universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Vienna Sky: database and a visualisation framework for the telescope infrastructure of the University of Vienna“

verfasst von / submitted by

Erik Dominik Brändli, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 861
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Astronomie
Betreut von / Supervisor:	Dr. Stefan Meingast, Bakk. MSc

Abstract

Context The Vienna Institute for Astrophysics (IfA) operates two telescopes in Austria, the 1.5m Leopold Figl Observatory (LFO) and the 80cm Vienna Little Telescope (vlt) within the Northern Dome of the Vienna Observatory. This work addresses the critical need for a centralized data repository by proposing and constructing a modern data archive.

Aims The development of an automated archive for two telescope facilities is planned. Historical and recent data should be transferred to a centralized database. The data stored in the database should be available to the end user through a web service that allows them to select data either through text-defined queries or through a browsable sky map.

Methods PostgreSQL is employed for database management, which stores the necessary metadata. A pipeline powered by Python is used to populate the database and initiate plate solving if required. Plate solving is accomplished with the nova.astrometry.net suite. Additionally, a webservice is set up with the Django framework, providing an Aladin lite frame to explore the sky and view images, as well as a textual query interface using SQL for more complex queries.

Results Docker Compose was employed to reduce the burden of hosting these services, resulting in an automated, low-maintenance archive that provides the Institute for Astrophysics with an appropriate infrastructure.

Conclusion This master's thesis addressed the urgent requirement for a modern data archive at the Vienna Institute for Astrophysics. By constructing a new archive with automated cataloging, coordinate appending, and a user-friendly web interface, this thesis has increased the impact of IfA and created a basis for data reuse.

Kurzfassung

Das Wiener Institut für Astrophysik (IfA) betreibt zwei Teleskope in Österreich: das 1,5-m-Leopold-Figl-Observatorium (LFO) und das 80-cm Vienna Little Telescope (vlt) in der Nordkuppel der Wiener Sternwarte.
Diese Arbeit befasst sich mit dem dringenden Bedarf an einem zentralen Datenarchiv und schlägt den Bau eines modernen Datenarchivs vor.

Ziele Es ist geplant, ein automatisiertes Archiv für zwei Teleskopanlagen zu entwickeln. Historische und aktuelle Daten sollen in eine zentrale Datenbank übertragen werden. Die in der Datenbank gespeicherten Daten sollen dem Endbenutzer über einen Webdienst zugänglich gemacht werden, der die Auswahl von Daten entweder durch textbasierte Abfragen oder über eine durchsuchbare Himmelskarte ermöglicht.

Methoden Für die Datenbankverwaltung wird PostgreSQL verwendet, in der die notwendigen Metadaten gespeichert werden. Eine mit Python betriebene Pipeline dient zum Auffüllen der Datenbank und zur optionalen Einleitung des astrometrischen Lösers. Die astrometrische Lösung wird mit der Suite nova.astrometry.net durchgeführt. Zusätzlich wird ein Webservice mit dem Framework Django eingerichtet, der einen Aladin-Lite-Frame zur Erkundung des Himmels und zur Anzeige von Bildern sowie eine textuelle Abfrageoberfläche mit SQL für komplexere Abfragen bietet.

Ergebnisse Docker Compose wurde verwendet, um den Aufwand für das Hosting dieser Dienste zu reduzieren. Dies führte zu einem automatisierten, wartungsarmen Archiv, das dem Institut für Astrophysik eine angemessene Infrastruktur bietet.

Schlussfolgerung Zusammenfassend lässt sich sagen, dass diese Masterarbeit den dringenden Bedarf an einem modernen Datenarchiv am Wiener Institut für Astrophysik adressiert hat. Durch den Bau eines neuen Archivs mit automatisierter Katalogisierung, Koordinatenanhängung und einer benutzerfreundlichen Webschnittstelle hat diese Arbeit die Wirkung des IfA erhöht und eine Grundlage für die Wiederverwendung von Daten geschaffen.

Contents

List of Tables

List of Figures

Listings

1 Introduction

The Institute for Astrophysics (IfA) at the University of Vienna currently operates two telescope facilities: the 1.5m Leopold Figl Observatory (LFO) located at the Mitterschöpfl in lower Austria at an altitude of approximately 880 m above mean sea level. And the 80cm Vienna Little Telescope (vlt) is housed at the Universitätssternwarte in the 18th district of Vienna at a height of approximately 250m above mean sea level. The LFO is the largest telescope in Austria, and the vlt is the second largest. Both telescopes are used for a variety of research projects, including the study of stars, planets, galaxies, and cosmology.

Instrumentation At the Leopold Figl Observatory, the main dome was designed to have three optical paths: the Ritchey-Chrétien (RC), Cassegrain, and Caude focus. However, only the RC and Cassegrain were ever put into action. Nowadays, only the RC focus is still in use. The available instruments include PICO, a lucky imaging device with an integrated polarimeter and coronagraph, and the SBIG ST-10XME CCD system. PICO is based on CMOS technology, allowing for short integration times and rapid readouts, and is equipped with three coronagraph masks and four polarization filters. When used in the RC focus, it has a field of view of 3.6 arcminutes. The SBIG ST-10XME CCD system has a filter wheel with 10 positions, including Bessel U, B, V, R, I; sloan u, g, r, i, z, and narrow band filter for Hα, Hβ, O III, and S II. The field of view in the RC focus is 5.6x3.8 arcminutes[1].
The Vienna Little Telescope has a 6.64m focal length in the Cassegrain focus and is fitted with a CCD detector from Finger Lake Instruments. It has a filter wheel with nine positions, which includes Bessel B, V, R, I, and narrow band filters for O III, H and S II. The field of view is approximately 18.9x18.9 arcminutes[2].

Scientific Contribution Despite not being the most advanced dimensions for telescopes, the 1.5m and 80cm telescopes have still made a scientific contribution. For instance, the Leopold Figl Observatory is used to detect exoplanet transits, observe extragalactic objects (e.g., Taris et al. 2018), and research comets (e.g., Galiazzo & Zeilinger 2013). The Vienna Little Telescope is used to measure light pollution (in combination with the LFO; see Puschnig et al. 2014), study open clusters, measure variable stars (e.g., Paunzen et al. 2017; Uttenthaler et al. 2016), and investigate asteroseismology (e.g., Hürkal et al. 2005; Kolenberg et al. 2005; Handler & Meingast 2011). Both telescopes are also used to

[1] More detailed information at `https://foa.univie.ac.at/instrumentation/15m-teleskop/`
[2] More detailed information at `https://astro.univie.ac.at/fileadmin/user_upload/i_astro/Bericht-Nordkuppel.pdf`

provide students with observational experience. Despite the historical significance of the Vienna Institute, there is no modern archive that contains information on measurements from the past decades.

Current State The absence of a current archive is likely due to two factors: first, there has been no recent requirement for a high level of data accessibility, and second, no one has taken the initiative to create one. Additionally, the vlt does not embed pointing information in the images, which makes it difficult to automatically archive the data.
As a result, it has become customary to take manual notes in an observation room notebook during measurements. Although these notes are informative, they cannot be easily extracted by automated means. Moreover, measurements are not publicly accessible because the data is stored on the measurement computer and it is not desirable for untrained individuals to access them. Given these circumstances, retrieving and using the data has proven to be challenging, which has amplified the call for a centralized collection of the data.

Benefits The IfA and its researchers would benefit from the creation of a new archive. This would make it easier to access and analyze past records, which would be especially beneficial for long-term studies that require data from different time periods. Additionally, it would allow the institute to protect its legacy and ensure that its data are accessible to future generations. Furthermore, it would make it simpler for the institute to distribute its data to the public and other research organizations.

Goals The envisioned new archive is intended to automatically catalog both historical and recent images and, as part of its integration into a database, even affix coordinates when possible. Additionally, the archive is planned to feature web connectivity, providing institute members with seamless access to the data.
The Web interface is intended to draw inspiration from the ESO Archive Facility (ESO Science Portal[3]) and the Gaia Archive[4].
Vienna Sky will feature a sky map that can be interacted with, as well as a textual query interface to access the data.
The purpose of this thesis is to construct a new archive and to elucidate its functioning. This will enhance the legacy of the institute and establish reusability.

[3] `https://archive.eso.org/scienceportal/home`
[4] `https://gea.esac.esa.int/archive/`

2 Theoretical Basics

This thesis uses a variety of concepts from computer science. As astronomers may not be familiar with these terms and ideas, this chapter aims to describe the theoretical and technical concepts that are required to construct the planned data archive.

2.1 HTTP

Hypertext Transfer Protocol, commonly referred to as HTTP, is a fundamental protocol that underpins the functioning of the World Wide Web. Developed to standardize communication between web browsers and servers, HTTP governs the exchange of various resources such as Hypertext Markup Language (HTML) documents, images, videos, and more across the Internet. It forms the foundation of how information is requested and delivered. HTTP operates on a client-server model, where clients, typically web browsers, send requests to servers, which then respond with the requested content or an appropriate status code. If a website is transmitted to a web browser, the content contains instructions and markup code to visualize it in the browser.

2.1.1 HTML & Javascript

In most cases, the data sent through HTTP is in HTML format. This language is used to give the web browser (client) instructions on how to display the data, which are usually static. Often it is needed to rearrange the information, e.g. a table could be sorted by different columns; there are two basic options, the client requests the server to sort the information and produce a new response, or the client's browser does this task. At this point, Javascript takes action. Javascript is often used to do tasks on the client side, it has evolved rapidly in the last 10 years, so that it is even more often used to request additional data from the server. Many of the Javascript tasks are already covered in frameworks.

2.1.2 Uniform Resource Locator

A Uniform Resource Locator (URL) is a standard way of identifying and locating resources on the Internet. URLs are most commonly used to point to webpages, but they can also be used to refer to other types of resources, such as images, videos, and files. A typical URL like `https://astro.univie.ac.at/ueber-uns/?q=1` is composed of four components: the protocol (`https`), the domain name (`astro.univie.ac.at`), the path (`/ueber-uns/`), and the query string (`?q=1`).

1. **Protocol**: The protocol outlines the steps necessary to access a resource. The two most popular protocols are HTTP and HTTPS, which are used to access webpages.
2. **Hostname**: The name of the host of the resource is called the hostname. This name should be able to be converted into a unique Internet Protocol (IP) address, which is a numerical identifier that is exclusive to each device connected to the Web.
3. **Path**: The path is the location of the resource on the computer. For example, the path for the example homepage above is "/ueber-uns/".
4. **Parameters**: The parameters are used to provide the host of the resource with additional information. For example, this could be pre-selected options in a form or language setting.

2.2 Databases

Databases are services that store and distribute data. The main operations in a database are to create (C), read (R), update (U), and delete (D), these are in short 'CRUD' (CUDs are the ones that modify data). There exist different types of databases which can roughly be separated into relational and non-relational databases. Relational databases are based on the ACID principle, which guarantees that databases are always in a secure and well-defined condition.

- **A**tomicity ensures that in a transaction (a chain of CRUDs) all operations are treated as one unit (all or nothing).
- **C**onsistency means that every data in the database must be always in a well-defined state.
- **I**solation states that every stack of transactions, which are often run in parallel, are executed as if they were run in sequential.
- **D**urability expresses that a finished transaction is persisted.

In the selection of the database product, one needs to evaluate the available datatypes, features, and performance. Non-relational ones are not structured by tables and their relations. This is needed when it is essential to have a database that is highly adaptable; e.g. one needs to manage data that are not uniform or details, but this goes against the ACID principle.

2.3 Flexible Image Transport System

In Astronomy, there exists a file standard that enables a uniform data transfer called the 'Flexible Image Transport System' (FITS). This standard was specifically developed for astronomical needs by NASA and has been continuously developed since 1993 as needs evolve over time (Wells et al. 1981).

FITS files can contain all kinds of astronomical data provided as spectra, tables, or images. A file consists of a mandatory header and a data unit. The header stores metadata for the data and is based on a mapping between keywords and values. The standard defines some required, optional, and restricted keywords. Using necessary keywords, it is defined, e.g., whether it contains an image or a table (Ponz et al. 1994).

Basic keywords for images include:

- BITPIX defines the bit-depth of the image, i.e. an 8 would mean valid data values range up to $2^8 - 1$.
- NAXIS declares the dimension of the data, a 2D-image would have a 2, an image-cube a 3, and so on.
- NAXISn states width of the image along the n-th axis.
- DATE-OBS gives a timestamp of the beginning of the exposure.
- EXPTIME holds the exposure duration.
- FILTER names the filter used.
- SIMPLE indicates whether it conforms to the FITS standard
- BUNIT is the unit of measurement for the data
- CRPIX1,CRPIX2 pixel coordinate to the reference CRVAL1,CRVAL2
- CRVAL1,CRVAL2 world coordinates of the reference pixel
- CDELT1,CDELT2 pixel scale in each dimension

There are many more keywords providing information about the telescope, detector, pointing, detector's readout mode, and many more. However, in principle, it is possible to infer the whole state of the setup in which the image is taken, nevertheless, the observatory decides which keywords are included (sometimes observatories do not have all information like exact pointing).

3 Data

The measurement computer currently holds FITS files, which can be divided into two types: calibration frames and science frames.

Frame types Calibration frames are necessary to address the various artifacts and flaws that can be present in astronomical images. These can be caused by the camera sensor, the telescope, or the atmosphere.

Bias frames are taken with the lens or telescope cap on to reduce the noise level of the camera sensor, which is a small amount of noise present in all images. These frames are usually very dark and contain only a small amount of noise caused by the camera's readout electronics.

Dark frames are taken with the lens or telescope cap on and the same exposure time as the science frames to reduce the thermal noise generated by the camera sensor when exposed to light. These frames are similar to bias frames, but contain more noise due to thermal heat that is generated in the camera.

Flat frames are taken with a uniformly illuminated surface, such as a white screen or a twilight sky, to reduce the vignetting effect, which is a darkening of the corners of the image caused by the lens or telescope.

Different techniques can be used to process the raw science frames, and the process generally follows the pattern outlined in Equation 3.1. In this equation, 'SF' stands for raw science frames, 'SFc' for corrected science frames, 'BF' for bias frames, 'DFn' for bias subtracted dark frames normed to 1 second of exposure, and 'FF' for flat frames.

$$\mathrm{SFc}_i = \frac{\mathrm{SF}_{i,t1} - \mathrm{BF} - \mathrm{DFn} * t1}{\mathrm{FF}_{t2} - \mathrm{BF} - \mathrm{DFn} * t2} \tag{3.1}$$

The IfA has collected several thousand images over the years, yet some of them have been lost due to mistakes in the way the data was managed. A smaller subsample of the existing data, 500 frames, was chosen to be used to test and create this software system.

Image previews Figure 3.1 shows two illustrative images that provide a compelling insight into the compositional spectrum of the Vienna Little Telescope dataset. These images serve as examples of the telescope's expansive utility. The first image prominently features the Bubble Nebula, an intricate emission nebula documented via the vlt in the narrow Hα band. The second depiction shows NGC 2281, an open cluster with an age of 435 ± 50 million years (Fritzewski et al. 2023).

Images from the Vienna Little Telescope (FOV of 19x19 arcminutes, Figure 3.1) and the Leopold Figl Observatory (FOV of 3.5x3.5 arcminutes, Figure 3.2) show significant

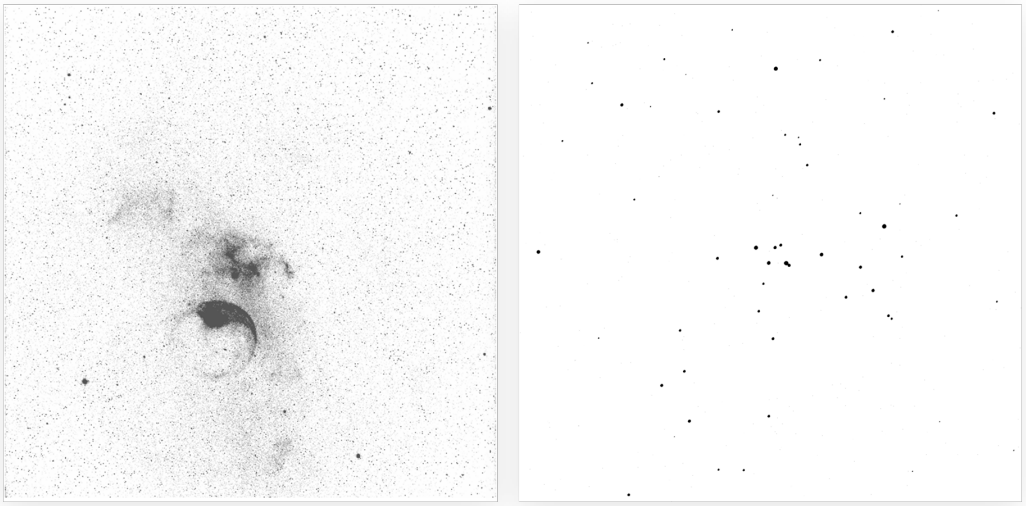

Figure 3.1: Raw images taken from the Vienna Little Telescope are displayed, the left panel shows the Bubble Nebula in the $H\alpha$ band, and in the right panel shows NGC 2281 in the V band. The pictures illustrate luminosity levels, which are reversed, so that the dark is actually brighter. As the images are not calibrated, they contain a lot of noise and artifacts.

differences. Presented below is an illustrative instance captured through the Figl Observatory. As illustrated in Figure 3.2, the constrained field of view and the image scaling used to reveal a singular stellar entity.

3.1 Image Properties

It is customary for filenames at the IfA's northdome to adhere to a certain "standard". This includes beginning with the date the observation night began, followed by the frame number, type, and filter. Unfortunately, since filenames are entered manually, they are susceptible to errors, such as typos and incorrect applications. Therefore, these data should not be trusted and must be verified by examining the associated header. (see verbatim)

```
SIMPLE  =                    T
BITPIX  =                   16 /8 unsigned int, 16 & 32 int, -32 & -64 real
NAXIS   =                    2 /number of axes
NAXIS1  =                 2048 /fastest changing axis
NAXIS2  =                 2048 /next to fastest changing axis
DATE-OBS= '2021-04-28T20:40:29' /YYYY-MM-DDThh:mm:ss observation start, UT
EXPTIME =   10.000000000000000 /Exposure time in seconds
```

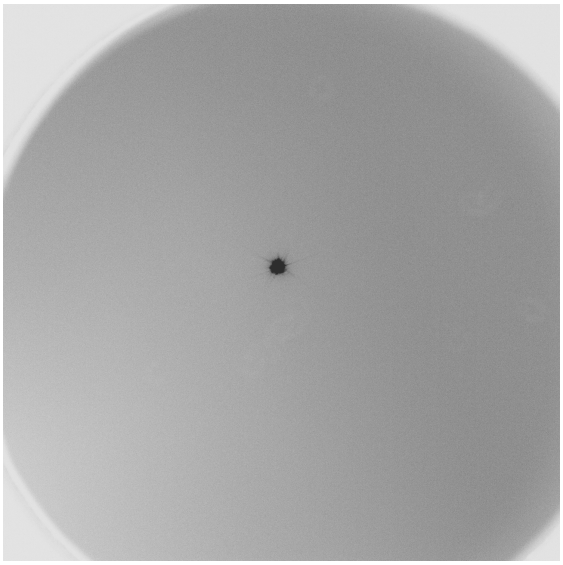

Figure 3.2: Images taken using the Leopold Figl Observatory may not display many objects due to its restricted field of view. Additionally, the telescope configuration is affected by strong vignetting.

```
EXPOSURE=   10.000000000000000 /Exposure time in seconds
CCD-TEMP=  -15.000000000000000 /CCD temperature at start of exposure in C
XPIXSZ  =   18.000000000000000 /Pixel Width in microns (after binning)
YPIXSZ  =   18.000000000000000 /Pixel Height in microns (after binning)
READOUTM= '8 MHz   ' /          Readout mode of image
FILTER  = 'B       ' /          Filter used when taking image
IMAGETYP= 'Light Frame' /       Type of image
FOCALLEN=   6640.0000000000000 /Focal length of telescope in mm
APTDIA  =   800.00000000000000 /Aperture diameter of telescope in mm
APTAREA =   502654.83856201172 /Aperture area of telescope in mm^2
SITELAT = '48 13 55' /          Latitude of the imaging location
SITELONG= '16 20 01' /          Longitude of the imaging location
JD      =   2459333.3614467592 /Julian Date at start of exposure
JD-HELIO=   2459333.3569522165 /Heliocentric Julian Date at exposure midpoint
OBJECT  = '20210428_'
TELESCOP= '        ' /          telescope used to acquire this image
INSTRUME= 'FLI     ' /          instrument or camera used
OBSERVER= '        '
END
```

The header contains information about the camera setup used to acquire the image. This information can be trusted as it is set by software, and the individual parameters are set only by the observers. A major issue with the image headers currently available is that

they do not include any directional information. Consequently, one would expect the keyword 'OBJECT' to indicate the pointing direction of the object of interest, but this is not the case in the example above. The goal is to have integrated pointing information that would guarantee re-usability.

World Coordinates in FITS The FITS standard defines how pixel coordinates can be translated to world coordinates and is explained in the paper Greisen & Calabretta (2002). Generally, one detector coordinate is linked to one celestial coordinate. Additionally, a matrix is established that offers a straightforward linear transformation between the two coordinate systems. Equation 3.2 of the paper Hanisch & Wells (1988) is used to convert relative coordinates, denoted "Δ", into absolute coordinates. The necessary keywords from the header are listed in Table 3.1, and Equation 3.3 is used to convert to absolute sky coordinates (SC_1 and SC_2). The detector coordinates (x and y) and the keyword CRPIX can be used to convert them to absolute detector coordinates.

$$\begin{bmatrix} \Delta\mathrm{SC}_1 \\ \Delta\mathrm{SC}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{CD}_{11} & \mathrm{CD}_{12} \\ \mathrm{CD}_{21} & \mathrm{CD}_{22} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} \mathrm{SC}_1 \\ \mathrm{SC}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{CRVAL}_1 \\ \mathrm{CRVAL}_2 \end{bmatrix} + \begin{bmatrix} \Delta\mathrm{SC}_1 \\ \Delta\mathrm{SC}_2 \end{bmatrix} \tag{3.3}$$

```
CTYPE1  = 'RA---TAN'           / WCS projection for axis 1
CTYPE2  = 'DEC--TAN'           / WCS projection for axis 2
EQUINOX =               2000.0 / Equatorial coordinates definition yr]
CRVAL1  =       222.6695419145 / RA of reference point
CRVAL2  =        74.1415486602 / Dec of reference point
CRPIX1  =        1296.74008179 / Reference pixel on axis 1
CRPIX2  =         634.63677597 / Reference pixel on axis 2
CUNIT1  = 'deg     '           / Pixel scale units for axis 1
CUNIT2  = 'deg     '           / Pixel scale units for axis 2
CD1_1   =   -2.77509687214E-05 / FITS CD transformation matrix
CD1_2   =  -8.790488559414E-06 / FITS CD transformation matrix
CD2_1   = -8.9877306115037E-06 / FITS CD transformation matrix
CD2_2   = 2.79943302851589E-05 / FITS CD transformation matrix
```

Listing 3.1: An example of a FITS header is presented here to demonstrate the WCS-related keywords.

4 Methods

This chapter discusses the requirements of the software developed in the context of this thesis. Big observatories around the world have employed sophisticated data archives. For example, the European Southern Observatory (ESO) offers data access via their Science Portal (shown in Figure 4.1). The user is unaware of the majority of the components,

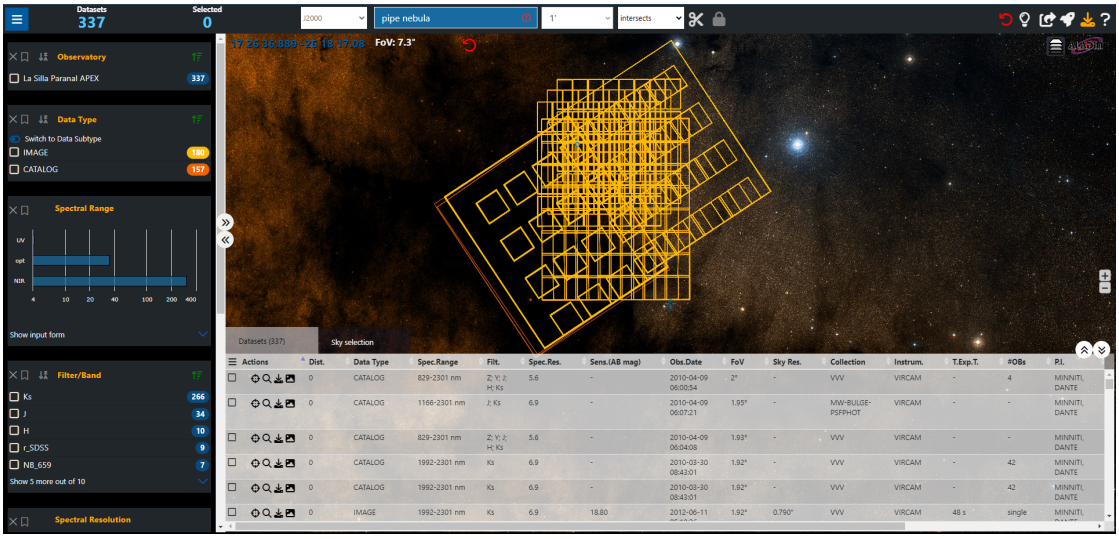

Figure 4.1: Users search and navigate through data using the interactive sky map deployed by the ESO Science Portal. All available images and data are displayed on a sky map, as well as in the table below. By simply checking the boxes, one can choose and download the needed data. Here is a screenshot when navigating to the Pipe Nebula. `https://archive.eso.org/scienceportal/home`

apart from the visualization. All the services provided are based on complex algorithms and database management that are not visible to the user. This includes, for instance, an automated pipeline that adds new observations to the archive, creates an astrometric solution, and many more jobs. In the following sections, these jobs are explained further.

4.1 Astrometric Solution

Celestial coordinates are essential for astronomical image archives. They allow users to quickly and easily locate the images they are looking for, regardless of the time or location of the observation. In addition to providing a way to locate images, celestial coordinates

also allow users to compare images taken at different times and from different locations. This can be useful for studying the evolution of astronomical objects or for comparing observations from different telescopes. Without celestial coordinates, astronomical image archives would be much less useful. They would be much more difficult to search and would not allow users to compare images taken at different times or from different locations.

Most observatories use robust systems to ensure that the pointing information for each image is accurately recorded in the image header. This ensures that the celestial coordinates associated with each image are accurate and reliable.

In certain scenarios, this approach is not applicable. For instance, in our setup, the computer that operates the telescope is unable to communicate with the computer that manages the camera, and thus the images do not include any celestial coordinates. This issue can be addressed by adding pointing information to the image during post-processing. This process is called plate solving and involves identifying objects, extracting their detector pixel positions, and correlating them with the given celestial coordinates. Suitable reference coordinates for this purpose can be obtained from astronomical databases such as Gaia or 2MASS. By repeating this process for multiple objects, a transformation from detector coordinates to sky coordinates and vice versa can be established.

4.1.1 Blind Solving

Blind solving is a special kind of plate solving in which no initial guess is given. It cannot be done by hand and, therefore, requires computational power. Usually, it is based on detecting sources in the frame, creating connections to nearby sources, and weighting them by their intensity. With this net and its intrinsic angles, a computer tries to match it to known structures. In most cases, these compare catalogs come in different image scales and are divided into healpix-like grids with overlaps between the healpix fields (see Lang et al. 2010). From these catalogs, the same net is produced and is tried to match by rotating and scaling it and vice versa.

I will now compare three of the most popular astrometric solvers: SCAMP, PlateSolve2, ASTAP, and Astrometry.net. I will look at their features, advantages, and drawbacks.

Astrometric SCAMP (see Bertin 2006) is a free and open-source software package for automatic astrometric and photometric calibration of FITS images. It produces solutions very fast and accurate. However, it is complex to configure and requires prior information, such as pixel scale and rough pointing.

PlateSolve2(N.I.N.A. 2022), developed by PlaneWave Instruments, is commercial software that allows users to compare their data with the APM and UCAC3 star catalogs. It is a robust solution for astrometric calibration, but it is important to note that it is no longer actively developed. Although this may not be a major concern for some users, it is worth considering for those who rely on regular software updates and improvements.

The Astrometric Stacking Program[1] (ASTAP) is a comprehensive suite for deep sky image stacking, including a variety of programs for image correction and stacking based on astrometry. ASTAP is a powerful tool for astrophotographers and researchers alike,

[1] `https://www.hnsky.org/astap`

Name	Advantages	Disadvantages
astrometry.net (solver only)	• Open Source • Multiple reference catalogs • No false positives • High accuracy solutions	• Only command line
nova.astrometry.net (clone)	• Extension of astrometry.net • Provides an HTTP-API	• Host extra Webservice
ASTAP	• Open Source • Handles various formats	• Complex configuration • Only GUI or CLI
PlateSolve2	• Free to use	• Only GUI • Commercial
SCAMP	• Open Source • fast • very accurate	• complex to setup • requires priors

Table 4.1: Comparison of astrometric solvers. nova.astrometry.net inherits the advantages of astrometry.net. Resulting in a favorable service to include.

but ASTAP is not open source and is no longer actively developed, which means that new features and bug fixes may not be available.

Astrometry.net(Lang et al. 2010), developed by the University of Waterloo, is a versatile and efficient astrometric solver that is specifically designed to be fast and reliable. It produces accurate results with a minimal chance of false positives (incorrect solutions). Moreover, the availability of nova.astrometry.net, a web interface to the Astrometry.net backend, improves user accessibility and usability. What sets Astrometry.net apart is its open-source nature and active development community, ensuring that it remains current and adaptable to evolving astrometric needs. Additionally, Astrometry.net allows users to select the catalogs they wish to compare against, offering flexibility in their astrometric analyses.

Overall, nova.astrometry.net is a good choice for use in an archive pipeline because it is free, open-source, web-based, supports more catalogs, and is fast and accurate. In addition, nova.astrometry.net is also actively maintained and updated, which is important for a software package that will be used to process future data.

4.1.2 Astrometry.net

In this section, I describe the blind-solving algorithm used in the developed archive system. It is desirable to run the service locally to avoid sharing computing resources. The service is provided as an unconfigured Docker container with a Web API that allows users to submit images for solving [2]. The solver produces pointing information that is compatible with the FITS standard and identifies objects likely present in the submitted field. The configuration of the container can be found in the next chapter in the listing

[2]The used Docker image can be found at `https://github.com/dam90/astrometry`

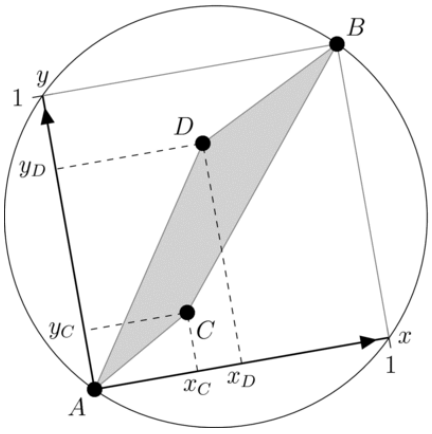

Figure 4.2: The astrometry.net solving procedure (described in Lang et al. 2010) picks four of the brightest sources that lie in a rhombus. The first two sources (A and B) are selected based on intensity, and then the algorithm searches for two additional sources that are within the circle that is centered on A and B and has a radius equal to the distance between A and B.

5.3. The coordinate solution method is based on quads, which are quadrilateral regions. The algorithm detects sources in the image and extracts their instrumental intensity and location in the frame as illustrated in figure 4.2. This process is repeated several times in the image. The developers of the algorithm defined further constraints, such as $x_C + x_D \leq 1$ and $x_C < x_D$, to create a system that is invariant to translation, rotation, and scaling.

Translation invariance is a fundamental property denoting that a system consistently produces identical results, irrespective of shifts in the input's spatial orientation. This principle is upheld by formulating constraints based on the relative positions of points, thus eliminating the reliance on their absolute coordinates. As an illustration, consider the constraint $x_C + x_D \leq 1$, which exemplifies translation invariance, as it exclusively relies on the spatial separation between points C and D.

In the context of rotation invariance, the system is designed to yield uniform outcomes even when the input undergoes rotational transformations. This is achieved through the expression of constraints in terms of the angles formed between points rather than their absolute orientations. For example, constraint $x_C < x_D$ is a prime example of rotation invariance, as it is only dependent on the relative order of points C and D.

Scaling invariance is a characteristic where the output of a system remains the same regardless of changes in the size of the input. This is accomplished by formulating restrictions that take into account the proportions of the distances between points, rather than their absolute distances. To illustrate, constraint $x_C + x_D \leq 1$ is an example of

scaling invariance, as it relies solely on the relationship between points C and D and the total distance of the four points.

The algorithm attempts to replicate the same set of coordinates (x_C, x_D, y_C, y_D) as those stored in the comparison catalog.

The astrometry suite offers reliable coordinate solutions by eliminating false positives. This implies that if a solution cannot be determined with a high level of certainty, it will rather not provide one than an incorrect solution, which is highly beneficial in scientific practices.

Interactions

The astrometry suite's source distribution includes a Python file designed to interact with its HTTP service. With minor modifications, it can be adapted to work with the local service. This client is utilized to communicate with the astrometry service's local HTTP binding. It facilitates actions such as uploading an image for processing or retrieving information about a file's current status, such as its solver status or WCS solution.

4.2 Pipeline

In order to migrate the FITS files to a searchable database on a computer, an automated workflow is needed. The main parts of an archive are its up-to-dateness and completeness, which can be guaranteed by updates on a regular basis. In particular, the pipeline has the task of periodically searching for new FITS images in a specified folder. From the new images, relevant header attributes are extracted, then classified according to the found setup, and inserted into the database. If the pointing information is missing, which is decided by the (missing) `CTYPE` FITS header value, it is tried to be provided by the local astrometry service. In case of success, the database entry is updated with the WCS information. By automating this process, the University Observatory data management system can be kept up-to-date, with just a minimum delay in terms of data availability. To ensure easy adaptations of the pipeline in the future, several configuration options are adjustable in a settings file (see Section 5.3.4).

Comparison of languages The pipeline can be implemented in multiple programming languages. However, in terms of convenience and maintainability, the selection is reduced to the following:

1. Python
    - widely used in the field of astronomy
    - currently taught language at the institute
    - version dependant
2. Java
    - Backward compatible

- Runs on any system

3. PHP
   - web-oriented language

All of them are object-oriented programming languages, which should be able to handle the tasks of the pipeline. However, the list is sorted by my current familiarity and is therefore considered as a preference.

4.3 Database

When considering data structures that involve multiple attributes, including metadata, it becomes important to consider the use of databases. Databases are designed to handle large datasets, although various implementations come with their own distinct advantages and disadvantages. There is no universally perfect database management system. To begin with, one must assess the level of flexibility required. If you are dealing with uniform data, you may ask yourself if all the entries in the database have the same attributes. For instance, an image could be associated with characteristics such as resolution and when it was created, while a plain text file would not have resolution but would include a character count. In simpler terms, if we understand the different types of data that we will encounter, we can employ a more rigid database structure. Whereas, if we lack awareness of all the potential types, a more flexible approach might be necessary.
The data consists of FITS files, which therefore are rather well-defined, and favor relational databases. Relational databases consist of tables. Each table has multiple columns, however at least one column must be defined as a unique 'primary key', this gives an exact identifier for each row in the table. It is common to use the table row number, but it would also work to identify a file in a file table by its path. Further tables can be linked by foreign keys. If a column has a foreign key constraint, then just the primary key of the referenced 'foreign' table row is written in this column. This gives multiple advantages, for example, reduced data size, data abstraction, and link of information. The physical data size is reduced when, instead of repeating strings, such as the filter name 'Johnson V', just a number is written, and whether it is useful to outsource information depends on the value periodicity. Another point to consider during database design is that a table, or even just one row of it, becomes locked when being written or updated. As a result, it is often recommended to add additional generated information into a separate table and link it through foreign keys. Otherwise, if a row is locked, many operations would be discarded.

4.4 Online Archive

To ensure easy and natural access to the data, a web interface can be developed that allows interactive selection of images on a sky map. The display of the images on the map is based on the edges of the images. However, some frames, such as correction

images, may not be suitable for visualization on a sky map. To provide access to this type of data, a text-based interface is created. Using this interface, the selection of these frames is possible based on certain criteria. The web interface can also provide advanced search options, such as filtering by observation date, telescope setting, and other relevant metadata. By providing such a user-friendly interface, the observatory can provide easy access to and analysis of the collected data.

4.4.1 Interactive Skymap

Aladin (Boch & Fernique 2014) is one of the most popular sky maps used in scientific research. Its lite version can be embedded into webpages and allows users to add objects such as rectangular overlays using JavaScript. Additionally, it has action listeners that provide client-side interaction when an object is clicked. This enables users to select frame tiles in the sky. However, because of the high number of frames, it is not feasible to load all frame borders at once into the Aladin container. Instead, it is detected when the sky map has been moved, and then it requests all frames that were inside the viewing area using Ajax. Ajax is a technology that allows requesting additional data loaded from the web server to the client. Therefore, the sites are initially small and grow as more data is demanded. This is crucial if one wants to avoid reloading the webpage.

4.4.2 Textual Query Interface

When the data structure is final, one can implement a text field that allows the insertion of plain SQL. Using this method allows for precise selection of data that follow a scheme that is also used by Gaia Archive https://gea.esac.esa.int/archive/. In the text fields, only custom operations that are required for the front end will be allowed. This includes SELECT for all tables and INSERT/DELETE just to download relevant tables.

4.5 Docker

When building services that require a lot of subservices, e.g. website, database, and load balancing, it is favorable to have each subsystem in its own machine to encapsulate these from each other. This makes updating and even swapping subsystems easier. However, by emulating whole machines, one faces a huge expense in resources. Therefore, there exist so-called containers, these constructs are very familiar to Virtual Machines, each container has its own file system, but instead of its own operating system, these have a layer that translates tasks for the host OS, so that the overhead of an additional OS is removed. Docker is a popular containerization software that is often used in conjunction with Platform-as-a-Service (PaaS). PaaS was developed to simplify the complex setup of cloud servers and their underlying infrastructure. In a PaaS model, applications are grouped into modules that can include databases and web services, among others. Since cloud servers typically host multiple applications and modules, it can be difficult to ensure that they do not interfere with each other. To address this, PaaS provides each application with its own isolated database instance, which gives each application an own

platform. Docker comes with extensions, e.g. Docker Compose; these allow managing a big conglomerate of Docker systems.

4.5.1 Images

Docker images represent a single system setup. An image sets the used operating system including used packages and exposed ports, as a container rejects all incoming connections except the previously defined ones. For many services, there exist pre-cooked images; however, it is possible to extend images by creating an own `Dockerfile`, which gives the opportunity of full customization. There are many base images for most operating systems, so it is possible to create suitable environments for each program. In general, images act as blueprints for containers.

4.5.2 Container

A Docker container is an instance of an image. Containers can be further configured, including so-called bindings, that mount a directory of the host system into the container, this is useful for persistence. Furthermore, it is possible to map ports from the container to the host or limit certain resources, such as CPU time or memory.

4.5.3 Compose

Docker compose is a utility to organize the collection of Docker containers/images. By using compose, it is possible to create setups for each project, e.g. which images to run, and even specify their startup parameters. Profiles for different startups like debug or production are also possible, and enable easy scaling as it can start and manage multiple instances of the same image, for further scaling Kubernetes should be considered.

Docker compose is an extension to Docker, it gives the user the opportunity to create setups of multiple containers so that these can be configured by just one file. Via Docker Compose, one can specify how many instances of an image should run, on which ports each container should be available, and which paths should be mounted in the individual containers. Using Docker, it is possible to group a complex structure of individual applications into one big application, as is done here. Each service has its own image.

5 Implementation

This chapter contains implementation-specific details. With the aid of this chapter, one should be able to gain a comprehensive understanding of several crucial components, including the database structure, the web service, the blind solver, and the pipeline configuration. This knowledge will provide valuable information that was used to obtain a solution to the complex set of problems and lead to the results.

5.1 Database

PostgreSQL is used as database system, for which there is an official Docker image. The database directory is mounted on the host system. The database works on the basis of two users: a back-end user who is allowed to change all tables and is used to fill the database with information, such as extracted data from image metadata, and a front-end user who is only allowed to select all tables and modify two tables that are required for online functionality. Furthermore, the front-end user can create views on the subschema ‘public.views‘, which are designed to create pre-defined queries. These views can be used in the same way as tables, allowing selection from them. This is a useful tool to reduce the amount of time spent writing queries, and they can also be included in the textual query interface when executing the query. Figure 5.1 shows the existing tables and relationships.

Database structure All tables (except ‘lframe_info‘) have a natural primary key which is the row number and is called ‘id‘.

Everything builds around the central table ‘file‘, which has the following attributes:

- ‘path‘: The physical path of the file on the server.
- ‘type‘: A single character, ‘B‘ for bias, ‘D‘ for dark, ‘F‘ for flat, ‘L‘ for light, and ‘U‘ for unknown image type. In the future, other types such as ‘S‘ for spectra could be supported.
- ‘exptime‘: The exposure time used to acquire the image.
- ‘wcs‘: The string returned from the Python method ‘astropy.wcs.WCS.to_header_string‘[1].
- ‘central_ra_dec‘: When the image is solved using the plate solving algorithm, this value is set to the position that is totally in the middle of the view. It is thought of more as a guide when manually browsing the table.

[1] `https://docs.astropy.org/en/stable/api/astropy.wcs.WCS.html#astropy.wcs.WCS.to_header_string`

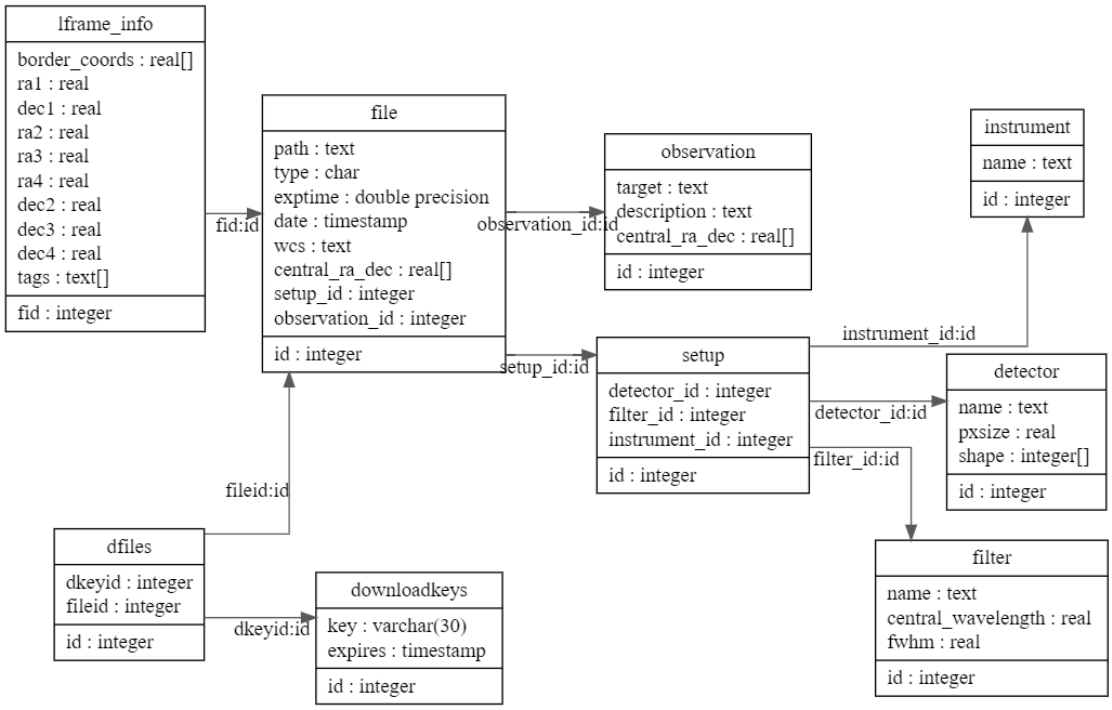

Figure 5.1: Entity Relationship Diagram of the deployed data structure. This diagram shows the relationship between the tables and their rows. On the connection lines, the tuple of column names is prompted to show that the relation is based on these columns.

- ‘setup_id‘: A reference to a row in the ‘setup‘ table with a value of ‘setup.id = setup_id‘.
- ‘observation_id‘: It is currently not used, however, the idea is to provide the possibility to group certain files together like for surveys.

The table ‘setup‘ consists of three references, that point to the tables: ‘detector‘, ‘instrument‘, and ‘filter‘. With these references it is possible to define the physical setup used to obtain the image.

The table ‘instrument‘ is supposed to hold a record of the telescope or spectrograph used, and is identified by the FITS header value of the key ‘TELESCOP‘.

The table ‘detector ‘ contains information on the camera used and is recognized by the value of the FITS header key ‘INSTRUME‘. Furthermore, the attributes ‘pxsize‘ and ‘shape‘ associate the size of one pixel and the physical dimensions of the pixel array.

The table ‘filter‘ holds information about the used filter, and is identified by its FITS

URL path	functionality
/	startpage
admin/	Administration page
ajax/	Communication interface for user actions on the interactive sky map.
dbview_files/	Listing of all file entities in a format that is easy to read.
download/	Download interface for non-modified FITS files.
download_w/	Download interface FITS files with added pointing solution.
lists/	Displayes available views
map/	Interactive map page
onlinequery/	Textual query page
oqr/	Result page of the textual query

Table 5.1: Overview of the available server paths and their functionality

header key ‘FILTER‘. Additionally, it is possible to set manually astronomical filter parameters, i.e. the central wavelength and its full-width-half-maximum (FWHM).

The tables ‘dfiles‘ and ‘downloadkeys‘ are made to provide limited download functionality. The table ‘downloadkeys‘ consists of a 30 character long random generated string called ‘key‘ and a corresponding expiration date called ‘expires‘. These keys are related to entities of the table ‘file‘ via the table ‘dfiles‘. ‘dfiles‘ just holds sets of tuples with IDs of the table ‘file‘ (‘fileid‘) and the table ‘downloadkeys‘ (‘dkeyid‘). These two tables are managed by the front-end, and with these, it is feasible to limit download access on files so that it is generally not possible to flood the server with random file requests. This is elaborated on further in section 5.2.3.

The table ‘lframe_info‘, which stands for "light frame info", has its primary key called ‘fid‘ and must be exactly the same number as the associated row ‘id‘ in table ‘file‘. The table itself holds information about the on-sky coordinates of the corners of the image denoted as ‘ra‘(1-4) and ‘dec‘(1-4), and additionally the column ‘border_coords‘ is a list of tuples of ‘ra(1-4)‘ and ‘dec(1-4)‘. When the image was solved using the plate solving service, it also holds information of objects in the frame called ‘tags‘.

5.2 Webservice

The web service is based on Django (Django Software Foundation 2022). In Django, one defines URLs or even url patterns[2] in `url.py`. There, these HTTP requests are converted into method calls, which then generate webpages and content for the desired functionality. The URL mapping used can be found in Table 5.1.

For the user experience in Django, one can work with templates which are HTML files that contain special block instructions. These blocks can be substituted for each

[2] `https://docs.djangoproject.com/en/4.2/topics/http/urls/#example`

individual page that is shown. Furthermore, it is also possible to create templates that extend an existing one.

A base template is defined to establish the general appearance of the website. The interactive map utilizes an extended template, allowing the setup of the Aladin-Lite frame and a table without requiring these instructions to be supplied via Python. The same principle applies to the textual query interface page.

5.2.1 Interactive Map

The interactive skymap is available via the path `map/` (shown in Figure 5.2). On this page, an Aladin frame is loaded into the template. The Aladin frame is initially centered on NGC 2281, as it is one of the most observed objects at the Vienna Little Telescope. Servers are typically unaware of what the end user is doing in their web browser, so it

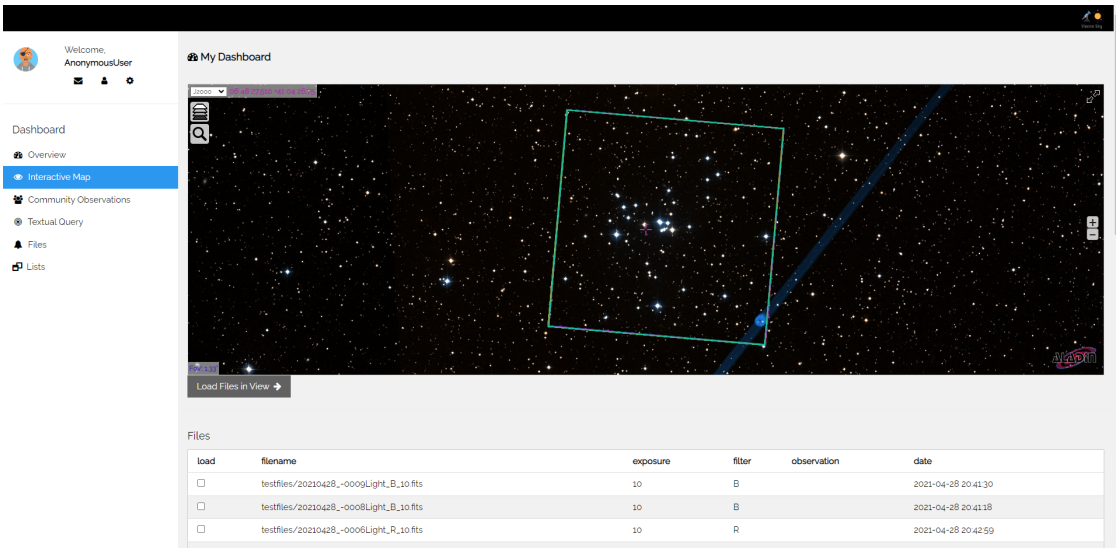

Figure 5.2: The interactive sky map has an Aladin Lite frame, which allows users to explore the sky. If frames are present in the area that is currently visible, then the frame borders will be displayed on the map. In the image, the table below lists the attributes associated with the visible frames.

is necessary to instruct the browser to communicate certain user activities, which are referred to as events. For instance, the server does not know whether the sky map is moved to a different celestial position. However, with events, it is possible to react to such phenomena. The ‘positionChanged‘ event is triggered every time the pointing in the Aladin frame (the sky frame in Figure 5.2) changes. Then, a method provided by Aladin extracts the corner coordinates of the visible frame and sends them using Ajax to the path ‘/ajax/‘.

There, a Python method makes a query to the database. The query method first checks if $|ra_{fov,max} - ra_{fov,min}| > 180$ is used to determine whether the zero point of the right

ascension (RA) is within the field of view. If the zero point is not within the current FoV, then the query requires at least one corner coordinate pair i of the image where $ra_{fov,min} < ra_i < ra_{fov,max}$ **and** $dec_{fov,min} < dec_i < dec_{fov,max}$. However, if the zero point is included, the RA constraints change to $ra_{fov,max} < ra_i$ **or** $ra_i < ra_{fov,min}$.

Therefore, all frames that have at least one corner coordinate within the FoV are selected. Subsequently, all relevant information displayed on the site is retrieved and converted into a JSON response. In the browser, the JSON response is parsed and the frame borders are drawn while populating the table with additional information. However, to select frames from different Aladin FoVs, the currently selected frames in the table are obtained using jQuery. The rows are copied into a temporary variable, and then the table is cleared and filled with the temporary information, along with the currently visible frames in the FoV.

For convenience, below the table, there are buttons that allow one to select or deselect all the rows of the table using jQuery. In addition, there are radio buttons that provide the option to choose between downloading frames with or without pointing information.

5.2.2 Textual Query Interface

The textual query interface page loads a 'codemirror' plugin to have an editor textfield with syntax highlighting. Below, one can choose between two types of results. Either a table is the result, so that one sees the output of the query on the next page, or an image, then a download script is generated. For the image result, it is important that the query starts with a `select ... from file` because otherwise it is not possible to reconstruct which files should be selected. For a file to be uniquely identified, its primary key `file.id` must be included in the query. If this is not done, the file cannot be identified, and thus cannot be downloaded. In case the result is a table, the executed query response contains the column names, these are then also used to display the table. Like the interactive map, it provides an option to download the files including the pointing information.

5.2.3 Download Functionality

The database incorporates a download mechanism that ensures secure file retrieval and mitigates abuse. This mechanism is based on two tables: `downloadkeys` and `dfiles`. When a download request is made, a unique key is generated and associated with a set of file IDs. This key-based system serves multiple purposes, including preventing server flooding and facilitating efficient retrieval of files, particularly when adding pointing information on the fly to FITS files, which may require additional processing time.

To initiate a file download, users can access the URL `download/fid/key/`, where the integer parameter `fid` specifies the file ID, and the string parameter `key` is the generated key that validates the download request. This URL provides direct access to the file stored on the hard disk drive. Alternatively, users can use a slightly modified URL, `download_w/fid/key/`, which not only retrieves the file but also appends the pointing information to the file response. For files with appended pointing information, a temporary file is created in memory. These files are generated on the fly and are intended for one-time

use. These measures help prevent abuse by minimizing opportunities for unauthorized access and misuse of the system's resources. To increase security even more, the generated keys have a predetermined validity period of two weeks (although, following the same steps as in Section 6.1.6, it is possible to adjust the validity to any length). This ensures that access to the download links is time limited and restricted to authorized users. In addition, the system can implement other security measures, such as authentication and validation processes, rate limiting, or usage quotas, to minimize the potential for abuse.

By implementing these measures, the database download mechanism aims to strike a balance between providing convenient access to files and safeguarding against misuse and server overload.

5.3 Pipeline

The pipeline is divided into four logical parts (classes). A Unified Modeling Language (UML) can be found in figure 5.3. The program employs a modular design that effectively separates various communication interfaces, enhancing its overall functionality and maintainability. Each communication aspect is encapsulated within a distinct class, optimizing code organization, and promoting reusability.

The first class is dedicated to seamless communication with the database, ensuring efficient data retrieval and storage. Establishes a reliable link between the program and the database.

Another crucial component is the astrometry solver class, as the solver service itself is designed as a Web service. It is distributed with the astrometry.net suite, however, some changes (elaborated in section 5.3.3) had to be made to provide better accessibility, e.g., to solver logs.

Interacting with the FITS files stored on the hard disk is facilitated by a specialized class. This component manages file input operations, offering read-only access to the FITS files so that the data maintain its original state.

Finally, to govern the program's flow of its different components, a dedicated control class is implemented. This class acts as the brain of the program, operating the interaction between the database, solver, and file handler classes.

By adopting this modular approach, the program becomes more organized, maintainable, and easier to extend with new features or adapt to changing requirements.

5.3.1 Controller

The controller is responsible for controlling the flow of the program. The standard workflow is as follows:

1. Scan a folder for paths of possible images (read_directory). This method returns a list of paths.

2. Pass each path to handle_fits_file. This method extracts and parses the header using the Fitsparser. This results in a dictionary with root nodes named like the

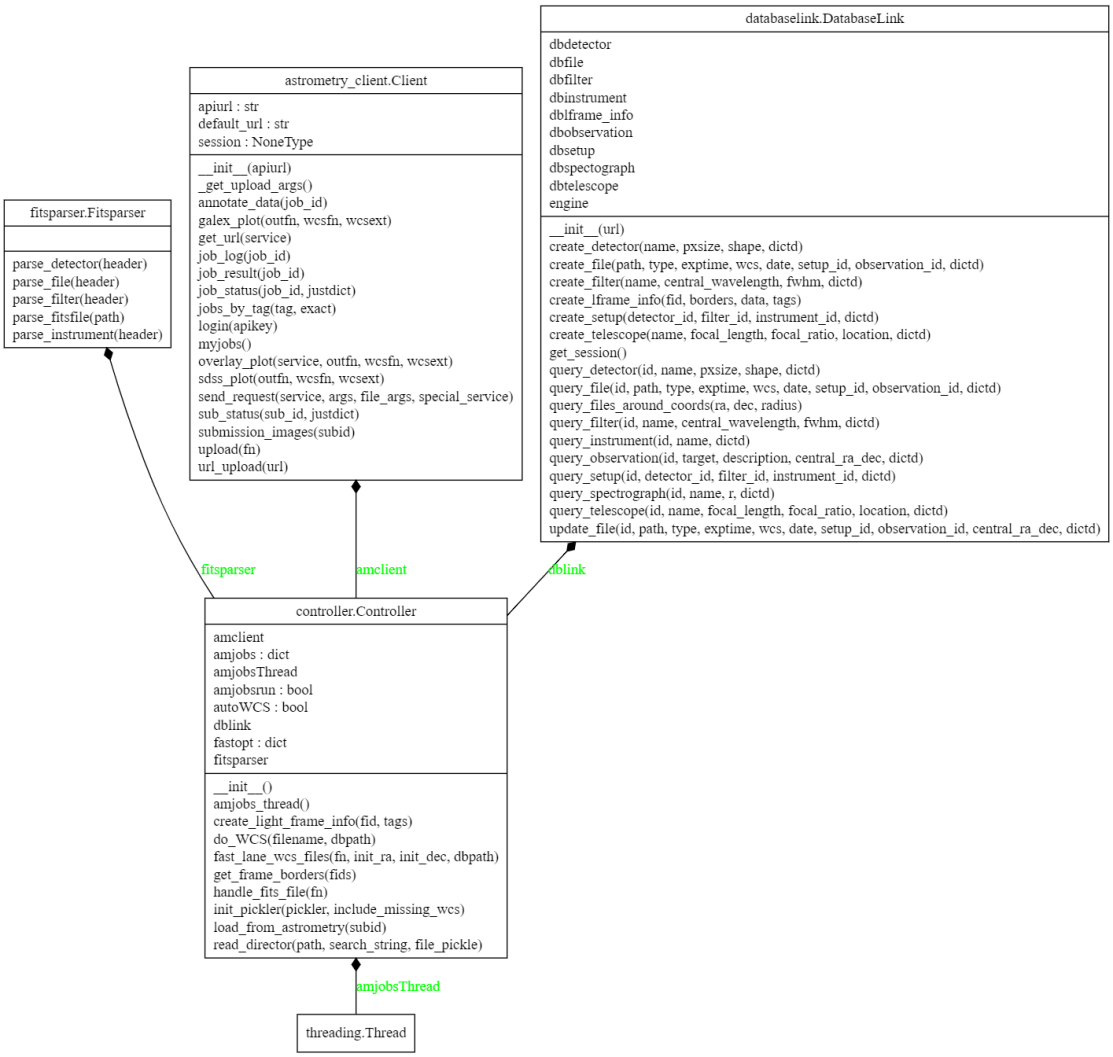

Figure 5.3: The UML diagram shows the dependencies and structure of the pipeline. The 'Controller' class executes the flow actions using the other classes. The 'FITSparser' class parses the FITS files and extracts the metadata for the database. The 'DatabaseLink' class is the only class that interacts with the database. The 'Client' class controls the plate solving process.

database tables, e.g. file. The individual entities in the database are either created or referenced (by querying their primary key).

3. Check the header for WCS information. If it is not available, call the do_WCS method. This method submits the file to the plate solving service and tells the thread to look for its solving status.

4. Once finished, call create_light_frame_info to calculate the celestial coordinates of the corners of the image. This method is called from inside the threading method amjobs_thread.

5. The threading method amjobs_thread periodically checks the dictionary amjobs. This dictionary is used to associate the image ID and path to the astrometry service internal jobs and submission number. amjobs_thread checks the state of the solver. Once the solver finishes successfully, the WCS information is written into the database.

Optionally, one can choose to read all locally available paths and cross-check them with the database first. If a file has a record in the database, it is skip processing it in the controller. One can choose to skip either all files with a record in the database or only those files with WCS information. This is done using the method 'init_pickler', if the argument 'include_missing_wcs' is true then files with missing WCS information are skipped.

5.3.2 Database Connection

The database connection is established using SQLAlchemy, which provides an object relational mapping (ORM) approach. This allows the creation of Python classes that represent existing table schemas. This enables seamless interaction with the database, including the ability to extend a table by adding a column without requiring changes to the Python code.

On the initial database connection (using the class 'DatabaseLink', see Figure 5.3), the program fetches the blueprints of the tables and represents them as classes. Next, these classes are stored as attributes in the database connection object. By instantiation, an entry (a row in the table) is created and can be used as any other object in Python. The attributes of that new object correspond to the values stored in the column of that table row.

For convenience, methods have been defined to create and query entities from the database ('create/query_[tablename]'). These methods have the same arguments as the corresponding table columns, except for the argument 'dictd', which is a Python dictionary of the arguments to provide a more simple way to interact. For example, to create a detector entity it is possible to call '`create_detector(name="test", pxsize=6, shape=[1024,1024])`' or '`create_detector(dictd={"name":"test", "pxsize":6, "shape":[1024,1024]})`'. The method 'update_file' is used to affix the WCS solution after solving. Despite these methods, the method 'get_session' is internally used, and is supposed to provide a direct database connection.

5.3.3 Astrometry Connection

The code used to communicate with the local astrometry service is based primarily on the work of Dustin Lang as part of his astrometry suite. Some functionalities have been extended (see listing 5.1), such as:

- The addition of the 'job_log' method to request solver logs to determine if the lack of a solution was due to a CPU time limit.
- The 'job_status' method was changed to return results instead of printing them to the console.
- The 'job_result' method was introduced to provide a shortcut to success or failed status.

However, efforts have been made to keep modifications to a minimum and maintain compatibility with the original code. This class implements an interface to the solver Web service. The web service is written in Python using the Django framework, and the astrometry client programmatically interacts with it.

```
def job_status(self, job_id, justdict=False): ### added justdict
    result = self.send_request('jobs/%s' % job_id)
    if justdict:  ### return the dict instead of printing it
        return result
    stat = result.get('status')
    if stat == 'success':
        result = self.send_request('jobs/%s/calibration' % job_id)
        print('Calibration:', result)
        result = self.send_request('jobs/%s/tags' % job_id)
        print('Tags:', result)
        result = self.send_request('jobs/%s/machine_tags' % job_id)
        print('Machine Tags:', result)
        result = self.send_request('jobs/%s/objects_in_field' % job_id)
        print('Objects in field:', result)
        result = self.send_request('jobs/%s/annotations' % job_id)
        print('Annotations:', result)
        result = self.send_request('jobs/%s/info' % job_id)
        print('Calibration:', result)
    return stat

def job_result(self, job_id):  ### Get job result
    # Own method
    result = self.send_request('jobs/%s/info' % job_id)
    return result

def job_log(self, job_id):  ### Get Solver log
    # Own method
    #result = self.send_request('joblog/%s' % job_id)
    rq = Request(url=self.apiurl.split('api/')[0]+'joblog/%s' % job_id)
    with urlopen(rq) as u:
        result = u.read()
```

```
32     return result
```

Listing 5.1: A cut out of the astrometry client. These are the methods that have been modified. Line 1 has a changed method signature, as the method jobs_status did just print the result and did not return anything of value. Method job_result is used to find out whether solving did work or failed. The method job_log obtains the solver log, by that it is determined if the solving time did excide its limit.

5.3.4 FITS Parser

The FITS parser utilizes an external configuration file that allows the creation of a mapping between the FITS header keys and the database columns. This configuration file (located at `settingsconf.py`, example in listing 5.2) enables flexible and customizable handling of FITS header data during parsing and database insertion. In addition to that, it utilizes the `astropy.io.fits` class in a read-only manner. It is ensured that the original data is not modified. The class consists of 5 methods. One of these methods ('parse_fitsfile') is responsible for extracting the plain header of the FITS file. The other methods ('`parse_[detector/file/filter/instrument]`')are dedicated to handling the data structure of the database. Each of these methods is designed to extract relevant information and store it in a dictionary that represents its corresponding table in the database.

```
1 file = {'type':{'KEY':['IMAGETYP']},...}
2 file['type']['B'] = ['bias Frame', 'zero','bias']
3 file['type']['D'] = ['dark frame', 'dark']
4 file['type']['F'] = ['flat field', 'flat']
5 file['type']['L'] = ['light frame', 'image', 'light', 'object']
```

Listing 5.2: The 'type' column in the database table file is filled by searching for the 'KEY' value 'IMAGETYP' (line 1). As the database only allows one character as a value for this column, 'B' is entered if the FITS header contains 'bias frame', 'zero', or 'bias' (line 2).

5.4 Astrometry Service

The astrometry solver is a Docker image based on the original Docker image of Dustin Lang. The modifications done in the extended image are mainly a quick and lightweight built-in database SQLite and a basic configuration of the web service so that it is not required to create an account to access the local nova.astrometry.net copy. In order to solve images, it is needed to supply precompiled catalogs. Due to sky coverage and precision, the software uses catalogs based on the Tycho, Gaia, and 2MASS surveys.

5.5 Docker Compose

Docker Compose is a powerful tool to deploy applications running in containers with Docker. It provides an easy way to connect multiple containers and manage their configuration in a single file. Docker Compose allows developers to define complex application environments consisting of different services, databases, and other dependencies, and easily create, start, and stop them. Using a clear and structured YAML-based syntax, it simplifies application automation and scaling. Docker Compose is an indispensable tool in the containerization of applications. In the following listing 5.3 one can find the exact configuration of the container setup.

```
version: '1.0'
services:
  django:
    image: "django-msc"
    ports:
      - "8080:8080"
    links:
      - database:db
    volumes:
      - ./djangoProject:/usr/src/app
  astrometry-fast:
    image: "dm90/astrometry"
    restart: on-failure
    ports:
      - "8000:8000"
    volumes:
      - ./astrometry:/usr/local/astrometry/data
      - ./settings_test_astrometrynet.py:/astrometry.net/net/settings_test.py
      - ./docker_astrometry/astrometry_fast.cfg:/astrometry.net/etc/astrometry.cfg:ro
      - ./docker_astrometry/astrometry_fast.cfg-dist:/astrometry.net/etc/astrometry.cfg-dist:ro
      - ./docker_astrometry/solve_script.sh:/astrometry.net/net/solve_script.sh
    links:
      - database:db
    profiles:
      - fastrun

  astrometry:
    image: "dm90/astrometry"
    restart: on-failure
    ports:
      - "8000:8000"
    volumes:
      - ./astrometry:/usr/local/astrometry/data
      - ./settings_test_astrometrynet.py:/astrometry.net/net/settings_test.py
      - ./docker_astrometry/astrometry.cfg:/astrometry.net/etc/astrometry.cfg:ro
```

```
36      - ./docker_astrometry/astrometry.cfg-dist:/astrometry.net/etc/
    astrometry.cfg-dist:ro
37      - ./docker_astrometry/solve_script.sh:/astrometry.net/net/
    solve_script.sh
38    links:
39      - database:db
40    profiles:
41      - production
42  database:
43    image: "mscdb"
44    #command: ["postgres", "-c", "log_statement=all", "-c", "
    log_destination=stderr"]
45    ports:
46      - "5432:5432"
47    volumes:
48      - ./postgres-data:/var/lib/postgresql/data
49    healthcheck:
50      test: ["CMD", "pg_isready" ,"-U" ,"postgres", "-d","mscdb"]
51      interval: 10s
52      timeout: 5s
53      retries: 5
54  pipeline-fast:
55    #image: 'mscparser'
56    image: 'testpipe'
57    depends_on:
58      astrometry:
59        condition: service_started
60      database: #try
61        condition: service_healthy
62    #restart:
63    links:
64      - astrometry-fast:astrometry
65      - database:db
66    volumes:
67      - ./PurePython:/usr/src/app
68    profiles:
69      - fastrun
70
71  pipeline:
72    #image: 'mscparser'
73    image: 'testpipe'
74    depends_on:
75      - astrometry
76      - database
77    #restart:
78    links:
79      - astrometry:astrometry
80      - database:db
81    volumes:
82      - ./PurePython:/usr/src/app
83    profiles:
```

```
84      - production
```

Listing 5.3: This listing states the configuration of the individual Docker images.

Understanding the structure The configuration is written in the YAML format. YAML files are widely used in configuration files, documentation, and other data serialization applications. They are human-readable and easy to write, making them a popular choice for data storage and interchange. YAML files are structured using indentation, with each level of indentation representing a nested data structure. Nested data structures can be of three basic kinds: scalars (which include text), sequences (similar to Python lists), and mappings (key-value pairs).

Understanding the configuration Docker lists the basic reference for configuring using Docker Compose online in their documentation[3]. The file starts its first indentation with 'services:', the next indentation level states services, i.e., containers or images which provide a service. These have a name which is only file-internal, e.g., 'django:' could be named 'web:' too. The next level lists a set of attributes:

- 'image': (scalar) name of the used Docker image.
- 'ports': (list of mappings) maps container port to host port, e.g., '8080:8080'. This results in the container port 8080 being published at the host on port 8080.
- 'links': (list of mappings) states which other containers can be accessed by this container. E.g., 'database:db' the service named 'database' can be accessed with the hostname (instead of an IP address) 'db' for this service.
- 'volumes': (list of mappings) the following key-value pairs are mounted in the container, i.e., host path:container path.
- 'restart': (scalar) decides how the container management deals with unexpected (to the management) shutdown. For example, the container stops due to an error or other reasons that are not known to the management.
- 'profiles': (list of mappings) one can define different container configurations. This is usually done to define testing and production environments. In this case it is used to implement the 'fastrun' (where solving time is limited to a few seconds) and default 'production' mode.
- 'depends_on': (list of mappings) in case the start-up sequence is important, it is possible to define which needs to be started before this service can be started.
- 'healthcheck': (list of mappings) defines when the service is ready to accept incoming connections. This is evaluated by the return code of a program in the container.
- 'build': (scalar) can be used instead of 'image', this string should point to the directory where special images are located, and should be built upon start-up.

[3] `https://docs.docker.com/compose/compose-file/`

5.5.1 Docker Image Customization

Docker images like ‘django-msc‘, ‘mscdb‘, and ‘mscparser‘ are customized because they cannot be conveniently configured using standard Docker images.

Special image: Pipeline The standard Docker Python image used for ‘mscparser‘ is based on Debian, which is a relatively heavy full-stack distribution that can result in excessive memory usage. Therefore, this special image is based on Alpine, which is super lightweight.

Special image: Django The standard Docker Django image does not include the libraries for the database connection. The requirements (located in ‘requirements.txt‘) are:

```
astropy[all]
sqlalchemy~=1.4.39
psycopg2-binary
numpy
```

Special image: Database The Docker image Postgres was extended to meet the setup criteria for this project. The container was created[4] using:

```
    docker run -d \
--name some-postgres \
-e POSTGRES_PASSWORD=mysecretpassword \
-e PGDATA=/var/lib/postgresql/data/pgdata \
-v /custom/mount:/var/lib/postgresql/data \
postgres
```

In order to get the database configured, it was necessary to setup the database users. This can be done in general with:

```
CREATE USER [username] with password '****';
GRANT [SELECT , UPDATE, DELETE, INSERT, USAGE, ALL] ON
ALL TABLES IN SCHEMA [public, views] TO [username];
```

[4] intructions from `https://hub.docker.com/_/postgres`

6 Results

The final software ecosystem is controlled by Docker Compose. All parts of the software run in individual Docker containers. This allows an easy selection of running services and maintenance. To start it, one navigates to the main project folder and types `docker compose up` into a shell. There exist two configurations in which software can run; there is the 'fastrun' and 'production' setup. In the fastrun mode, the execution time for image solving is limited to 15 seconds. This ensures that the huge backlog of 100,000 frames is processed in a reasonable time. Then there is also the production mode, where the time restriction is set to 300 seconds. Also, it is possible to access the service interfaces locally, e.g. to inspect errors or access the astrometry.net container to view the results of the astrometric solution, when it is launched in the fastrun configuration. In the production environment, these are disabled, as it should not be needed to access them at all, and enhances the system security. A study of 290 files revealed that the average time to solve the problem was 0.949 seconds, with an RSME of 6.360 seconds.

6.1 Web Interface

The web interface (see Figure 6.1) features a menu on the left side of the screen that outlines its main components. To the right of the menu is the content, which occupies most of the screen. The following sections explain the primary functions.

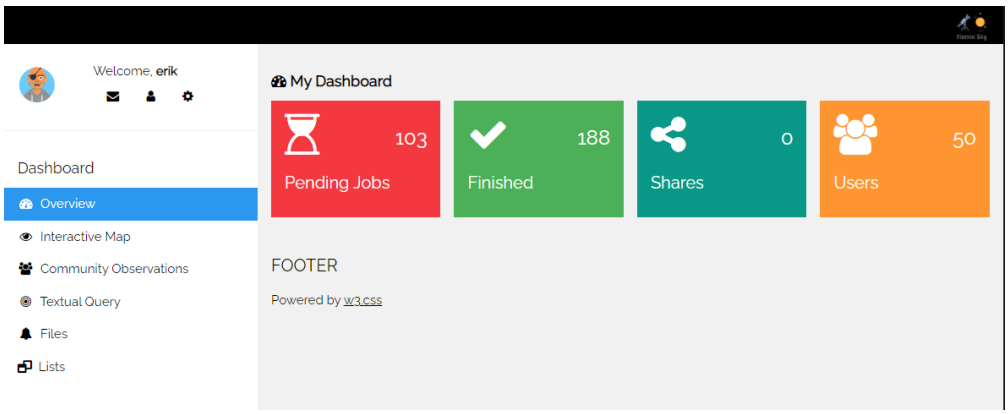

Figure 6.1: Here the general layout of the web interface is shown, in the left area (white background), the menu is displayed, and their functions are individually elaborated in the sections 6.1.1-6.1.6. In the grey area, the content of the current page is displayed.

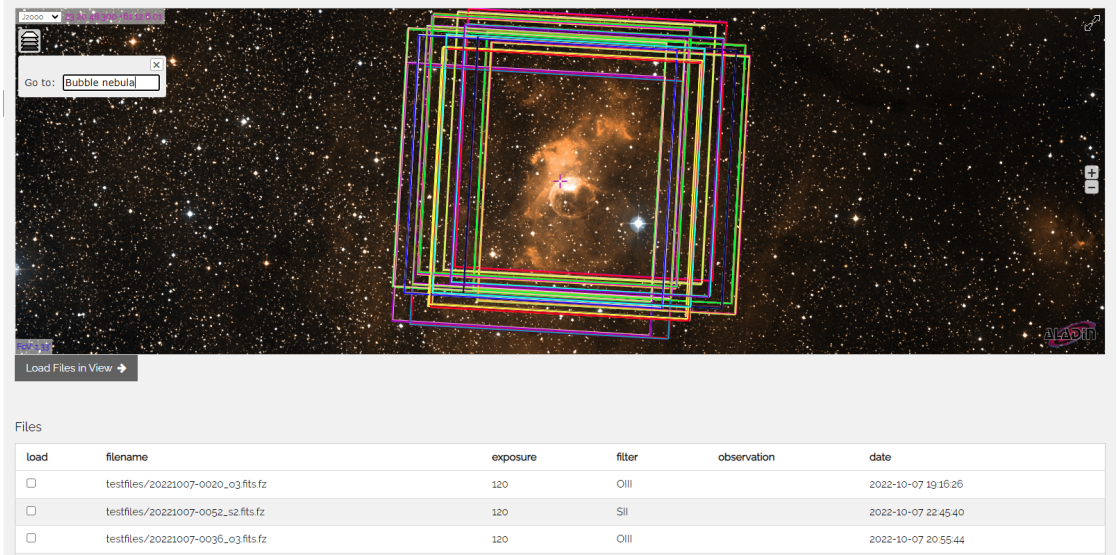

Figure 6.2: The sky map features a search field, frames that are in the current field of view are marked inside the map, and are listed in the table below. Currently, it was searched for 'Bubble Nebula' (NGC 7635).

6.1.1 Sky Map

The interactive skymap (see Figure 6.2) includes an Aladin lite frame that can be used to explore the sky. Different background images from various all-sky surveys, such as DSS2 or 2MASS, can be selected. The search bar allows users to type search terms or coordinates. When the viewed area is changed, frames that have at least one corner visible in the Aladin window are returned and displayed in the Aladin frame. This information is also added to the table below, as seen in figure 5.2. The table shows the most important information for each file shown in the Aladin frame. If the file is no longer visible due to a movement of the Aladin container, it is removed from the table. It will only remain in the table if it was selected before. Furthermore, it is possible to de-/select all frames currently visible. The table allows users to download the selected frames and, if possible, add pointing information. Pointing information is added only to individual frames and never removed. If the pointing information is already included in the file, it will not be removed. This is illustrated in figure 6.3.

6.1.2 Textual Query

On the 'Textual Query' site (see Figure 6.4) an image is shown that visualizes the actual database, this is meant as an orientation help. Below there is a text field that has syntax highlighting for SQL and is a raw database connection, it is possible to query all tables and it is possible to modify the tables `downloadkeys` and `dfiles`.

The purpose of this page is to enable specific queries that would not be possible with

Figure 6.3: Every file download offers the choice to include WCS information if it is absent. If the original option is chosen, the standard, unaltered file is obtained.

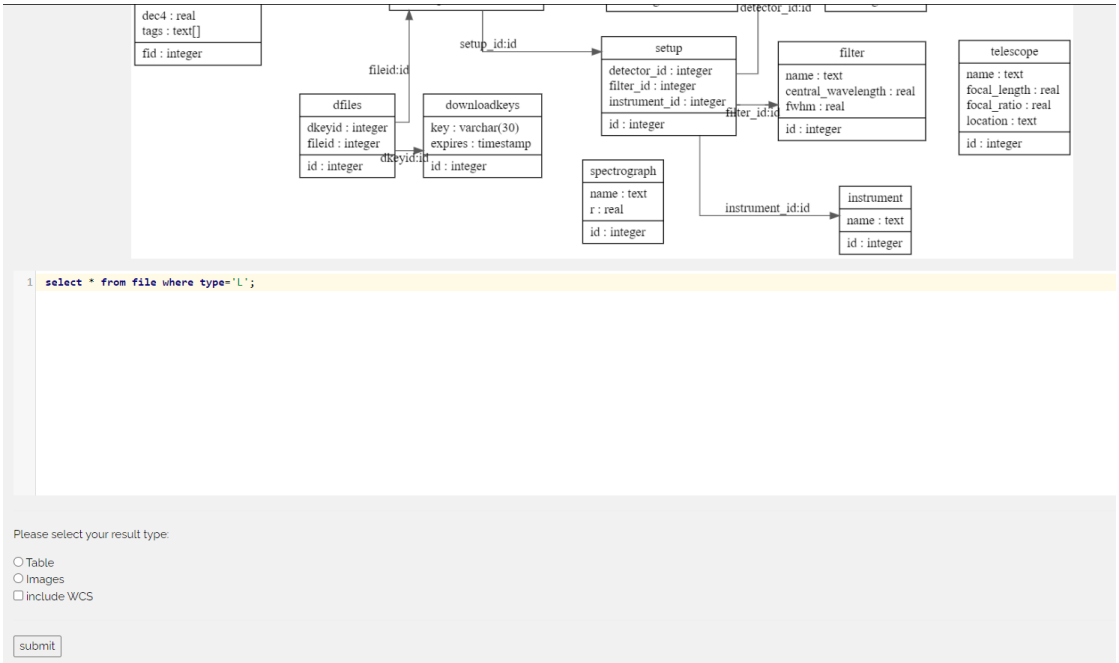

Figure 6.4: It is possible to enter instructions for complex requests using Structured Query Language (SQL). The Entity Relationship Diagram (ERD) above provides an orientation (full ERD in figure 5.1).

the sky map, however, ADQL (e.g. cone search) is not supported. For example, if one wants to investigate the evolution of dead pixels on the detector, in this case only bias frames would be of interest. One can obtain these frames by typing:

```
SELECT * from file where type="B";
```

The query reads (`SELECT`) the table `file`, the star (`*`) indicates all columns, as an example it is possible to write just columns of interest too (`SELECT id, path, type ...`), the condition (`where`) indicates which rows should be returned, in the example above, the column 'type' must be equal to `'B'`. The statement ends with a semicolon (`;`). Tables, columns, and data types[1] can be found in Figure 5.1.

Return type Below the query field, it is possible to choose the kind of result, one type is a table, which states all requested information, and the other option is the image kind, with this option it is possible to download all images that match the criteria. For the image type to function properly, the file table and the primary key (`file.id`) must be selected.

More complex queries Often it is not possible to query all informations from just one table, e.g. it is not possible to select the name for a filter using just the table file, therefore it is possible to make 'joins'. These do extend the table horizontally (adding columns) by attributes of other columns[2].

```
SELECT * FROM file JOIN lframe_info on file.id = lframe_info.fid;
```

This query joins the table file with lframe_info, the condition which columns must be matched is `file.id = lframe_info.fid`. However, when this query is executed only rows that can be matched are returned, as only science images have coordinates stored in lframe_info all other frame types (and those which do not have coordinates) are discarded. This phenomena can be changed by replacing `JOIN` with `LEFT OUTER JOIN`, which keeps all rows from file and fills the columns of lframe_info with NULL in unmatched cases.

Below is an example of how to select all flat field frames with the filter name 'B'.

```
select * from file AS T1
 join setup AS T2 on T1.setup_id = T2.id
 join filter AS T3 on T2.filter_id = T3.id
where type='F' and T3.name = 'B';
```

The query uses aliases, as indicated by 'AS'. Aliases should be used in complex queries. To build such a query, refer to the ERD in Figure 5.1. Identify the tables and columns to return, starting with the 'file' table, moving to 'setup' (following the arrows), and then to 'filter'. The conditions for joins are stated on the connecting arrows, e.g., the arrow from 'file' to 'setup' is labeled 'setup_id:id', which means that 'file.setup_id' is connected to 'setup.id' respectively.

[1] Definitions of datatypes can be found at `https://www.postgresql.org/docs/current/datatype.html`
[2] More about joins at `https://www.postgresql.org/docs/16/queries-table-expressions.html`

Figure 6.5: In the "Lists" section, one can find the pre-defined queries. These queries can be created with the text query interface using the "Create View" command.

6.1.3 Predefined Queries

In the rubric lists it is possible to visualize pre-defined queries. This is of use if one wants to do the same query several times, i.e. list all files and see if any classification errors occurred like a file type of 'U'. To create a new view, one needs to wrap the query in a simple context.

```
CREATE OR REPLACE VIEW views.name_of_view AS
    The query you want to use often
```

For instance, if you want to select the newest 50 files one could do this by:

```
CREATE OR REPLACE VIEW views.recent_files AS
    Select * from files order by (date) DESC LIMIT 50;
```

The important constraint here is that the name of the view needs to start with '`views.`' otherwise it will not work due to lack of permissions. To see the outcome of the query, go to the 'list' section and click on the link in the result table that displays the given name (like in figure 6.5).

6.1.4 Files

The menu entry 'Files' lists all entities of the table `file`. The list is sorted by the field id, and supports pagination, which is currently set to 100 entities per page.

6.1.5 User Management

Certain features require more privileged user rights, e.g. the download functionality or the modification of attributes in the database. Django provides a user management system that enables the concealment of potentially intrusive activities behind a login page. A so-called superuser is created. With that superuser, it is possible to create more users, for example, a student user, which has permission to download files, whereas the superuser is even allowed to change values in the database tables.

Frames

id	path	type	ra dec	tags	filter	exptime	date
800	testfiles/20221007-0012_dark.fits.fz	D	None		None	120,0	7. Oktober 2022 16:45
802	testfiles/20221007-0015_flat_ha.fits.fz	F	None		Halpha	6,0	7. Oktober 2022 16:23
473	testfiles/20221007-0041_o3.fits.fz	L	None		OIII	120,0	7. Oktober 2022 21:31
124	testfiles/20210615T192641.0.pico.object.fits	L	None		Bessel R	0,1000032	15. Juni 2021 19:26
804	testfiles/20221007-0011_dark.fits.fz	D	None		None	120,0	7. Oktober 2022 16:42
806	testfiles/20221007-0002_flat_s2.fits.fz	F	None		SII	5,0	7. Oktober 2022 16:20
808	testfiles/20221007-0002_flat_o3.fits.fz	F	None		OIII	5,0	7. Oktober 2022 16:25
810	testfiles/20221007-0030_flat_ha.fits.fz	F	None		Halpha	15,0	7. Oktober 2022 16:34
129	testfiles/20210428_-0001Flat_B.fits	F	None		B	20,0	28. April 2021 19:37
812	testfiles/20221007-0023_flat_s2.fits.fz	F	None		SII	10,0	7. Oktober 2022 16:31
814	testfiles/20221007-0004_dark.fits.fz	D	None		None	120,0	7. Oktober 2022 16:04

Figure 6.6: In the "Files" section, one can find a list of all `file` entities in a format that is easy for humans to read. Instead of showing the filter id, the name is shown.

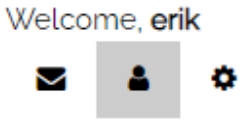

Figure 6.7: The gray rectangle in the top bar containing the user icon hides the admin interface and user management.

6.1.6 Changing Data

In the event that something goes wrong during an observation, such as mistakenly setting the frame type to 'flat' instead of 'image', the pipeline might not be able to detect this error. However, it is possible to rectify this situation through the admin interface.

In the event that 'file 914' has been incorrectly labeled as a flat frame when it is actually a light frame, one can go to the admin page (Figure 6.7), select the 'Files' option (Figure 6.8, top panel), and then click on the file that needs to be edited ('File object(914)', Figure 6.8, bottom panel). In the detailed file listing for entity ID 914 (Figure 6.9), it is possible to change the type from 'F' (flat frame) to 'L' (light frame). This technique can be employed for other modifications, for instance, to incorporate WCS data into a file or to alter the validity of a downloadkey.

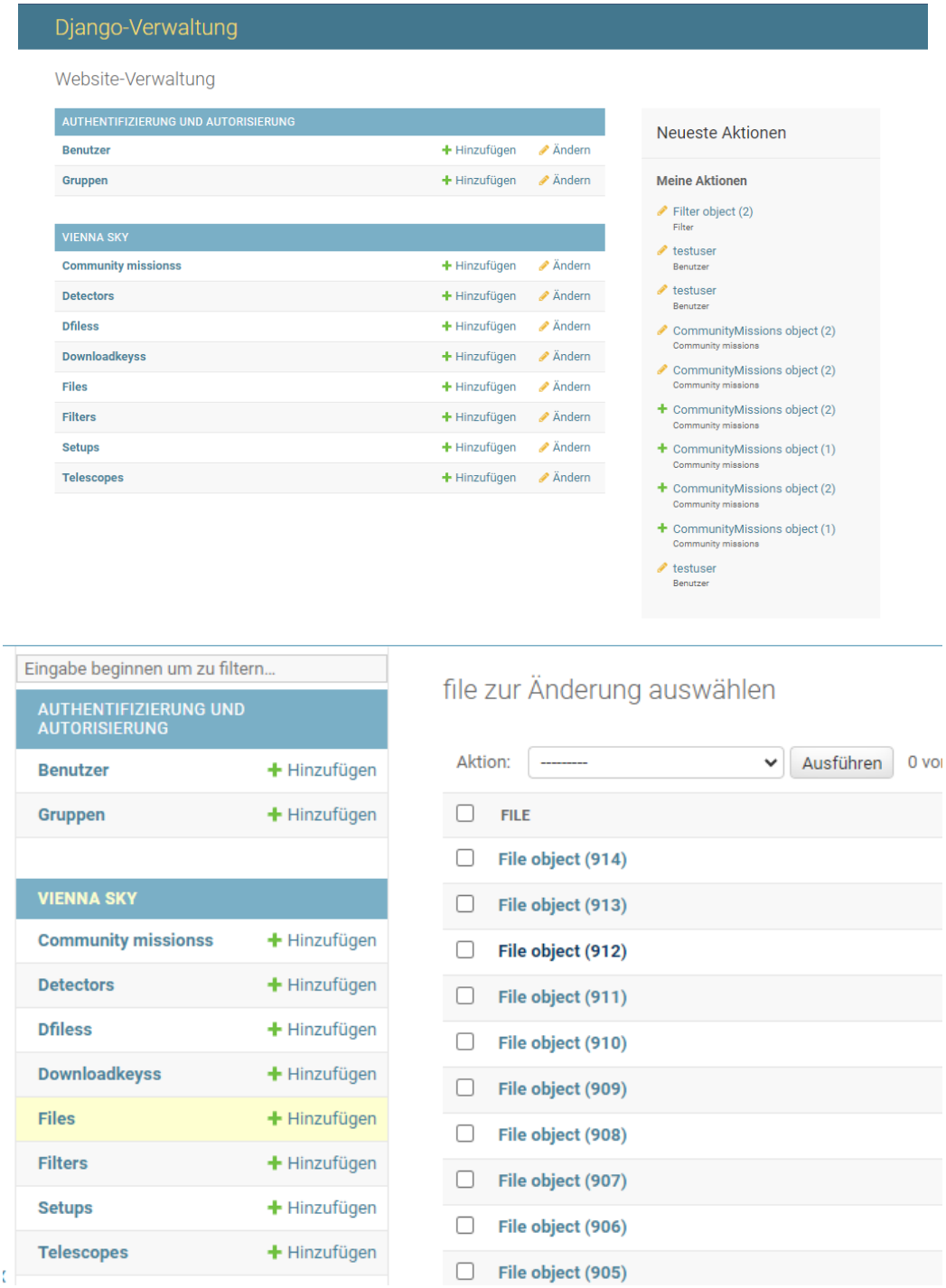

Figure 6.8: The administrative interface allows for the alteration of entities in all database tables. Top panel: The tables on the left can be accessed by clicking on their names, which will allow you to edit the rows. Bottom panel: This list displays row objects, with the primary key (in this case, `file.id`) in brackets.

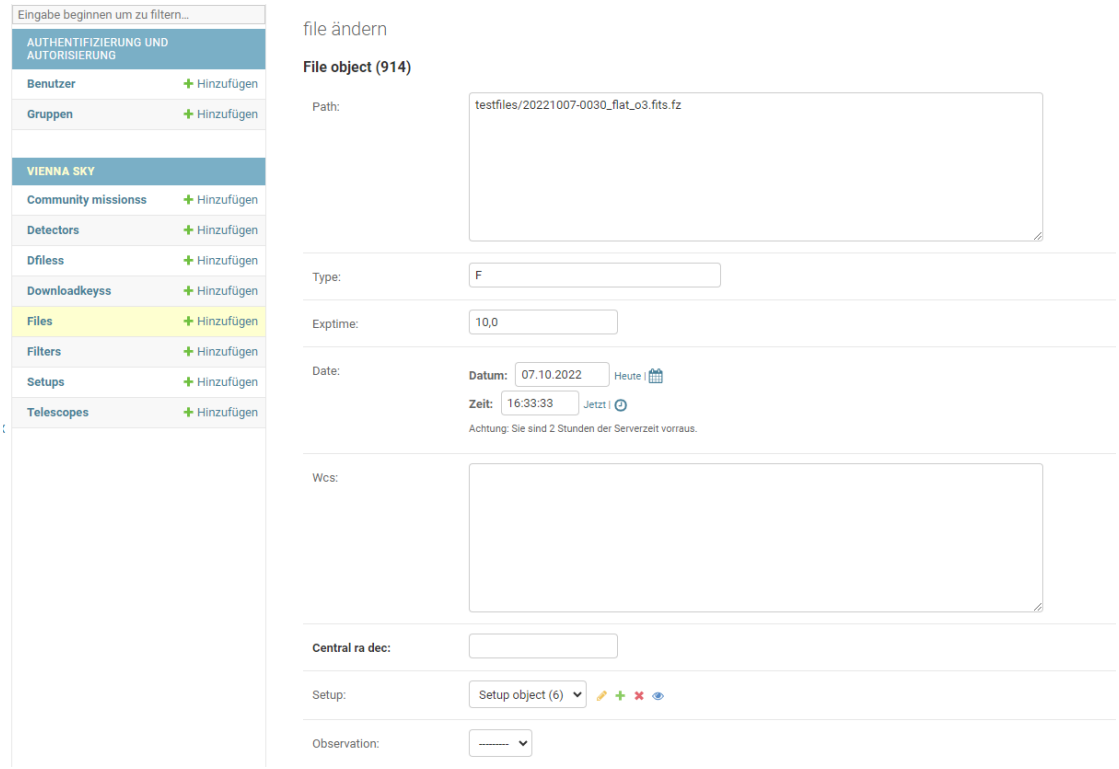

Figure 6.9: It is possible to make alterations to all the columns of the table entity during the detailed editing process. For example, changing the type from 'F' to 'L' would convert the file from a flat frame to a light frame in the database.

7 Summary

A key aspect of data management in modern observatories is the administration and distribution of the collected data. In Austria, the two telescopes operated by the University of Vienna, the Vienna Little Telescope (vlt) at the Institute for Astrophysics and the Leopold Figl Observatory (LFO), collect data for scientific use. Currently, these data are stored on the respective computers that control the telescopes or on the user's computer, and it does not migrate to a central collection point, relying on manual organization. Unfortunately, there is no register that lists when, what, or how the observations were made. Furthermore, the vlt does not embed pointing information in the images, which makes it difficult to automatically archive the data. As a result, it has become customary to take manual notes in an observation room notebook during measurements. Although these notes are informative, they cannot be easily extracted by automated means. Moreover, measurements are not publicly accessible because the data is stored on the measurement computer, and it is not desirable for untrained individuals to access them.

Introduction of the archive Given these circumstances, retrieving and utilizing the data has proven to be challenging, which has amplified the call for a centralized collection of the data. Steps taken to remedy the issues:

- A database was designed to hold information on metadata related to individual FITS images (see Section 5.1). The database was created using PostgreSQL.
- The nova.astrometry.net service was deployed locally, so that networking and computing resources are available on demand (see Sections 4.1.2, 5.4).
- A pipeline (see Section 5.3) was developed that automatically extracts metadata from measurements and aggregates them into a suitable format to be inserted into the database. In addition, missing pointing information is generated using the local Astrometry.net suite.
- A web service was deployed using Django, which features an interactive sky map that allows for an intuitive selection of images (see Section 6.1.1).
- The web service provides a textual interface to the database, where the end user can select data using SQL (see Section 6.1.2).
- Docker Compose was used to keep the overhead of hosting these services at a minimum (see Section 5.5).

7 Summary

The new archive constructed during this thesis unlocks a wealth of data for astrophysical research. It not only aids ongoing projects within the IfA, but also positions our institute as a valuable contributor to global astronomical endeavors. By adhering to a modular approach in software engineering, it is possible to expand this project with additional features in the future. This makes it possible to incorporate and combine further data products.

Conclusion This master's thesis addressed the urgent requirement for a modern data archive at the Vienna Institute for Astrophysics. By constructing a new archive with automated cataloging, coordinate appending, and a user-friendly web interface, I have increased the impact of IfA and created a basis for data reuse.

Bibliography

Bertin, E. 2006, in Astronomical Society of the Pacific Conference Series, Vol. 351, Astronomical Data Analysis Software and Systems XV, ed. C. Gabriel, C. Arviset, D. Ponz, & S. Enrique, 112

Boch, T. & Fernique, P. 2014, in Astronomical Society of the Pacific Conference Series, Vol. 485, Astronomical Data Analysis Software and Systems XXIII, ed. N. Manset & P. Forshay, 277

Django Software Foundation. 2022, Django, `https://djangoproject.com`

Fritzewski, D. J., Barnes, S. A., Weingrill, J., et al. 2023, , 674, A152

Galiazzo, M. A. & Zeilinger, W. W. 2013, Journal of the Association of Lunar and Planetary Observers, the Strolling Astronomer, 55, 17

Greisen, E. W. & Calabretta, M. R. 2002, , 395, 1061

Handler, G. & Meingast, S. 2011, , 533, A70

Hanisch, R. & Wells, D. 1988, World Coordinate Systems Representations Within the FITS Format, `https://lweb.cfa.harvard.edu/~jzhao/SMA-FITS-CASA/docs/wcs88.pdf`

Hürkal, D. Ö., Handler, G., Steininger, B. A., & Reed, M. D. 2005, in Astronomical Society of the Pacific Conference Series, Vol. 334, 14th European Workshop on White Dwarfs, ed. D. Koester & S. Moehler, 577

Kolenberg, K., Guggenberger, E., Lenz, P., et al. 2005, Communications in Asteroseismology, 146, 11

Lang, D., Hogg, D. W., Mierle, K., Blanton, M., & Roweis, S. 2010, , 139, 1782

N.I.N.A. 2022, Plate solving, `https://nighttime-imaging.eu/docs/master/site/advanced/platesolving/`

Paunzen, E., Handler, G., Lendl, M., et al. 2017, , 52, 133

Ponz, J. D., Thompson, R. W., & Munoz, J. R. 1994, , 105, 53

Puschnig, J., Posch, T., & Uttenthaler, S. 2014, , 139, 64

Taris, F., Damljanovic, G., Andrei, A., et al. 2018, , 611, A52

Uttenthaler, S., Meingast, S., Lebzelter, T., et al. 2016, , 585, A145

Wells, D. C., Greisen, E. W., & Harten, R. H. 1981, , 44, 363