

**UNIVERSITATEA DE VEST DIN TIMIȘOARA**

**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**

**PROGRAMUL DE STUDII DE LICENȚĂ:**

**Automatic Detection of Meteors**

**COORDONATOR ȘTIINȚIFIC: ABSOLVENT:**

Marc Frincu Dan Mircea

**TIMIȘOARA**

**2019**

**Abstract**

**Content**

1. Introduction ......................................................................... 4
2. Technologies and Frameworks

2.1 Programming Languages and Frameworks

2.1.1 Python ...........................................................

2.1.2 Java .................................................................

2.1.3 Py4J .................................................................

2.1.4 Flask .................................................................

2.1.5 Astropy .................................................................

2.1.6 OpenCV .................................................................

2.1.7 Numpy .................................................................

* 1. IDEs and other development tools

2.2.1 Pycharm & IntelliJ IDEA ...........................................

2.2.2 AvisFV .................................................................

1. Implementation
   1. Project Diagram
   2. Server Startup
   3. Image Requests
   4. Correction Lines
   5. Meteor Handler
   6. Image Stacking
   7. Star Calculations
   8. Meteor Detection
   9. Image Processing
   10. Meteor Database
   11. REST API
2. Testing
3. Conclusion
4. Bibliography

**Introduction**

What this project tries to achieve is automatic system for identifying and analyzing meteors from a constant stream of images taken constantly by cameras located in certain points around Romania.

Without the use of computer systems, astronomers would have to do a number of complicated operations themselves, which can be a very time consuming and, sometimes, inaccurate process.

With the use of computer systems such as this one, the process of identifying and analyzing certain aspects of the meteor is much more accurate, and while astronomers also have to check if the detections are correct, requires a lot less time than having to do the whole process by hand.

This project was created in partnership with the MOROI (METEORITES ORBITS RECONSTRUCTION BY OPTICAL IMAGING) network project, which is an all-sky network operated by the Astronomical Institute of the Romanian Academy which aims to monitor and track the detections of atmospheric entries of natural or arificial bodies[8].

This project aims to monitor the images taken constantly by the MOROI all-sky cameras and to identify and analyze them for any potential natural bodies (meteors) that they might contain. The project was designed specificaly for the type of cameras that the MOROI network utilizes and was tested with images taken directly by the MOROI cameras.

What this project does is it creates a server that constantly checks images of the night sky taken by cameras with fish-eye lenses, saved on the machine it is being run on and checks said images for any meteors that they might contain. Once found, certain aspects of the meteor are analyzed and all of the information about the event is saved inside of a database.

The greatest challenges posed by this problem were the accuracy of the detection algorithm and the performance. How these problems were overcome will be discussed in depth in the following chapters.

In chapter 2, entitled “Technologies and frameworks”, the different technologies that were used in order to create this project are discussed. We will discuss the programming languages, libraries, frameworks, IDE’s and other tools that went into creating this project. Each of these elements will be described and their role in the project will be detailed. We will also discuss how these technologies work with one another and the decisions for choosing them over other similar technologies, also how implementation them affected the overall performance of the project.

In chapter 3, entitled “Implementation”, a diagram that illustrates how the project operates is shown and all of the different major components are discussed separately. Here we will talk about the actual implementation of the project, how each component that makes this project operates and how they interact with each other. We will also talk about the algorithms used for this project and give an in depth description of how they work. We will also talk briefly about the math that was used for some of the algorithms.

In chapter 4, entitled “Testing”, we will discuss the program’s performance over a set of data. We will talk about the accuracy of the detection and analysis algorithms. We will also talk about the expectations that we had for how the program should behave versus how behaves with a set of data.

In chapter 5, entitled “Conclusions”, we will discuss what the project has achieved this far, its future and how it can be further improved.

**Chapter 2**

**Technologies and frameworks**

* 1. Programming Languages and Frameworks

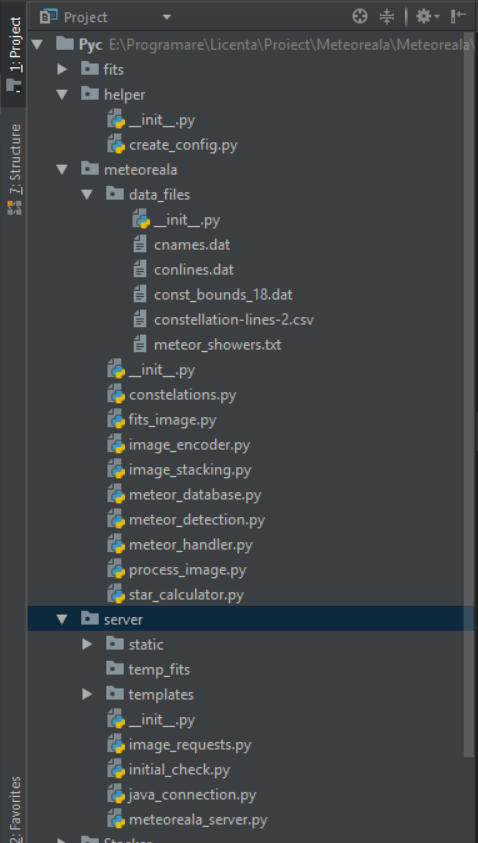
2.1.1 Python

Python is an interpreted programming language that supports functional and object oriented programming. It is a high-level language that focuses heavily on code readability and speed of development. The version that was used in the creation of this project was version 3.6.1.

The majority of the project was developed in Python, either with the base classes distributed with the language or with libraries and frameworks developed by third parties.

The project files were structured according to repository structuring rules, even if some of the files included in the project were not python files. The root of the project contains a configuration file, a setup file and a README file, directories that contain non-python files (such as HTML files, compiled Java classes, images etc.) and python packages, which are directories that contain a special \_\_init\_\_.py file which enables python modules to import modules from other python packages.

This photo shows a part of the collapsed project tree:



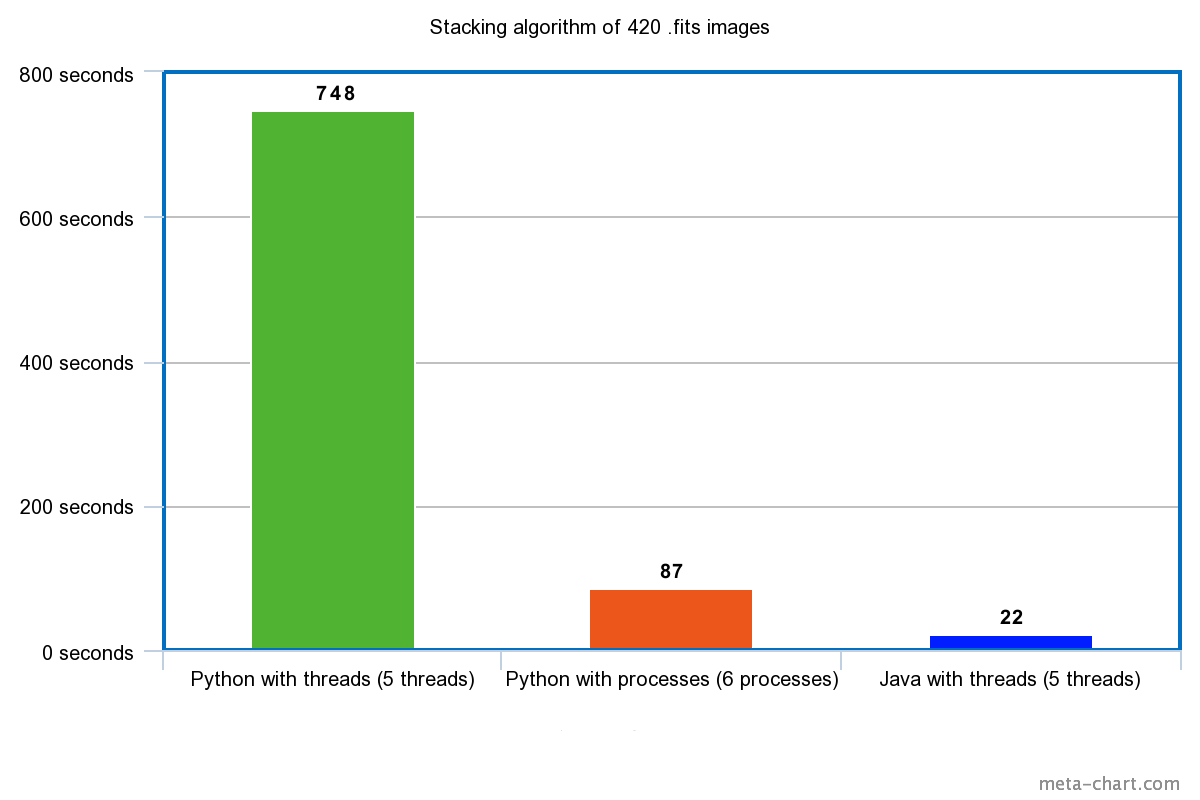
2.1.2 Java

Java is a high-level object oriented programming language that was released in 1995 by Sun Microsystems. The version of java used in this project is Java 9.

Java was used in this project for its parallel programming capabilities. Python lacks in this area because of its GIL (Global Lock Interpreter) which prevents more than one thread to be in a state of execution at once, meaning that performance gained by using multi-threading in python was negligible. While some level of parallelism can be achieved in python using separate processes, these seemed to use to much processor power and RAM, while still not performing as good as the same algorithm written in java.

The algorithm in question was a stacking algorithm on many .fits images. In order to visualize the whole meteor, many frames that captured only a small part of the meteor had to be combined, more on this on the chapter that explain the meteor analysis algorithms.

Below you can see a chart that demonstrates the differences between python multi-threading, python multi-processing and java multi-threading on a set of 420 .fits images:



(The values were chosen based on the average of running the programs 10 times, each had the same set of images)

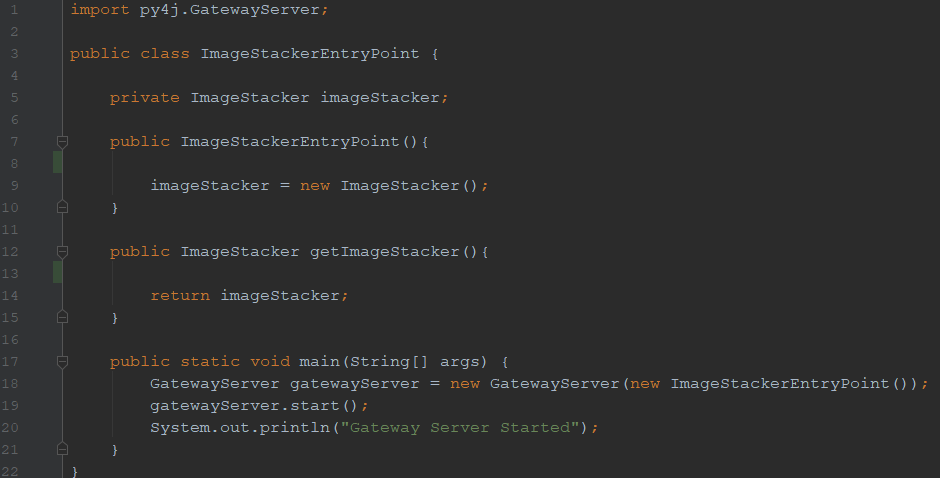
The paths of the images get sent from the Python module to the Java class that performs the stacking, creates a new .fits image and saves it to disk, then sends the path of the resulting image back to the Python module, more on how this is achieved will be discussed in the subchapter about the Py4J framework.

2.1.3 Py4J

Py4J is a framework that enables Python and Java programs to communicate with each other, making it possible for the Python interpreter to access Java objects and Java programs to access Python objects. In this project, the project was used only for accessing a Java object through Python.

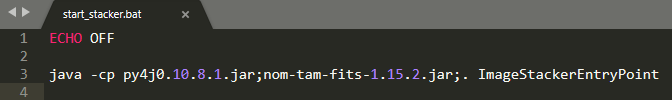
This is achieved by creating a Java object, referred to as a Gateway Server, that allows Python and Java programs to communicate through a local network socket , bound to a specific port, by default this being port 25333.

Below you can see the code for this object:

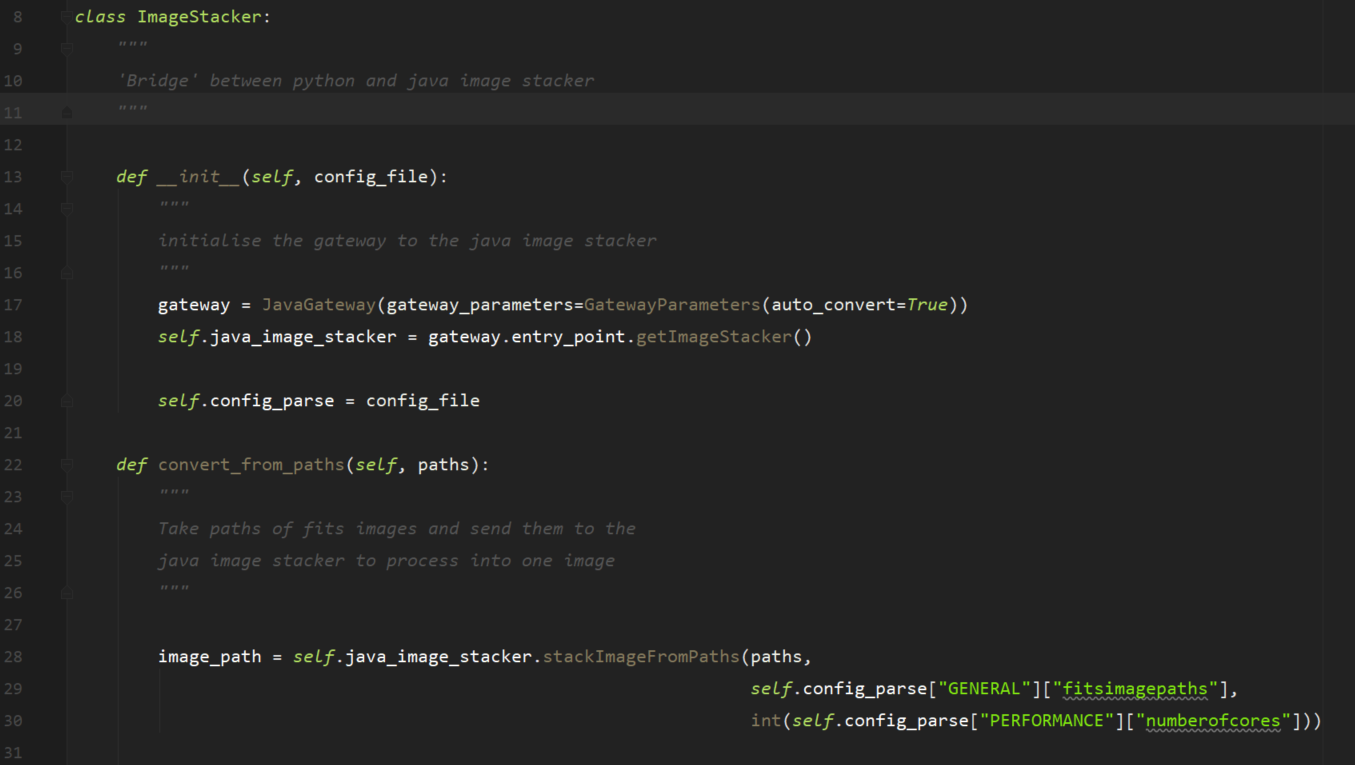


This class will be compiled and the resulting .class file will be called and the start of the program using by a Windows or Linux script, depending on the user’s operating system.

Below you can see the .bat script for Windows:

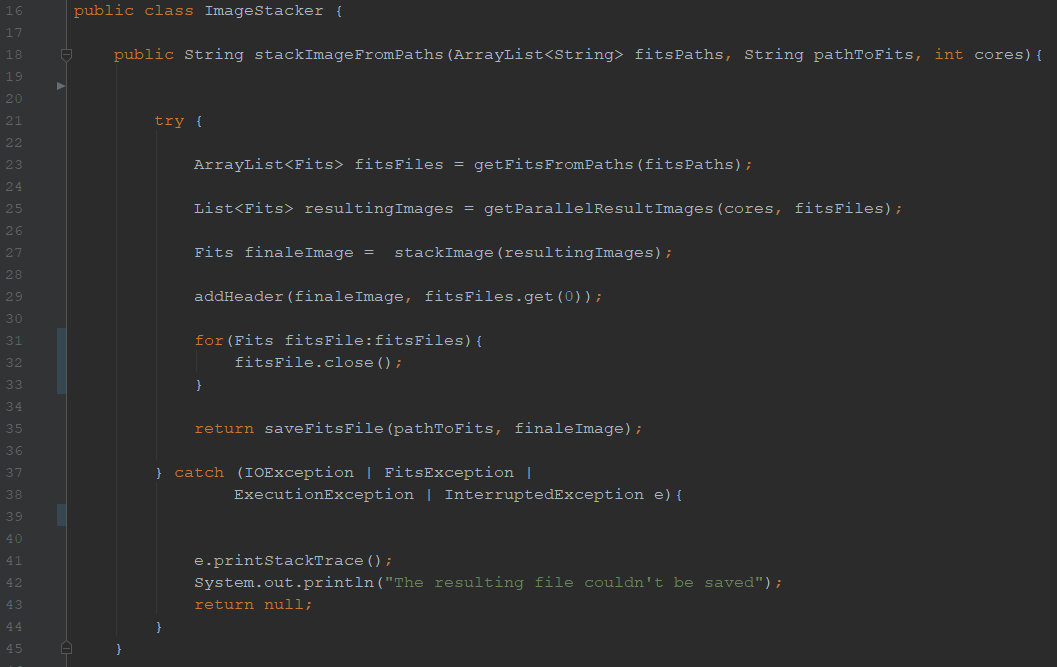


Below you can see the Python code that call the Gateway Server object and calls the function from the Java object that processes the .fits images:



In the \_\_init\_\_ method you can see the code that obtains an instance of the ImageStacker object that contains the method “stackImagesFromPaths” that you can see called in the “convert\_from\_paths” Python method.

Below you can see the Java method “stackImagesFromPaths”:

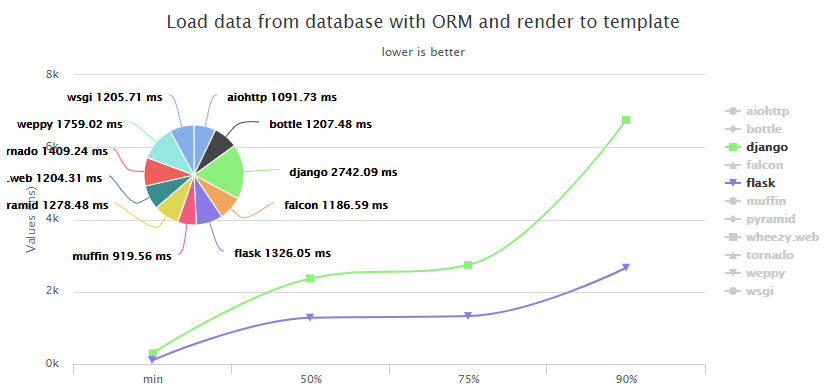


2.1.4 Flask

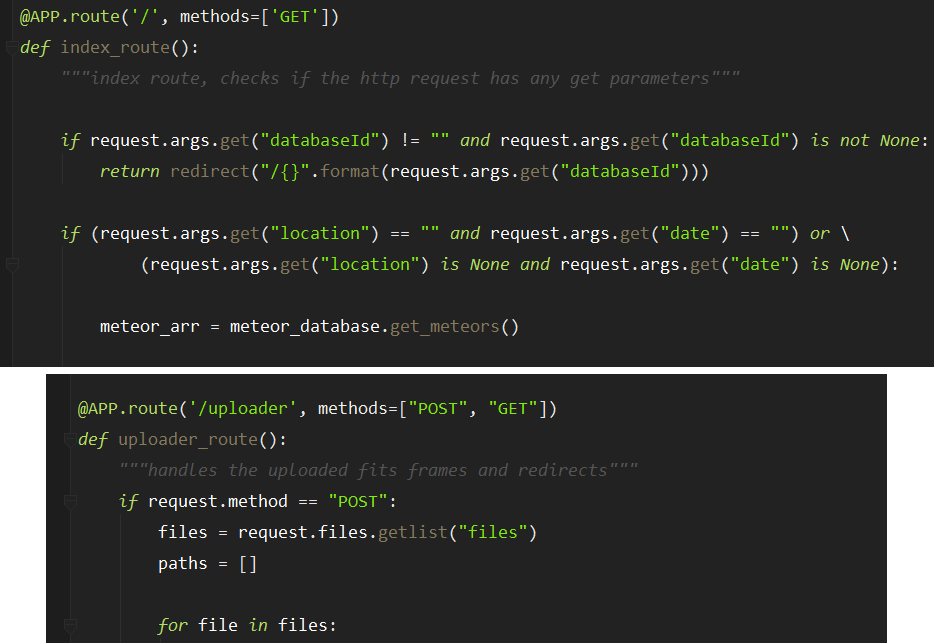
Flask is a popular Python micro framework designed for web development. It is a very light-weight framework because it lacks some features that you can find in other Python web development frameworks, such as form validation, a database abstraction layer, user authentication etc.

The motivation for using Flask in this project instead of a more complex web framework, such as Django, was driven by the speed of development and the increased performance that Flask offers. While Django offers many useful features, most of these would be more suited for large web applications that imply a lot of user interaction. Instead, while Flask may be missing some useful features, it offers greater ease of use, that speeds up the development process and it offers greater performance in terms of response time.

Below you can see a chart that illustrates the difference in performance among many web frameworks, including Flask and Django:



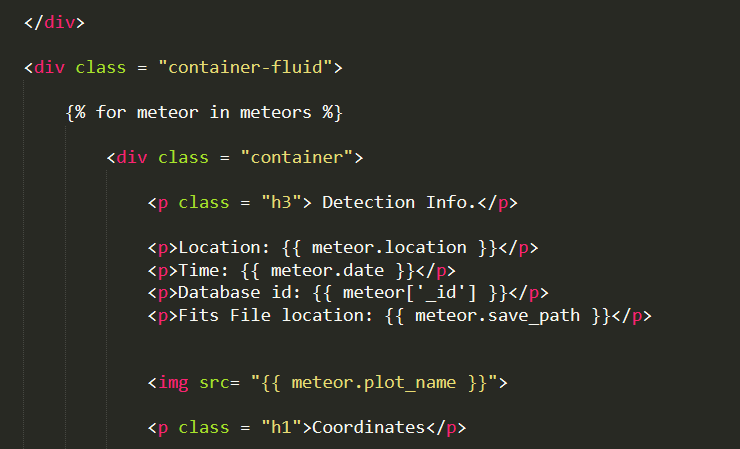
Flask was used to configure the server part of this project. Its main use was to provide routing and the rendering of HTML templates that the user can interact with by using a web browser and accessing the IP address and port that the server runs on.  
  
Below you can see some methods that handle routing:



When a user accesses the server with the route “/” with a HTTP GET request the method “index\_route” will be invoked if the user accesses the route “/uploader” with either a GET or a POST request the method “uploader\_route” will be invoked. Both of these methods do some type of back-end interaction (for example retrieving certain documents from the database) and then send back the user the HTML template for that specific route.

Along Flask, some other Python libraries were used in handling the code for the server, one being Jinja 2. Jinja is used for creating templates that have HTML code intertwined with Python variables and expressions. This is useful when creating web pages that are populated with dynamic elements, such as the documents from a database.

Below you can see a Jinja template that uses a python for loop iterating over the array “meteors”, to display HTML for many database documents:



2.1.5 Astropy

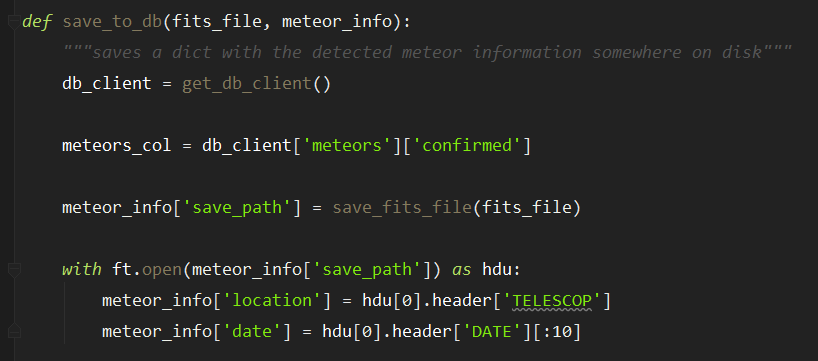
Astropy is a collection of Python software libraries designed for use in astronomy. Astropy has many features, but the one that was used the most in this project is its I/O capabilities for .fits files (built upon the former PyFITS interface).

Fits files are an image format widely used in astronomy. The images consist of two parts, the header and the data. The data represents the actual pixels of the image and is similar to other image formats. The header contains metadata about the image (the resolution of the image, name of the file, the time it was taken) but also other useful information independent of the image itself (the name of the camera that took the image, the location where it was taken, the altitude etc.). The header can hold a lot of information and custom information can be added to it, making this format ideal for usage in astronomy.

Astropy makes it possible to do many types of I/O operations on the .fits images e.g. retrieving the header and the data as a two dimensional Numpy array of pixels, updating them and creating new .fits images.

Being able to access information from the header and certain parts of the images was vital in the detection and analysis of meteors.

Below you can see code for opening and retrieving information from a .fits image header.



2.1.6 OpenCV

OpenCV is a library of programming functions, developed initially by Intel, designed for computer vision, image processing and manipulation. The software is written in C/C++ but it has bindings for Python, Java, MATLAB/OCTAVE and several wrappers for other languages have been developed.

OpenCV’s library for Python was used in this project for identifying meteors in the images and analyzing certain aspects. The algorithms for detection analysis will be detailed in the later chapters.

2.1.7 Numpy

Numpy is a Python library designed for interacting with large scale multi-dimensional arrays and matrices. The library casts normal Python lists to a Numpy specific type called numpy arrays. The library provides special methods for accessing and interacting with the numpy arrays and array elements.

The pixel data of the fits images is converted into a numpy array of 16 bit integers. Since the images are grey-scale, a pixel is represented as a single value.

The fits images used in this project have a resolution of 1296 by 966 pixels, resulting in numpy arrays of over 1.000.000 integers. Numpy arrays are ideal for this situation, since normal Python lists would be significantly more inefficient.

Numpy arrays also provide many useful built-in functions for interacting with the pixels of the images, such as selecting all the pixel locations with a certain value.

* 1. IDEs and other development tools

2.2.1 PyCharm & IntelliJ IDEA

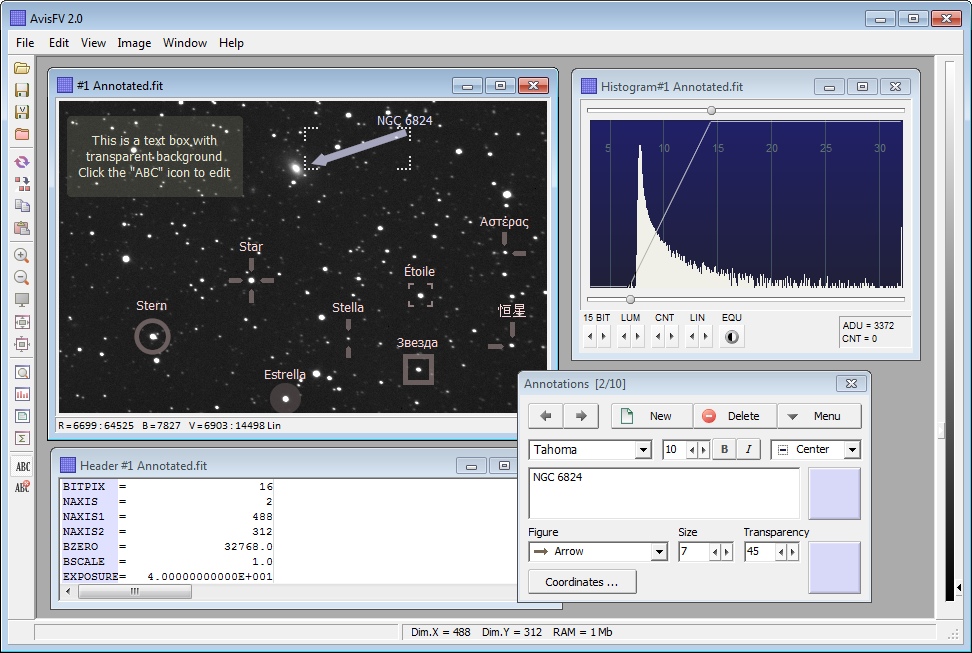
PyCharm and Intellij IDEA are two integrated development environments (IDEs) developed by JetBrains. They were used for writing the Python and Java code.

The reasoning for choosing these IDEs over text editors, such as Sublime Text 3 or Vim, is because of the many extra features that these IDEs offer, such as integrated refactoring shortcuts, faster code completion, version control assistance tools.

2.2.2 AvisFV

AvisFV is a fits image viewer; it makes it possible to visualize fits images and to access all of their header’s information.

It was used in order to verify the detections used by the detection algorithm and also to check if the stacking algorithm works correctly, by visualizing images and comparing them to other stacked images.

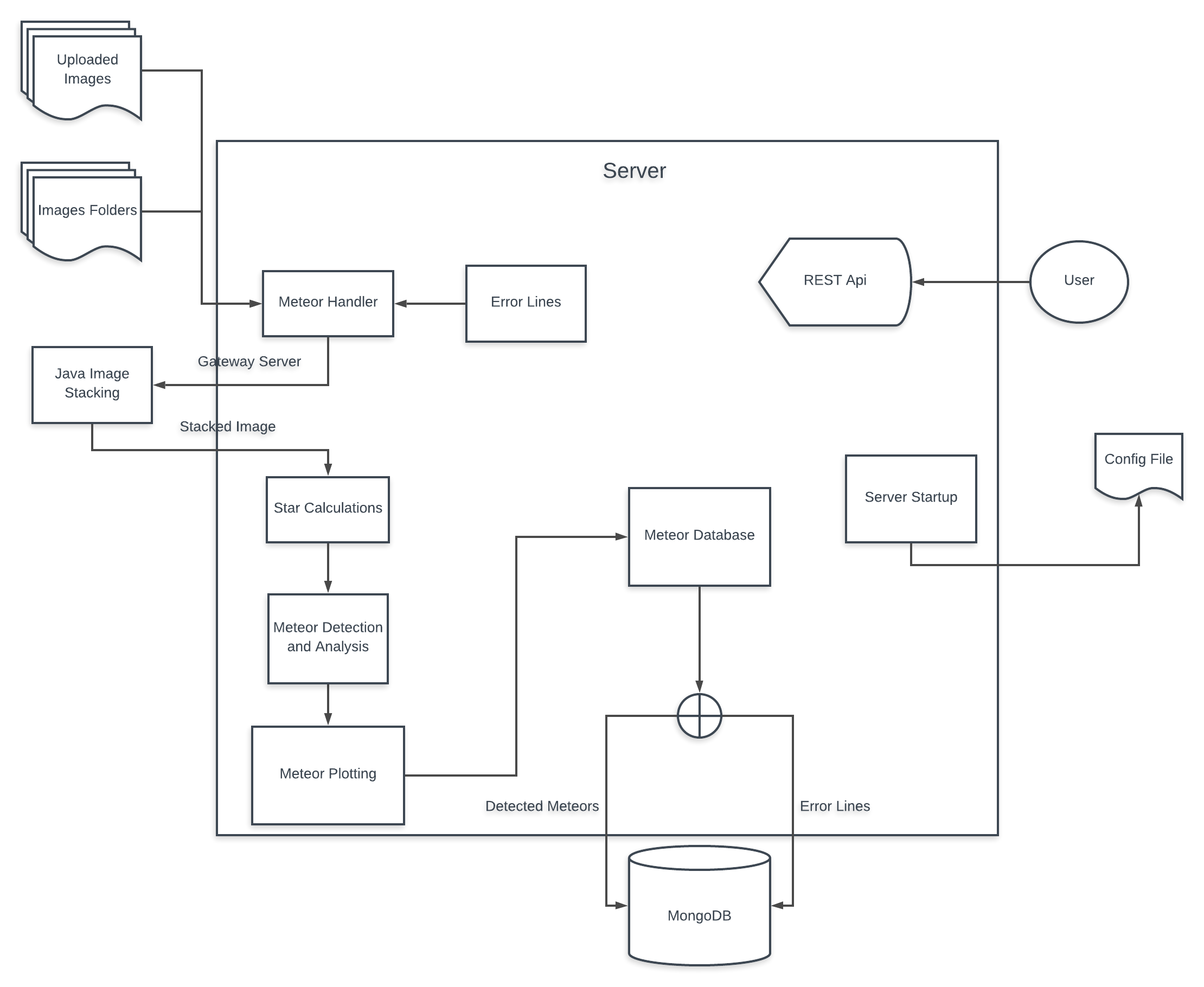


**Chapter 3**

**Implementation**

3.1 Project Diagram

This diagram represents all the important components of the project, and their work flow:



3.2 Server Startup

This component handles what happens when the program is initially started.

Firstly, some initial checks are being made to unsure the program can run correctly:

* The program checks if the configuration file exists
* It checks to see if all the values in it are completed and not empty
* All the values in the in configuration file that are paths to folders are checked to see if they exist, and if they don’t then they are created

In all of these cases the program will either try to solve the problem (creating folders that should exist but don’t) or close the program and signal the error to the user.

The image below represents the portion of code for checking the integrity of the configuration file:

After these checks are completed then the program initializes some components of the project.

The first one to be initialized is the Java Gateway sever, that enables the communication of the program with a Java class. The program checks what system it is being run on, either Windows, MacOS or a Linux distribution, then the program checks if a process is currently running on the port that the java server is should be running (the default port 25333) and if it finds a process currently running on that port it kill. After that it executes the script file corresponding to the operating system that will initialize the Java Gateway server.

Next, the program starts the threads for the error correction lines and the image requests. These parts will be discussed more in depth in their respective sub-chapters.

After these steps, the flask server bounds to a specific IP and port and can be interacted with.

3.3 Image Requests

This component delegates fits images to the algorithm that analyzes them. It has three sub-components:

* Images that were directly uploaded to the server through the REST API
* Images from internal storage files that get continuously checked
* Single images from internal storage files used to create correction lines

All of these images go through the meteor analysis algorithm, yet the final results for the correction lines images are stored in a separate collection in the database, the motivation for this will be discussed later.

The absolute paths to all the images are gathered from the folders they are stored in (the uploaded images also get saved to the server temporarily) and separate threads for analyzing them get started, for the images from internal storage one thread gets started for each set of images from different cameras.

3.4 Correction Lines

“Correction lines” is term I used to refer to noise and false positives that the detection algorithm might pick-up and confuse with meteors inside of images. Because of certain complications from the placement of the cameras and some other natural phenomena, the detection algorithm might confuse parts of the image with meteors.

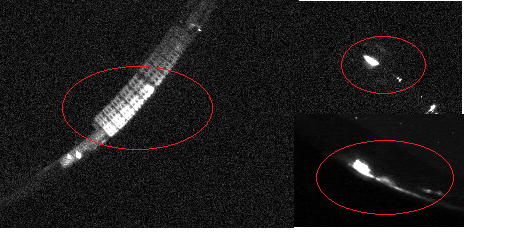
Mostly the complications come from the cameras being placed too close to light sources that interfere with the quality of the images, like street lights from roads and from cities.

The image below illustrates this case:



This image was taken from the camera in Barlad. The light sources from this image might will be interpreted as meteors, even though they come from street lights nearby.

The image below was formed from locations in images that the detection algorithm interprets as meteors.



This results in detection of false positives that get in the way of analyzing meteors correctly and to the overall performance of the detection.

To circumvent this, the program detects “correction lines” which are the parts of an image that the program falsely interprets as being meteors. The way this works is that the program will start a thread at a certain time of the day (the time is configurable) that selects a single image from each camera location and will purposefully detect the incorrect lines that come from noise and store them in a separate collection in the database. Then every detection made in the images that the program analyses will be checked to differ from the correction lines, this way the only events detected by the algorithm will be the correct ones.

This works because a single image cannot contain a fully visible meteor, since the meteors are only visible through stacking of many images together, so if any type of event is detected in a single image than it cannot be a meteor.

The flow for creating correction lines is similar to the regular way of detecting the meteors, with a few key differences:

* When creating the correction lines, only a single image from every camera is used, as opposed to many images from every camera, therefore the stacking process is skipped here.
* The images are only checked for detected lines, they do not get analyzed further.
* Astronomical objects such as stars and constellations do not get calculated or plotted for these particular images.
* The resulted detections are stored in a separate collection in the database from the detected meteors.

3.5 Meteor Handler

The meteor handler component serves as a “bridge” between the server and the meteor detection algorithm; it receives the images and calls upon the correct methods to further handle these images.

The images that will be analyzed for potential meteors get sent to the stacking algorithm and the resulting single image is converted to a python object from a custom class “FitsImage” that contains the image data and the header, along with some custom methods for the object, such as transforming the image data from 16 bit integers to 8 bit integers, updating the image data etc.

The images for the correction lines do not get stacked, they all get transformed separately into the custom class “FitsImage” and are added to an array “FitsImage” objects.

After this, the resulting images get sent to the analysis part of the algorithm.

3.6 Image Stacking

The image stacking component serves to combine many fits images into a single image, it is a widely used technique in astronomy. This is needed because in a single fits image only a small part of a meteor is visible, making it impossible to detect or analyze them this way.

The images below illustrates why image stacking is needed:



The circled areas in these images are meteors as seen in single images, they cannot be analyzed correctly.



The image above represents the meteor after 63 single images were stacked together, including the ones previously shown. In this form much information can be extracted from the resulting image, such as the direction of the meteor, it’s length etc. Also detecting a meteor in this form is much easier and offers greater accuracy.

The stacking algorithm gathers all of the fits that need to be stacked into a single image and divides equally among a configurable number of threads and stacks them into single pictures. Then all of the resulting images get go through the stacking process once again to create a final single image.

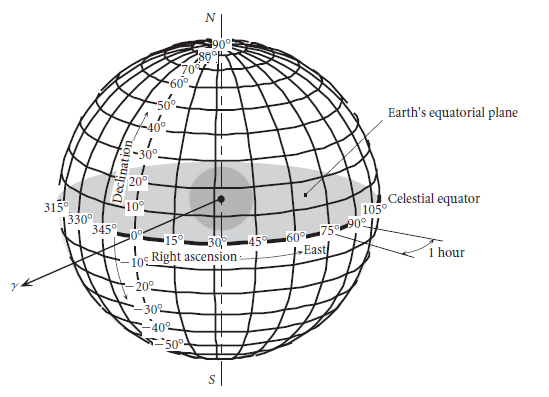
The actual stacking process of images is quite simple. The algorithm compares all the pixel values from two different images and creates a new matrix with all of the maximum pixel values. This way all of the light sources from the images get overlapped, creating the effect that can be seen in the image above. After the stacking of the image data is done, a new fits image is created with it and it receives an image header from one of the single image frames.

3.7 Star Calculations

This component deals with the calculations involved with converting celestial objects from their real world coordinates to the type of the projection the images use and vice versa.

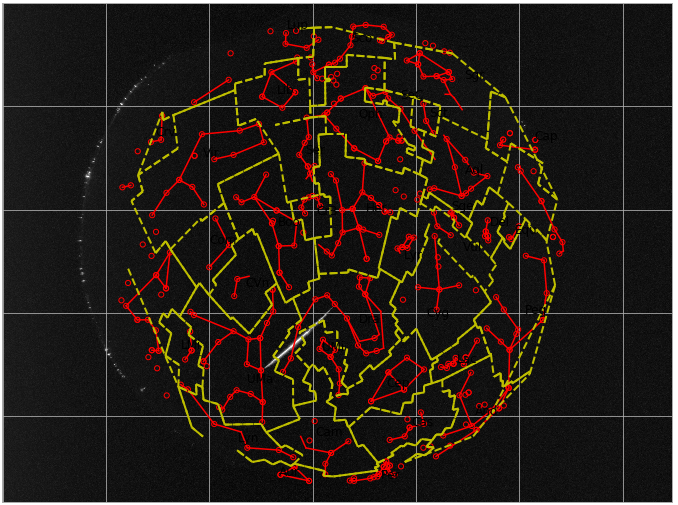
In astronomy the celestial coordinate system is a system for specifying positions of celestial objects[1]. There are many celestial coordinate systems, the one the celestial objects in this project used was the Equatorial coordinate system. This system represents a celestial object’s coordinates as a pair, the declination and the right ascension.

The image below illustrates the declination and right ascension system.



In order to visualize the images better, some celestial objects were converted from their Equatorial coordinates to the images projection in order to represent the objects as seen on the sky at the location cameras and at the time the images were taken.

The image below is a regular image from one of the cameras that had the celestial objects represented on it.

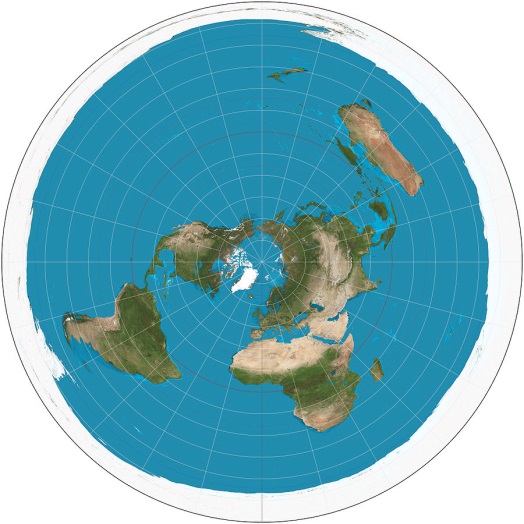


The celestial objects that can be seen here are:

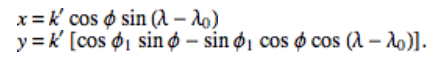
* Stars, represented by the red circles
* Constellations, represented by the red lines
* Constellation Bounds, represented by the yellow lines
* Constellation Names

The images use a type of fish-eye distortion effect called Azimuthal equidistant projection in order to better see the entirety of the night sky captured by the cameras.

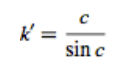
The image below shows a map of Earth represented in an Azimuthal equidistant projection:



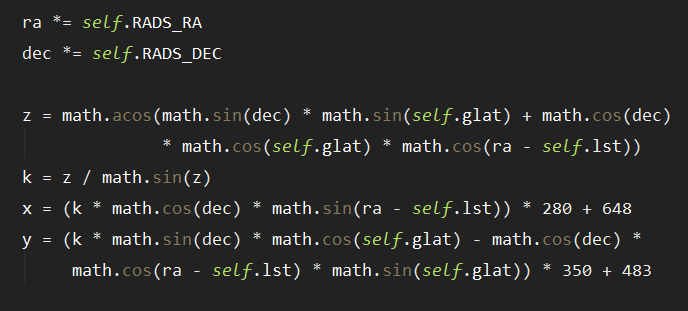
In order to be able to correctly map the celestial objects to the pictures taken by the cameras the following formulas were used where phi_1 and lambda_0 represent the declination and right ascension of the celestial object:



In the formulas above x and y represent the resulting pixel values of the celestial objects.



Below you can see the Python implementation of these formulas:

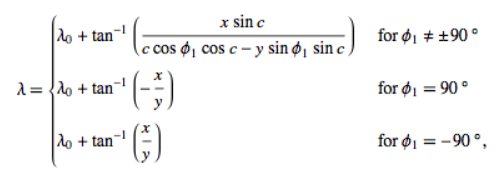


In practice the x and y coordinates were adjusted to better fit the fits images, hence the why x variable was multiplied by 280 + 360 and the y variable was multiplied by 350 + 483.

This segment of code was used to calculate all of the celestial objects pixel values.

The other major part of the star calculations component is the applying the inverse formula to the one used above to calculate a meteor celestial coordinates from its pixel values.

We can calculate the declination and right ascension of a celestial object from its pixel coordinates in the image using the following formulas:



Where the angular distance from the center, represented by c given by:

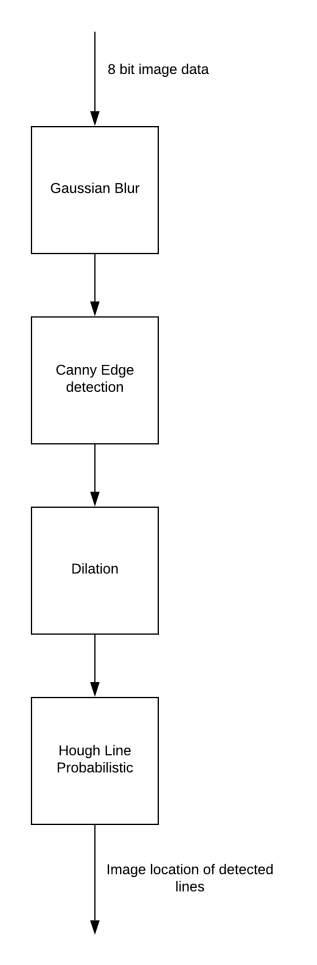


3.8 Meteor Detection

This component handles the meteor analysis and detection part of the project. Here the images are checked for any events and if any are found then they are analyzed. Here, Python’s OpenCV library is heavily used.

**Meteor detection**

The process for the detection of meteors is illustrated in the image below:



Firstly, the image data is converted from 16 bit integers into 8 bit integers in order for it to be properly processed by OpenCV’s elements.

The first filter applied to the image a Gaussian Blur, an image smoothing technique[3]. Gaussian Blur uses a gaussian kernel with a modifiable width and height. It is very useful in removing gaussian noise from the image.

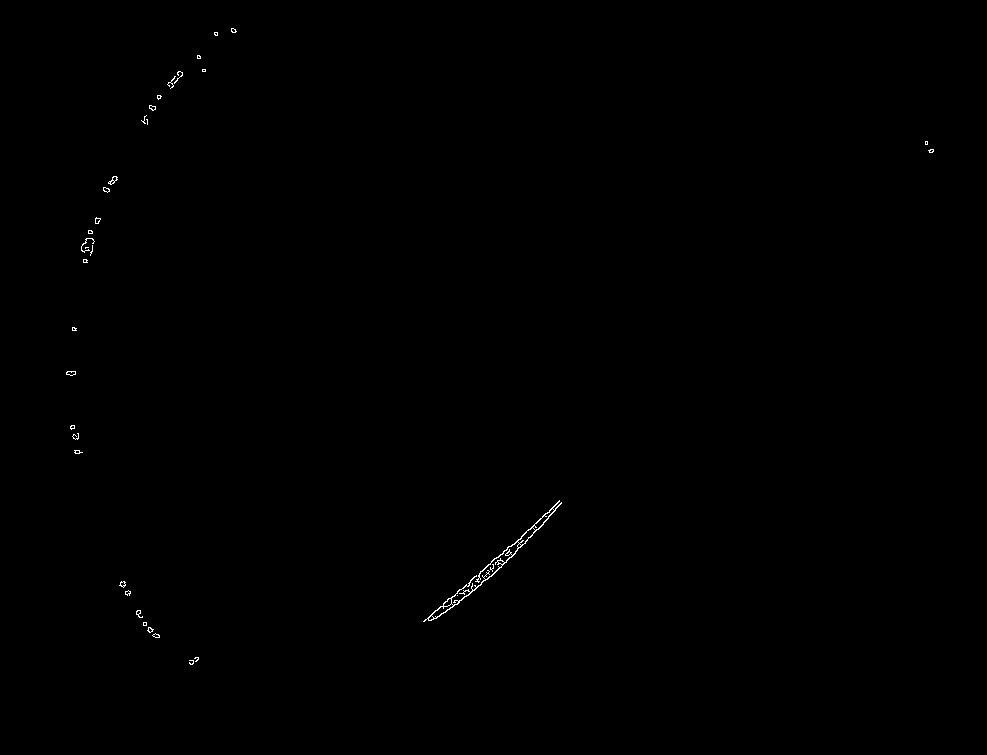
Gaussian noise is statistical noise having the probability density function equal to the normal distribution (Gaussian distribution).

The image below shows the image data before and after the Gaussian Blur filter is applied to it:



In the image with the Gaussian Blur applied to it, the edges of the light sources were smoothed out or removed.

The next step is applying the Canny Edge detection algorithm to the resulting image data in order to further remove noise and some of the unnecessary information from the image.

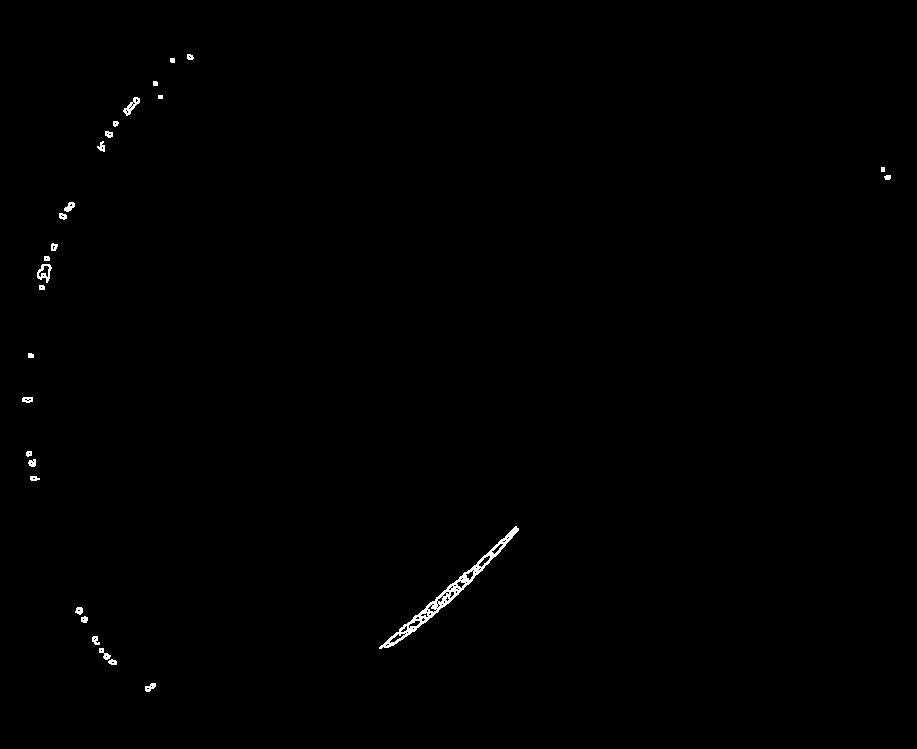


As you can see, all of the pixels with lower values have been reduced to pure black pixels, leaving only the more prominent light sources. This will make it easier for the line detection to identify straight lines in the meteor since the edge detection algorithm tries to identify the outlines of objects.

What we get from this is a binary image of black and white pixels.

The next step algorithm applied to the image is an OpenCV function called dilation[4]. Dilation is an image manipulation technique where a kernel of fixed size is moved along the image pixel grid and checks if any of the pixels inside kernel have the value 1 and if at least one pixel satisfies this condition the all pixels inside the kernel will be updated with the value 1.

This will result in the white region of the image expanding. The following image demonstrates this:

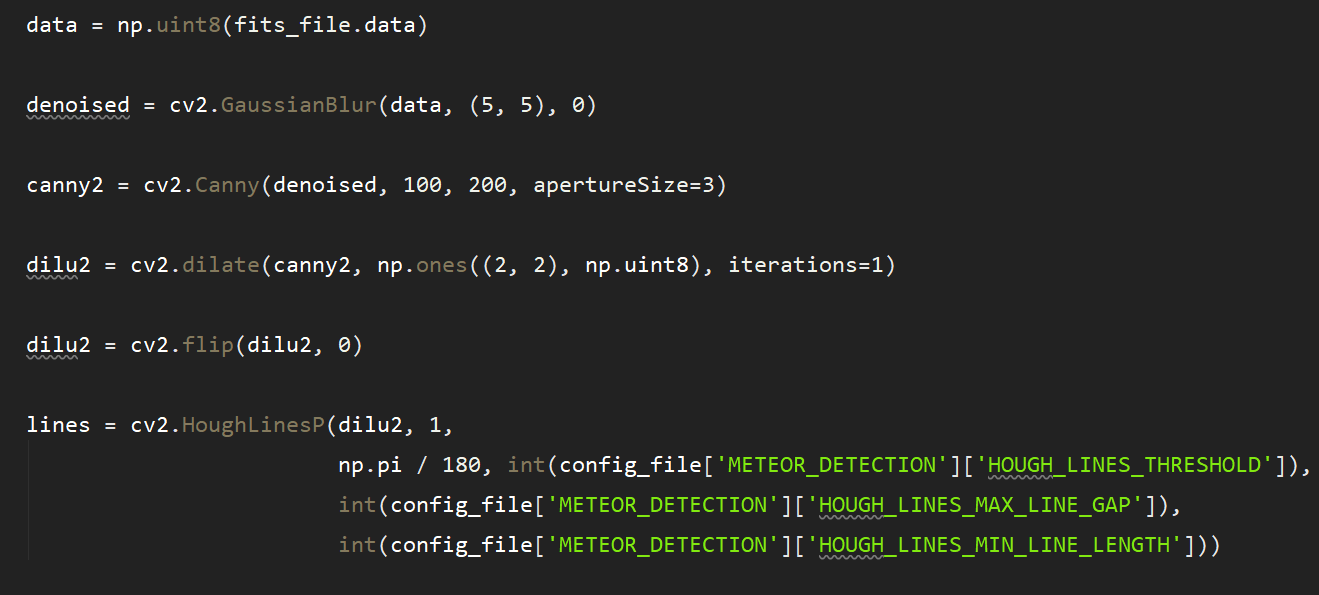


Now the edges detected by the Canny algorithm have expanded. This will result in the straight line detection algorithm detecting more straight lines than before. The drawback to using this is that the algorithm may become more inaccurate, detecting other light sources in the image than the meteor itself, however we can afford to do so because we use the correction lines explained in earlier sub-chapter.

The final step is applying the Hough Line algorithm. Hough Line is a straight line detection algorithm for images. Considering the meteors take appear as straight, with a very slight curvature, the algorithm is ideal for the problem at hand. The program used the Hough Lines Probabilistic version of the algorithm, a more efficient implementation of the original Hough Lines algorithm, in OpenCV it is implemented with the function HoughLinesP().

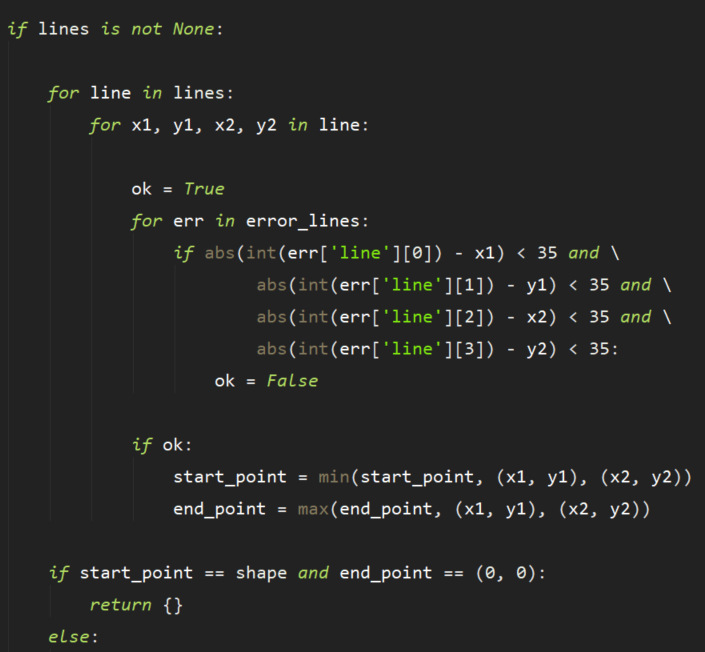
From this we will get the position of the straight lines detected in the algorithm. The HoughLinesP() function returns arrays of 4 points that denote the lines (x1,y1, x2,y2), which represent the location of those points in the matrix of pixels found in the image.

Below you can see the Python implementation of the meteor detection algorithm:



After the detection, the lines are checked to see if all of their points are within a 35 pixel radius of any of the correction lines and if they are not then the program saves the two lines that are placed at the most extreme points from others on both axis, these lines should represent the start point and end point of the meteor.

Below you can see the Python code for this:



After both ends of the meteor have been found, they are converted into their celestial coordinates using the Star Calculations component.

Having their celestial coordinates (declination and right ascension) we can start to analyze the meteor.

**Meteor Analysis**

Once the meteor is identified we can analyze it. The aspects of the meteor that the program analyses are:

* Angular Distance
* Magnitude
* Possible Origin Meteor Shower

**Angular Distance**

The angular distance represents is the [angle](https://en.wikipedia.org/wiki/Angle" \o "Angle) of [length](https://en.wikipedia.org/wiki/Length" \o "Length) between the two directions originating from the observer and pointing toward these two objects. The observer in this case representing the camera taking the picture and the two objects representing the both ends of the meteor.

Because we have the celestial coordinates of the objects, we can use the calculate the angular distance of the meteor’s ends using the following formula:

c = cos-1(sin(d0)\*sin(d1) + cos(d0)\*cos(d1)\*cos(r0-r1))

Where c represents the angular distance, d0 and d1 represent the declination of the start and end of the meteor and r0 and r1 represent the right ascension of the start and end of the meteor.

**Magnitude**

In astronomy, the magnitude is an unitless measure of the brightness of an object.

In order to calculate a meteor’s magnitude the following steps need to be taken:

1. Choose the region of the image that contains the meteor
2. Get all of the pixels within a radius of 3-4 pixels of said region, denote the total number of pixels with N\*, then calculate the sum of pixel values of that region, denote the sum with S
3. Choose all of the pixels within a radius of 6-9 pixels, excluding the area that forms N\*, then calculate the average of the pixel values, denote it with B
4. The magnitude of the meteor is given by the formula S’ = S – N\* \* B[5]

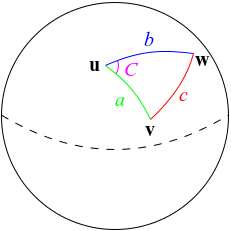
In order to get maximum accuracy when calculating the meteorg magnitude, we would also calculate the magnitude of a visible star using the same process and then compare the magnitudes. This step was skipped since some of the stars that should be visible in the images are not because of camera settings.

**Possible Origin Meteor Shower**

In order to calculate the meteor showers that the meteor might originate from we need the celestial coordinates of all the known origin showers.

After a list was compiled[6] with all of the known meteor showers, then the program checks if any of the meteor showers is the possible origin of the meteor. In order to do this we use the spherical law of cosines formula[7].

First we need to check the period in which the meteor shower is active, if the meteor was detected in that time period then the we use the spherical law of cosines, picking the meteor shower as one point, and both ends of the meteor as two separate points.



Then we calculate the angular distance from each point to another, denote the angular distance from one point of the meteor to another with a, and from each end of the meteor to the meteor shower with b and c. Then we use the formula:



From this formula we get C, which is the angle formed, if this angle is smaller than a configurable number in degrees (by default 5) then then meteor shower might be the origin of the meteor. Since we do not know which end of the meteor is the start point and which is the end point, we need to calculate this again for the angle formed by the other end of the meteor.

If none of the meteor showers are found to be possible origin sources for the meteor then the meteor is classified as sporadic.

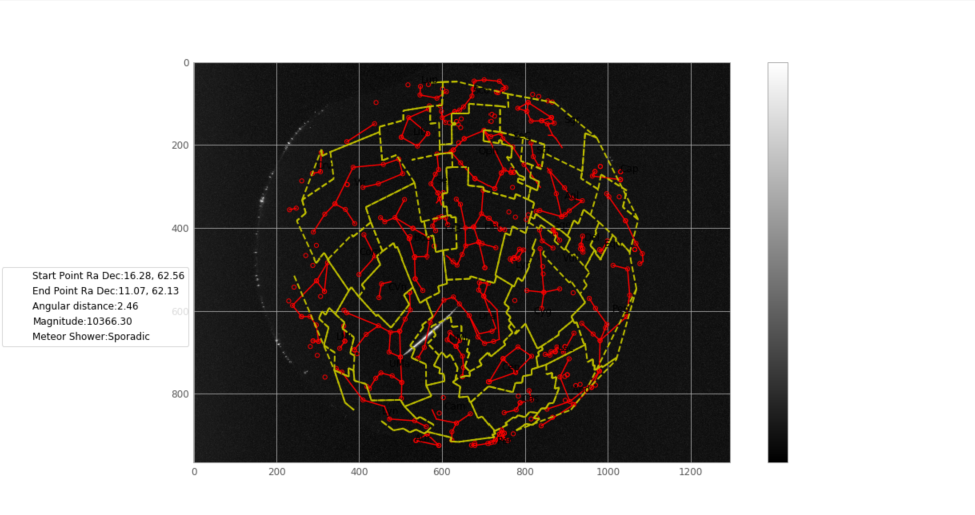
After all of the these aspects are identified and analyzed they are compiled inside of a Python dictionary along with their real world coordinates, pixel coordinates and location of the saved fits file that was analyzed.

3.9 Image Processing

This component deals with mapping the celestial objects that were converted from their celestial coordinates to their pixel coordinates to the fits images along with a legend containing the analyzed meteor elements.

The images that result are saved in a folder on the server so they can be displayed in the REST API.

Below you can see one of these images:



The pixel values are displayed on a graph using the Python plotting library Matplotlib with a greyscale representation. After this the legend is added and the converted celestial objects get plotted in their respective positions.

3.10 Meteor Database

This component deals mostly with interacting with the MongoDB database, with the only exception being it also saves the resulting fits files from the final analyzed images. The database is external to the program, having to be setup separately by the user. Its IP address and port have to be added in the config file, where the program will check in order to make the connection to it.

The only operations that the program makes to the database are adding and searching. The only elements added are Python dictionaries in two separate collections. The searching is done either for all elements of a collection or for a specific subset of a collection through the use of search filters, such as mongo id, meteor date and location.

The two collections that the database holds are for the identified meteors and for the correction lines.

The collection named “confirmed” contains information about the detected meteors while the one named “error\_lines” contains individual lines detected as correction lines from images, in the form of arrays with 4 points.

These images show the resulting collections as seen with a visualization tool:

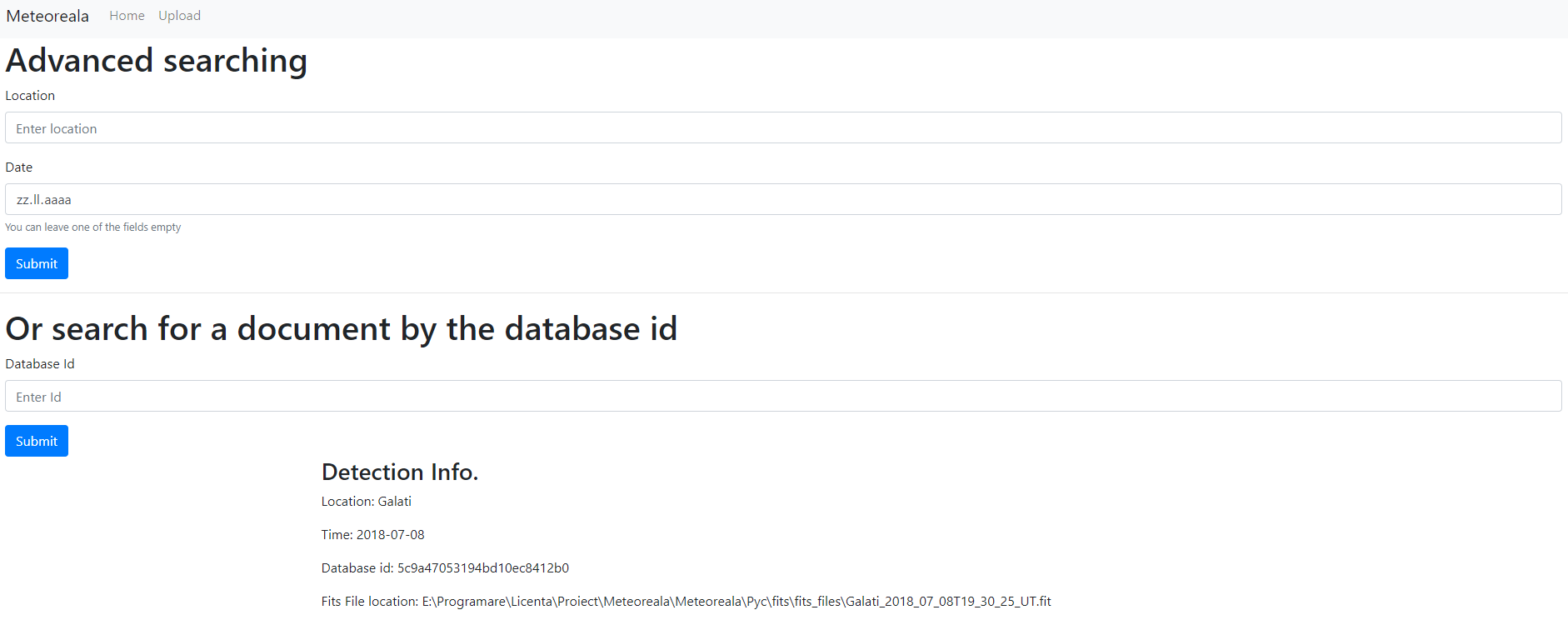


3.11 REST API

The REST API’s allows the user to interact with the program. Its main purpose is to visualize all of the meteor data, but the user can also upload to the server specific fits images in order for them.

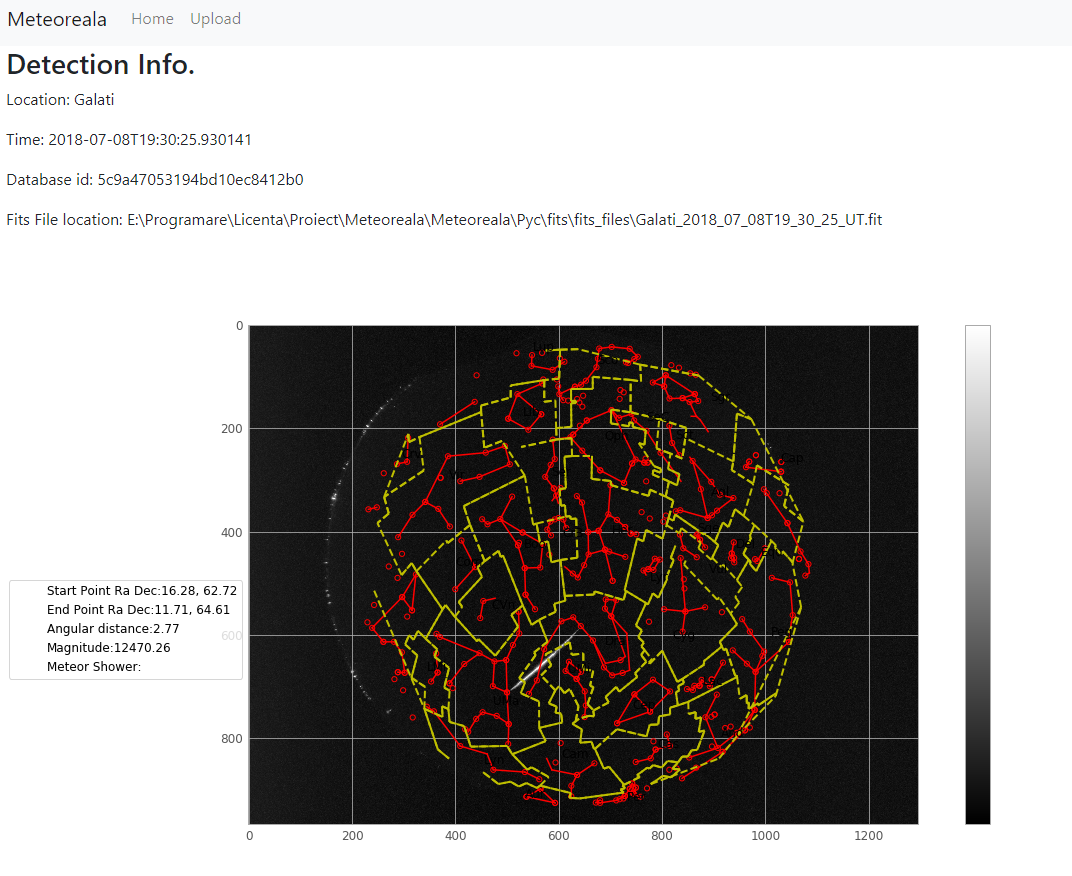
The REST API features many routes created with the Python Flask micro-framework. If certain routes are accessed by the user the server displays specific templates, which are HTML code intertwined with Python code in order for them to be able to hold dynamic data. This is realized by using the Python framework Jinja 2.

Below you can see the template for the server’s index route as rendered in browser:



Here the user can see all of the detected events, or search for specific ones, either one through its id, location or date.

If the user chooses to search for certain events, then the server will point him to a specific route. For example if the user chooses to search for one particular event through its id the server will redirect him to a route that renders only that particular event, like in the image below:



This is done by the server redirecting the user to a route that contains the index route along with concatenating the database id of the event, like so:

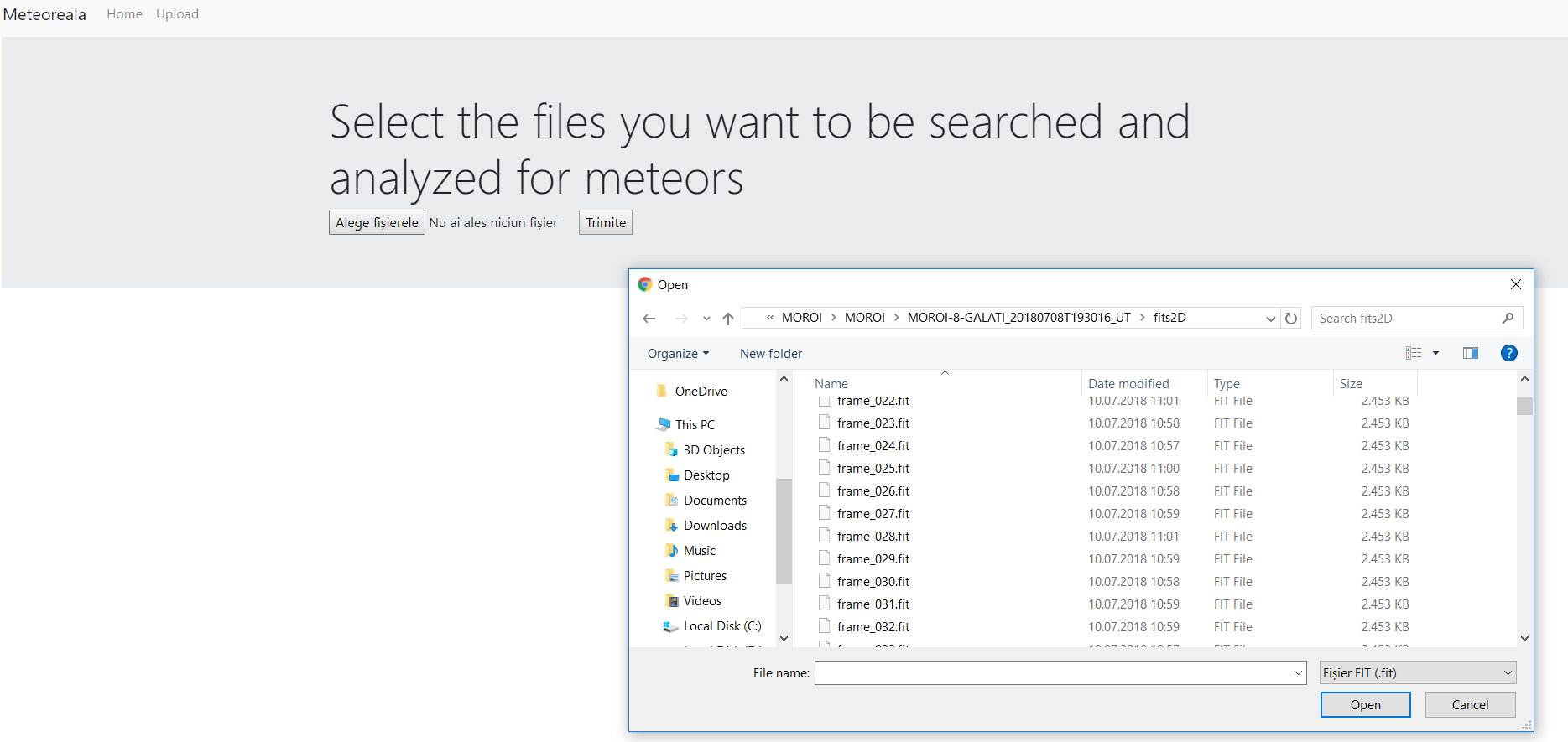


The search for specific meteor characteristic, is done in similar fashion, however the server redirects to the index route only with HTTP GET request added to it, like so:



The other feature of the REST API is the ability to upload specific fits files to the server in order for them to be analyzed just like the images from the folders on the machine.

If the user accesses the route “/upload” then the following template will be rendered:



The fits files the user selects will go trough the same detection algorithm and then will be saved to the database.

Below you can see the Python code for handling a request to the index route using Flask:



**Chapter 4**

**Testing**

In order to test the accuracy of the detection algorithm we will use 3 separate sets of 60 images that will be analyzed, two that contain meteors and one that does not contain any meteors and has significantly more noise. We will look at every step needed for the meteor detection, from the stacking process to the actual detection and analysis.

**Chapter 5**

**Conclusions**

The project achieved its initial purpose, in creating a monitoring system that automatically detects and analyzes meteors from the MOROI network images, however development of the project will continue.

Further the program will be converted to work with many more types of images in order for it to be more generally useful. This would require a wider set of data from many types of camera projections. The detection algorithm is already suited for the detection of meteors from different types of images, however the analysis algorithm would need further development since, in its current form, it relies on the type of camera distortion effect caused by the MOROI cameras.

There are many other future improvements to be made to the program.

Firstly, the detection algorithm can be further improved through the use OpenCV’s filters and algorithms. The main feature that would help improve the accuracy of the detection algorithm would be the addition of further filters that would help with “denoising” the image. In its current state, nearby light sources also get picked up by the edge detection algorithm, which might end up interfering with the detection of meteors. Eliminating them fully would also mean that there would be no use for the correction lines component, which would help to simplify the project.

Secondly, the performance of the project can be improved by converting the image stacking algorithm from Java to Python. Removing the gateway server for the communication between Python and Java would mean the removal of potential problems with the communication. In its earlier stages, the stacking algorithm was done in Python, however it showed really poor performance. However, through the use of Numpy’s matrix manipulation function the stacking algorithm could be implemented in Python and could offer accuracy comparable to the accuracy of the current stacking algorithm.

Thirdly, many features can be added to the REST API in order to make it more user-friendly. Many more search filers could be added and the ones that are currently availed could be further improved, such as selecting all of the detected meteors before or after a certain date.

Fourth improvement, many other attributes could be added to the analysis algorithm. In its current form, the only aspects currently analyzed are the meteor’s celestial coordinates, angular distance, magnitude and origin meteor shower. One very useful feature that can be added would be the triangulation of a detected meteor between many cameras that are in close proximity to one another. This way the height to which the meteor burned out could be detected and it’s trajectory towards earth. Knowing this, astronomers can see which meteors burned out in the atmosphere and which might have landed on the ground.

**Bibliography**

1. <https://en.wikipedia.org/wiki/Celestial_coordinate_system>
2. <https://en.wikipedia.org/wiki/Equatorial_coordinate_system#Spherical_coordinates>
3. <https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>
4. <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html>
5. <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180003501.pdf>
6. <https://en.wikipedia.org/wiki/List_of_meteor_showers>
7. <https://en.wikipedia.org/wiki/Spherical_law_of_cosines>
8. <http://www.astro.ro/~roaj/28_1/09-nedelcu_2018.pdf>