzAl, name, red_var, station_id)

        # Create horizon data
        if int(id) - int(id0) < 5:
            if mode == 'optimist':
                HorDat = merge_lower_horizon_optimist(red_AzAl, HorDat)
            if mode == 'pessimist':
                HorDat = merge_lower_horizon_pessimist(red_AzAl, HorDat)
            if mode == 'weighted':
                HorDat = merge_lower_horizon_weighted(red_AzAl, HorDat)
        else:
            HorDat = merge_upper_horizon(red_AzAl, HorDat)
    return(HorDat)

# Calculate the minimum angle for the limiting magnitude and the meteor level H
def get_limiting_angle(alt, lim_mag, H):
    R = 6371
    d_1 = H # meteor level
    h = alt/1000
    alpha_0 = 1.3
    wave_len = 0.51 # lambda in mircrons

```

```
    A_ray = 0.1451 * math.exp(-h/7.996)
    A_aer = 0.05 * wave_len**(-alpha_0) * math.exp(-h/1.5)
    A_oz = 0.016
    A_star = A_ray + A_aer + A_oz
    z = 85
    mag_red = 99
    while mag_red > 4:
        X = 1/(math.cos(math.radians(z)) + 0.025*math.exp(-11*math.cos(math.radians(z))))
        A = X * A_star

        epsilon = 90 - z
        beta = math.degrees(math.acos(R * math.cos(math.radians(epsilon))/(R+H))) - epsilon
        d_2 = (R + H) * math.sin(math.radians(beta))/math.cos(math.radians(epsilon))
        mag_dist = 5 * math.log10(d_2/d_1)
        mag_delta = mag_dist + A
        mag_red = lim_mag + mag_delta
        z -= 1
    return(epsilon)

def remove_nan(data):
    nan_removed = 0
    for element in data:
        if math.isnan(element[0]) or math.isnan(element[1]):
            data.remove(element)
            nan_removed += 1
    return(data)

def get_active_stations(station_ids):
    active = []
    notactive = []
    missing = []
    for station_id in station_ids:
        days = get_system_health(station_id)
        if days is not None:
            if days < 30:
                active.append(station_id)
            else:
                notactive.append(station_id)
                continue
        else:
            missing.append(station_id)
            continue
    return(active, notactive, missing)

def stations_to_text(data, name):
```

```
    f = open(r’/home/ams/amscams/pipeline/Horizon/’ + name + ’.txt’, ’w’)
    for station in data:
        f.write(str(station) + ’\n’)
    f.close()

# station_ids = [’AMS80’]
# json_file = ’/home/ams/amscams/conf/as6.json’
id_path = ’/mnt/archive.allsky.tv/’
station_ids = get_station_ids(id_path)
remove_list = [’AMS1’, ’AMS41’, ’AMS42’, ’AMS48’, ’AMS129’]
station_ids = [x for x in station_ids if x not in remove_list]

act_stations, nact_stations, miss_stations = get_active_stations(station_ids)
stations_to_text(act_stations, ’active_stations’)
stations_to_text(nact_stations, ’non_active_stations’)
stations_to_text(miss_stations, ’missing_stations’)

# Create Error File
error = open(r’/home/ams/amscams/pipeline/Horizon/error.txt’, ’w’)
processed_stations = open(r’/home/ams/amscams/pipeline/Horizon/processed_stations.txt’, ’w’)

#mode = get_mode()
mode = ’optimist’
mag_zenith = -3
H = 100 # meteor level

for station_id in station_ids:
    # Get station information
    station_inf = get_station_information(station_id)
    if station_inf is not None:
        station_lat, station_lon, alt, cams_id, id0, json_conf = station_inf
    else:
        continue

    # Create station folder if it does not exist
    file_path = ’/home/ams/amscams/pipeline/Horizon/’ + station_id + ’/’
    os.makedirs(os.path.dirname(file_path), exist_ok=True)

    # Open mask files and search for horizon
    HorDat = get_horizon_data(cams_id, station_id, json_conf, id0, mode)
    if HorDat is not None:
```

```
        remove_nan(HorDat)

    if HorDat is not None:
        # Save horizon data in text file
        write_data_to_txt(HorDat, ’HorDat’, [’az’, ’al’], station_id)
    else:
        if len(os.listdir(file_path)) == 0:
            os.rmdir(file_path)
        continue

    #height = get_height()

    min_elevation = get_limiting_angle(alt, mag_zenith, H)
    #min_elevation = 0
    print(min_elevation)

    # Distance at H km height
    Distance = calc_distance(HorDat, H, min_elevation)
    # Distance at 50 km height
    Distance50 = calc_distance(HorDat, 50, min_elevation)
    # Distance at 1500 km height
    Distance150 = calc_distance(HorDat, 150, min_elevation)

    write_data_to_txt(Distance, ’Distance’,
                      [’az’, ’dist’, ’dist_h’, ’dist_g’, ’beta’], station_id)

    plot_distance(Distance, station_id)

    Area = calc_area(Distance)
    print(Area)
    Area2 = int_pol(Distance)
    print(Area2)
    Area3, Volume3 = int_pol_sphere(Distance, H, H-20, H+20)
    print(Area3, Volume3)
    Area50, Volume50 = int_pol_sphere(Distance50, 50, 30, 70)
    Area150, Volume150 = int_pol_sphere(Distance150, 150, 130, 170)
    f = open(r’/home/ams/amscams/pipeline/Horizon/’ + station_id
            + ’/area.txt’, ’w’)
    f.write(’area␣=␣’ + str(Area3) + ’²km␣␣␣␣␣’ + str(Area50) + ’²km␣␣␣␣␣’
            + str(Area150) + ’²km’ + ’\n’)
    f.write(’volume␣=␣’ + str(Volume3) + ’³km␣␣␣␣␣’ + str(Volume50)
            + ’³km␣␣␣␣␣’ + str(Volume150) + ’³km’)
    f.close()

    LonLat = AzEltoLonLat(Distance, station_lon, station_lat, H)
```

```
    write_data_to_txt(LonLat, ’LonLat’, [’az’, ’lon’, ’lat’], station_id)
    plot_hor_line(HorDat, station_id)
    plot_LonLat_cart(LonLat, station_id, station_lon, station_lat)
    processed_stations.write(station_id + ’\n’)
processed_stations.close
error.close()
```

Coverage.py

```
from asyncore import write
import math
import numpy as np
import os
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from mpl_toolkits.axes_grid1 import make_axes_locatable
from cartopy.mpl.ticker import (LongitudeFormatter, LatitudeFormatter,
                                LatitudeLocator, LongitudeLocator)
from lib.PipeUtil import load_json_file

# Load config file, extract cam ids, get device position
def get_station_information(station_id):
    json_file = ’/mnt/archive.allsky.tv/’ + station_id + ’/CAL/as6.json’
    if os.path.exists(json_file):
        if os.stat(json_file).st_size > 0:
            json_conf = load_json_file(json_file)
            cams = json_conf[’cameras’]
            cams_id = []
            lat = float(json_conf[’site’][’device_lat’])
            lon = float(json_conf[’site’][’device_lng’])
            alt = float(json_conf[’site’][’device_alt’])
        else:
            print(’conf␣file␣empty’)
            return(None)
    else:
        print(’conf␣file␣not␣found’)
        return(None)


    # Create list of cam ids
    for cam in cams:
        cam_id = json_conf[’cameras’][cam][’cams_id’]
        cams_id.append(cam_id)

```

```
    id0 = cams_id[0] # first cam id
    return(lat, lon, alt, cams_id, id0, json_conf)

# Calculate the distance and horizontal distance at a specific height
def get_distance(data, H, min_elevation):
    Distance = []
    R = 6371
    r = R + H
    for point in data:
        if point[1] < min_elevation:
            point[1] = min_elevation
        phi = point[1]
        alpha = math.radians(90 + phi)
        beta = 90 - phi - math.degrees(math.asin(R*math.sin(alpha)/(r)))
        distance = math.radians(beta)*r
        distance_h = math.sin(math.radians(beta))*r
        distance_g = math.radians(beta)*R
        az = point[0]
        distance = round_half_up(distance, 3)
        distance_h = round_half_up(distance_h, 3)
        distance_g = round_half_up(distance_g, 3)
        beta = round_half_up(beta, 3)
        Distance.append([az, distance, distance_h, distance_g, beta])
    return(Distance)

# Calculates the longitude and latitude with given camera position,
# azimuth and distance
def AzEltoLonLat(data, station_lon, station_lat, H):
    LonLat = []
    R = 6371 + H

    for element in data:
        az_deg = element[0]
        dist = element[1]
        beta = element[4]
        b = dist/R
        c = math.radians(90 - station_lat)
        az = math.radians(az_deg)
        a = math.acos(math.cos(b)*math.cos(c) + math.sin(c)*\
            math.sin(b)*math.cos(az))
        B = math.asin(math.sin(b)*math.sin(az)/math.sin(a))
        Lat = 90 - math.degrees(a)
        Lon = math.degrees(B) + station_lon
        LonLat.append([az_deg, round_half_up(Lon, 3),
                         round_half_up(Lat, 3), dist, beta])
```

```

    return(LonLat)

def round_half_up(n, decimals=0):
    multiplier = 10 ** decimals
    return math.floor(n*multiplier + 0.5) / multiplier

def write_data_to_txt(data, name, var, station_id):
    f = open(r’/home/ams/amscams/pipeline/Horizon/’ + station_id
             + ’/’ + name + ’.txt’, ’w’)
    for name in var:
        f.write(name + ’␣␣’)
    f.write(’\n’)
    for point in data:
        for element in point:
            f.write(str(element) + ’␣’)
        f.write(’\n’)
    f.close()

# Create an empty grid
def get_grid(lat_range, lon_range, steps):
    lon_start = lon_range[0]
    lat_start = lat_range[0]
    lon_end = lon_range[1]
    lat_end = lat_range[1]
    n = 3
    step = 1/steps
    width = (lon_end - lon_start)*steps + 1
    height = (lat_end - lat_start)*steps + 1
    grid = [[[0 for k in range(n)] for j in range(width)]
            for i in range(height)]

    lat = lat_end
    for latitude in grid:
        lon = lon_start
        for longitude in latitude:
            longitude[0] = lat
            longitude[1] = lon
            if lon < lon_end:
                lon += step
        if lat > lat_start:
            lat -= step
    return(grid)

def get_az_and_dist(station_lat, station_lon, lat, lon, H):
    R = 6371 + H
    a = math.radians(90 - lat)
    c = math.radians(90 - station_lat)
```

```
    B = math.radians(lon - station_lon)
    b = math.acos(math.cos(a) * math.cos(c)
                  + math.sin(a) * math.sin(c) * math.cos(B))
    az = math.acos((math.cos(a)-(math.cos(b)*math.cos(c)))
                   /(math.sin(c)*math.sin(b)))
    az_deg = math.degrees(az)
    dist = b * R
    if lon < station_lon:
        az_deg = 360 - az_deg
    return(round_half_up(az_deg, 2),round_half_up(dist, 3))

# Calculate average
def get_average(data, value):
    lat_sum = 0
    lon_sum = 0
    dist_sum = 0
    length = len(data)
    for element in data:
        lon_sum = lon_sum + element[1]
        lat_sum = lat_sum + element[2]
        dist_sum = dist_sum + element[3]
    lat_avg = lat_sum/length
    lon_avg = lon_sum/length
    dist_avg = dist_sum/length
    average = [value, lon_avg, lat_avg, dist_avg]
    return(average)

# Get nearest point
def get_minimum(value, points):
    minimum = 1000
    nearest_points = []
    for element in points:
        difference = abs(value - element[0])
        difference = round_half_up(difference, 2)
        if difference < minimum:
            minimum = difference
            nearest_point = element
        elif difference == minimum:
            nearest_points.append(nearest_point)
            nearest_point = element
    nearest_points.append(nearest_point)
    nearest_point = get_average(nearest_points, value)
    return(nearest_point)

# Get coverage of cam
def get_coverage(data, grid, station_lat, station_lon, H):
    for latitude in grid:
        for longitude in latitude:
```

```
            az, dist_gp = get_az_and_dist(station_lat, station_lon,
                                          longitude[0], longitude[1], H)
            nearest_points = []
            for element in data:
                if int(az) == int(element[0]):
                    nearest_points.append(element)
            if nearest_points:
                nearest_point = get_minimum(az, nearest_points)
            else:
                for element in data:
                    if abs(int(az) - int(element[0])) < 6:
                        nearest_points.append(element)
                if nearest_points:
                    nearest_point = get_minimum(az, nearest_points)
                else:
                    print(’Point:␣[’ + str(longitude[0]) + ’,␣’
                        + str(longitude[1]) + ’]␣is␣not␣in␣range␣5’)
                    continue
            if dist_gp < nearest_point[3]:
                longitude[2] += 1
    return(grid)

# Plot longitude and latitude on scatter plot with cartopy
def plot_LonLat_cart(data, station_id, station_lon, station_lat):
    lon = []
    lat = []

    for point in data:
        lon.append(point[1])
        lat.append(point[2])

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
    ax.plot(lon, lat, ’g.-’, linewidth=1, markersize=1)
    ax.plot(station_lon, station_lat, ’go’, markersize=3)
    ax.gridlines()
    ax.add_feature(cfeature.BORDERS)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.OCEAN, facecolor=(0.5,0.5,0.5))
    plt.savefig(’/home/ams/amscams/pipeline/Horizon/’ + station_id
                + ’/Coverage_cart.pdf’)
    #plt.show()

# Plot heatmap of coverage grid
def plot_heatmap(grid):
    # generate 2 2d grids for the x & y bounds
```

```
    lat_start = grid[-1][0][0]
    lat_end = grid[0][0][0]
    lon_start = grid[0][0][1]
    lon_end = grid[0][-1][1]
    width = len(grid[0])
    height = len(grid)

    x = np.linspace(lon_start, lon_end, width)
    y = np.linspace(lat_start, lat_end, height)
    X, Y = np.meshgrid(x,y)
    Z = np.zeros((height, width))

    for i in range(height):
        for j in range(width):
            Z[i,j] = grid[-i][j][2]

    # x and y are bounds, so z should be the value *inside* those bounds.
    # Therefore, remove the last value from the z array.
    Z = Z[:-1, :-1]
    z_min, z_max = 0, np.abs(Z).max()

    fig, ax = plt.subplots()

    c = ax.pcolormesh(X, Y, Z, cmap='Reds', vmin=z_min, vmax=z_max)
    #ax.set_title('pcolormesh')
    # set the limits of the plot to the limits of the data
    ax.axis([x.min(), x.max(), y.min(), y.max()])
    fig.colorbar(c, ax=ax)

    plt.savefig('/home/ams/amscams/pipeline/Horizon/Coverage/Coverage.pdf')
    plt.show()

# Plot heatmap of coverage grid on map
def plot_heatmap_cart(grid):
    # generate 2 2d grids for the x & y bounds
    lat_start = grid[-1][0][0]
    lat_end = grid[0][0][0]
    lon_start = grid[0][0][1]
    lon_end = grid[0][-1][1]
    width = len(grid[0])
    height = len(grid)

    x = np.linspace(lon_start, lon_end, width)
    y = np.linspace(lat_start, lat_end, height)
    X, Y = np.meshgrid(x,y)
    Z = np.zeros((height, width))

```

```
    for i in range(height):
        for j in range(width):
            # if grid[-i][j][2] > 2:
            #     grid[-i][j][2] = 2
            # elif grid[-i][j][2] <= 5:
            #     grid[-i][j][2] = 5
            # elif grid[-i][j][2] <= 10:
            #     grid[-i][j][2] = 10
            # elif grid[-i][j][2] <= 15:
            #     grid[-i][j][2] = 15
            # elif grid[-i][j][2] <= 20:
            #     grid[-i][j][2] = 20
            # elif grid[-i][j][2] <= 25:
            #     grid[-i][j][2] = 25
            # elif grid[-i][j][2] > 25:
            #     grid[-i][j][2] = 27
            Z[i,j] = grid[-i][j][2]

    # x and y are bounds, so z should be the value *inside* those bounds.
    # Therefore, remove the last value from the z array.
    Z = Z[:-1, :-1]
    z_min, z_max = 0, np.abs(Z).max()

    proj = ccrs.PlateCarree()
    fig, ax = plt.subplots(1, 1, subplot_kw=dict(projection=proj))

    c = ax.pcolormesh(X, Y, Z, cmap='plasma_r', vmin=z_min, vmax=z_max)
    #ax.set_title('pcolormesh')
    # set the limits of the plot to the limits of the data
    ax.axis([x.min(), x.max(), y.min(), y.max()])
    ax.add_feature(cfeature.BORDERS)
    ax.add_feature(cfeature.COASTLINE)
    divider = make_axes_locatable(ax)
    ax_cb = divider.new_horizontal(size="5%", pad=0.1, axes_class=plt.Axes)

    fig.add_axes(ax_cb)
    plt.colorbar(c, cax=ax_cb)

    ax.yaxis.tick_left()
    ax.set_xticks([-10,0, 10, 20, 30], crs=ccrs.PlateCarree())
    ax.set_yticks([35, 40, 45, 50, 55, 60], crs=ccrs.PlateCarree())
    lon_formatter = LongitudeFormatter(zero_direction_label=True)
    lat_formatter = LatitudeFormatter()
    ax.xaxis.set_major_formatter(lon_formatter)
    ax.yaxis.set_major_formatter(lat_formatter)
```

```

    gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=False,
                      linewidth=1, color='gray', alpha=0.5, linestyle='--')
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.savefig('/home/ams/amscams/pipeline/Horizon/Coverage/Coverage.pdf')
    plt.show()

# Load station list from stations.txt
def get_stations_list():
    station_ids = []
    with open(r'/home/ams/amscams/pipeline/Horizon/processed_stations.txt', 'r') as f:
        for line in f:
            station_ids.append(line.rstrip('\n'))
        f.close()
    return(station_ids)

# Calculate area and volume by integration of sphere volume in polar coordinates
def int_pol_sphere(h, h0, h1, phi, theta1, theta2):
    R = 6371
    r = R + h
    r0 = R + h0
    r1 = R + h1
    phi = math.radians(phi)
    theta1 = math.radians(theta1)
    theta2 = math.radians(theta2)
    dA = r*r * phi * (math.cos(theta1) - math.cos(theta2))
    dV = 1/3 * phi * (
                    r0*r0*r0 * (math.cos(theta2) - math.cos(theta1))
                    + r1*r1*r1 * (math.cos(theta1) - math.cos(theta2)))
    return(dA, dV)

# Calculate area of AllSky7 coverage by integration of sphere volume
def get_coverage_area(grid, steps, h, h0, h1):
    A = 0
    V = 0
    for lat in grid:
        for lon in lat:
            if lon[2] > 1:
                phi = 1/steps
                theta1 = 90 - lon[0] - (1/steps/2)
```

```
                theta2 = 90 - lon[0] + (1/steps/2)
                dA, dV = int_pol_sphere(h, h0, h1, phi, theta1, theta2)
                A = A + dA
                V = V + dV
    return(A, V)

# Calculate the minimum angle for the limiting magnitude and the meteor level H
def get_limiting_angle(alt, lim_mag, H):
    R = 6371
    d_1 = H # meteor level
    h = alt/1000
    alpha_0 = 1.3
    wave_len = 0.51 # lambda in mircrons

    A_ray = 0.1451 * math.exp(-h/7.996)
    A_aer = 0.05 * wave_len**(-alpha_0) * math.exp(-h/1.5)
    A_oz = 0.016
    A_star = A_ray + A_aer + A_oz
    z = 89
    mag_red = 99
    while mag_red > 4:
        X = 1/(math.cos(math.radians(z)) +
            0.025*math.exp(-11*math.cos(math.radians(z))))
        A = X * A_star

        epsilon = 90 - z
        beta = math.degrees(math.acos(R * math.cos(math.radians(epsilon)) \
                            /(R+H))) - epsilon
        d_2 = (R + H) * math.sin(math.radians(beta)) \
                                 /math.cos(math.radians(epsilon))
        mag_dist = 5 * math.log10(d_2/d_1)
        mag_delta = mag_dist + A
        mag_red = lim_mag + mag_delta
        z -= 1
    return(epsilon)


station_ids = get_stations_list()
remove_list = ['AMS1', 'AMS41', 'AMS42', 'AMS48', 'AMS117', 'AMS129', 'AMS153',
               'AMS154', 'AMS157', 'AMS159', 'AMS160', 'AMS20', 'AMS44', 'AMS52',
               'AMS61', 'AMS66', 'AMS7', 'AMS76', 'AMS83', 'AMS9', 'AMS95']
station_ids = [x for x in station_ids if x not in remove_list]
```

```
lat_range = [27, 67]
lon_range = [-20, 37]
steps = 4 # step size = 1/n
H = 120 # meteor level
H_1 = H - 20
H_2 = H + 20
mag_zenith = -3
grid = get_grid(lat_range, lon_range, steps)
file_exists =
   os.path.exists(’/home/ams/amscams/pipeline/Horizon/Coverage/Coverage.txt’)
if file_exists:
    coverage = []
    with
       open(r’/home/ams/amscams/pipeline/Horizon/Coverage/Coverage.txt’,
       ’r’) as f:
        for line in f:
            line_list = [elt.strip("[]") for elt in
               line.split(’,’)]
            lat_list = []
            if ’’ not in line_list:
                for element in line_list:
                    lon_list = []
                    if element != ’\n’:
                        point_list = element.split(’␣’)
                        for value in point_list:
                            lon_list.append(float(value))
                        lat_list.append(lon_list)
                coverage.append(lat_list)
        f.close()
else:
    for station_id in station_ids:
        print(’Processing␣’ + station_id)
        station_lat, station_lon, alt, _, _, _ =
           get_station_information(station_id)
        az_el = []

        with open(r’/home/ams/amscams/pipeline/Horizon/’ +
           station_id
                + ’/HorDat.txt’, ’r’) as f:
            next(f)
            for line in f:
                inner_list = [elt.strip() for elt in line.split(’␣
                   ’)]
                az_el.append(inner_list)
            f.close()

        HorDat = []
        for element in az_el:
```

```
            if element[0] and element[1] != 'nan':
                HorDat.append([float(element[0]), float(element[1])])

        #min_elevation = 0
        min_elevation = get_limiting_angle(alt, mag_zenith, H)
        Distance = get_distance(HorDat, H, min_elevation)
        write_data_to_txt(Distance, 'Distance',
                      ['az', 'dist', 'dist_h', 'dist_g', 'beta'], station_id)

        LonLat = AzEltoLonLat(Distance, station_lon, station_lat, H)
        write_data_to_txt(LonLat, 'LonLat', ['az', 'lon', 'lat'], station_id)

        coverage = get_coverage(LonLat, grid, station_lat, station_lon, H)

    f = open(r'/home/ams/amscams/pipeline/Horizon/Coverage/Coverage.txt', 'w')
    f.write('[')
    for lat in coverage:
        f.write('[')
        for lon in lat:
            f.write('[')
            for element in lon[:-1]:
                f.write(str(element) + '␣')
            f.write(str(lon[-1]) + '],')
        f.write(']\n')
    f.write(']')
    f.close()

plot_heatmap_cart(coverage)
A, V = get_coverage_area(coverage, steps, H, H_1, H_2)
phi = lon_range[1] - lon_range[0]
theta1 = 90 - lat_range[1]
theta2 = 90 - lat_range[0]
A_grid, V_grid = int_pol_sphere(H, H_1, H_2, phi, theta1, theta2)
print('Coverage␣Area:␣' + str(A))
print('Coverage␣Volume:␣' + str(V))
print('Grid␣Area:␣' + str(A_grid))
print('Grid␣Volume:␣' + str(V_grid))
f = open(r'/home/ams/amscams/pipeline/Horizon/Coverage/CoverageArea.txt', 'w')
f.write('CoverageArea␣=␣' + str(A) + '\n')
```

```
f.write('GridArea␣=␣' + str(A_grid) + '\n')
f.write('CoverageVolume␣=␣' + str(V) + '\n')
f.write('GridVolume␣=␣' + str(V_grid)+ '\n')
f.close()
```