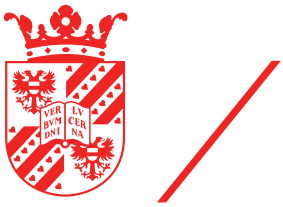
University of Groningen

Bachelor Thesis

Creating the Blaauw Dashboard

A new way to interact with the Blaauw Archive



university of groningen

faculty of science and engineering

kapteyn astronomical institute

Author:
Heine Jan Lindemulder

Supervisor:
Dr. Jake Noel-Storr
Second examiner:
Prof. dr. Marc Verheijen

Abstract

Currently, getting to and inspecting the data generated by the Blaauw telescope can be rather confusing. One has to go through many folders before reaching the data and download the data before being able to inspect it. In this thesis we will go through the process of creating a new application to interact with the data in the Blaauw archive: the Blaauw Dashboard and is available on GitHub[1]. The Blaauw Dashboard hooks in to the Blaauw archive using a TAP service, created by Sipma (2021). With the Blaauw Dashboard, users will be able to search the whole archive and immediately inspect the data they want to. Users can search for data in a specific time frame and for data containing a certain object or set of coordinates. We will do this by using Python and some of its libraries, which allow us to create an interactive web-based application that users can use without needing to have any knowledge of the archive, Python or SQL.

[1] `https://github.com/PracticalAstronomyCrew/Blaauw_Dashboard`

Acknowledgements

I would like to thank Jake Noel-Storr, who was my supervisor for this project. I came to Jake wanting to do a 'practical' thesis about 'something with data' and they really delivered. I really enjoyed creating the Blaauw Dashboard and it has made me even more certain about what I want to do next. Furthermore, I would like to thank Marc Verheijen for being the second examiner. I would also like to thank Sten and Fabian for giving me an introduction to the Blaauw Archive and how to communicate with it. Finally, I would like to thank Ilse for her support and for making the last few months much more fun.

Contents

Abstract . . . 2
Acknowledgements . . . 3

1 Introduction **5**
1.1 Problem . . . 5
1.2 Goal . . . 5
1.3 Outline . . . 6

2 Astronomical data in the Blaauw archive **7**
2.1 About the archive . . . 7
2.2 Infrastructure . . . 7
2.3 Problems with the data . . . 8

3 Tools **9**
3.1 Application Creation Tools . . . 9
3.2 ADQL . . . 9
3.3 Python Libraries . . . 11
3.3.1 PyVO . . . 11
3.3.2 PyPika . . . 11
3.3.3 Panel . . . 12
3.3.4 Param . . . 13
3.3.5 astropy . . . 14
3.3.6 Astroquery . . . 15
3.3.7 Matplotlib . . . 15
3.3.8 pandas . . . 17
3.3.9 NumPy . . . 17

4 Process **18**

5 Program showcase **20**
5.1 Querying the database . . . 20
5.2 Plotting the data . . . 23
5.3 Nightly statistics . . . 24

6 Conclusion **26**
6.1 Solved and unsolved problems . . . 26
6.2 For the future . . . 27

7 References **28**

8 Appendix **30**
8.1 User guide . . . 30
8.1.1 Introduction to searching the database . . . 30
8.1.2 Plotting data . . . 33
8.1.3 Nightly statistics . . . 34

Chapter 1

Introduction

1.1 Problem

Since 2008 the Blaauw Observatory sits atop the Bernoulliborg at the Zernike campus. It houses the Gratama Telescope inside the 6.5 m diameter dome, which is a 40 cm reflector telescope of the Ritchey-Chrétien type (University of Groningen 2011). The observatory is sometimes opened to the public, but it is mostly a facility for astronomy students. With this telescope the students get their first taste of observing in a professional manner, which is a valuable skill for an astronomer.

Whenever someone is observing, the images taken with the telescope are automatically stored at the Blaauw archive, a server at the Kapteyn Institute. After the observation, one can then access those files from their own computer or through one of the Kapteyn computers, but this is often not very user-friendly. One way of getting to the files is using the terminal or an FTP-program, such as FileZilla, but then you must first work your way through the labyrinth of folders, download the files to your own storage and then the images can be inspected. Another method of accessing the data is through the Blaauw Observatory data archive. The archive was set up by Sipma (2021), while Gunnink (2021) has set up a pipeline, which automatically corrects data. This is already quite a bit easier, but you cannot filter or sort the images and you cannot download a whole folder at once, which could be done with the previous method. So, while both of these options work, it is not ideal and it can be confusing, especially for new users.

Then, there is also the problem of the weather in Groningen. When you are going to observe, you want a clear sky, but this is not always the case here in Groningen. Because of this, users can experience delays in their projects, because they do not yet have data. A solution is for them to go through old data and use that, but if they want data on a specific object, they will have to go through all the observation logs and find the night the object was observed.

1.2 Goal

What we at Kapteyn would like, is to have an application which can deal with the aforementioned problems, while also being user-friendly. To try and achieve this, we are creating the Blaauw Dashboard. The Blaauw Dashboard will be a web-based application which aims to solve many of the problems students and researchers may have in accessing and inspecting their data and tries to make it easier. To use the dashboard, no knowledge about databases or the query language used to communicate with databases is needed. This ensures that everyone can use the application to find what they are looking for.

We want users to be able to set filters, with which they can request the data that they are looking for. They should be able to have a good overview of the data that is returned, so they know what they are looking at and if it is what they need. We also want them to be able to inspect the data the server returned, by plotting it. This should allow users to have a first look at what their images look like and if they will be useful. Finally, we want to display some basic information on the calibration files for a specific night chosen by the user. This should allow them to see if their calibration can be used or if they should look for calibration files for other nights.

All the while, the dashboard should also be easy to use and intuitive. We will provide a user guide, but users should not have to look at the user guide each time they want to use the application. We also want users with color blindness to be able to use the application, so we have to make sure that we use colors that can be distinguished by anyone.

1.3 Outline

In this thesis we will go through the complete process of creating the application and talk about what more could be done. We will first discuss the astronomical data in the Blaauw Archive and how to communicate with it. This is very important, because without being able to communicate with the archive, the program will not work. After this we will talk about all the different tools we used. The application is written in Python and is heavily dependent on quite a few of the Python libraries.

Once we know what problems we want to solve and what tools to use, we will go through the creation process. We will talk about what we did, how we did it and why we did it, to give an insight into the complete process. Afterwards, all of the features will be shown, how they can be used and why we think those features are useful. Finally, we will conclude the thesis by checking if all of the problems were solved and what could be done in the future.

Chapter 2

Astronomical data in the Blaauw archive

The Blaauw Archive houses all the data that is gathered during observations. Within it, users can find the images they took and download them. In this chapter, we will give some general information about the archive, talk about the infrastructure and discuss some of the problematic data.

2.1 About the archive

To retrieve data from the server, the application needs to have a way to communicate with it. To achieve this, the Blaauw Archive has been tied into the Virtual Observatory (VO). The VO is a way for astronomers to connect to the data of other observatories, to try and make the boundaries between all the data sets of different observatories disappear. The standard upon which the VO is built, is created and upheld by the International Virtual Observatory Alliance (IVOA) (International Virtual Observatory Alliance 2022).

For an observatory to be tied into the VO, it has to adhere to the standards that are set by the IVOA. Within the IVOA, the standards for accessing data of a Virtual Observatory are decided by the Data Access Layer (DAL) working group. This working group has defined the Data Access Layer Interface (DALI), which consists of the base requirements all Virtual Observatories have to adhere to (Dowler, Demleitner, et al. 2017). The DALI can then be extended using other standards, such as the Table Access Protocol (TAP), which we will use for the dashboard (Dowler, Rixon, et al. 2019). The TAP is a fairly general standard and is there for tabular data. There are other standards for more specific types of data, such as spectral or line data, but those are not relevant for our application.

For some of the files in the archive, there is also an Astrometry version of the file available, which was done using the Astrometry API (Lang et al. 2010). With this API it is possible to fit a World Coordinate System (WCS) to a FITS file, which often results in better coordinates. In our dashboard we will clearly mark these files, because it is often best to use these files or at least inspect them to see if they could be more useful than the files without a WCS.

2.2 Infrastructure

During observations, after the CCD has read out all the pixels, two things happen: the image is transferred to the server and an entry of the observation is created in a database. This was set up by Sipma (2021)[1]. With the file on the server, users can find their image, download it and start working on it. Having the image on the server is of course very important for the dashboard, but having an entry of the file in the database is just as important. The database entry contains a unique ID, the filename, right ascension and declination where the telescope was pointing at, the information contained in the FITS file header and much more. This is critical for the dashboard, because instead of having to look through all the folders, opening the image and checking the headers, the program can just use the information from the database. Without this information, the Blaauw Dashboard just could not work.

[1] `https://github.com/PracticalAstronomyCrew/blaauw-archive`

Currently, this is where the road ends for the observation data, but Gunnink (2021), has created a pipeline which automatically reduces the data[2], but it has yet to be implemented. The reduced data is then stored on the server and an entry is made in another database, which is also maintained by Sipma (2021). For now, the data pipeline and second database are not yet up and running, so the dashboard cannot hook into it and only the raw observation data can be viewed.

2.3 Problems with the data

At the moment the `observations.raw` table has 52,305 entries and 43,738 of those are light frames. While all entries are useful, many of them are missing information. For example, there are 31,033 light frame entries that do not have a declination or right ascension specified. That means that almost 71% of all light frames in this table cannot be used for something like a box, cone or coordinate search and can thus not be used when searching for specific objects or coordinates.

We also have to take the coordinates of the telescope itself into account. Groningen lies at about 53 degrees north, which means that we cannot observe objects with a declination of $53 - 90 = -37$ degrees. Of all the light frames with a declination, only ten of them have a declination -37 degrees or lower, which is only about 0.08%, so this is a negligible amount, luckily.

Some of the files have had a WCS fit to them, using Astrometry and this is indicated by the text 'astrom' in their filename. Out of all the files with a right ascension and declination, 6,591 of them have the text 'astrom' in their filename and normally they should be preferred over the files without 'astrom' in their filename. Unfortunately, Astrometry is not perfect and it can sometimes make mistakes, which can result in an abnormal plate scale. This was also looked at by Sipma (2021) in one of his examples for the archive[3]. We can see this in Table 2.1, which shows the minimum, median and maximum for all three binnings.

Table 2.1: This table shows how a wrong Astrometry fit can result in skewed results. Some spread is expected, but the minimum and maximum for both the 1x1 and 3x3 binning is well outside the normal expectations.

Plate Scale			
	Minimum	Median	Maximum
1x1 binning	0.178	0.566	17.774
2x2 binning	0.821	1.132	1.137
3x3 binning	0.526	1.698	17.468

In the table we can clearly see a very large spread in the plate scale for the 1x1 and 3x3 binning. A small spread is expected, but the results for these two binnings clearly show that the Astrometry fits for these files are not correct. Luckily, values that are close to these minima and maxima are quite rare. So, while Astrometry files are generally preferred, it is important to not just trust whatever it says and know what to expect so one can judge if the file should be used or not.

[2] `https://github.com/PracticalAstronomyCrew/BlaauwPipe`
[3] `https://github.com/PracticalAstronomyCrew/blaauw-archive`

Chapter 3

Tools

3.1 Application Creation Tools

There are many great options one can use for creating applications, so before deciding on one tool, we explored multiple routes. In the end we came up with two different options we thought might work: Qt and Python with its many libraries.

Qt is a development framework with which you can create applications for desktop, mobile and embedded devices (The Qt Company 2019). The applications can be written using either C++ or Python. Qt is a very well known and established framework, so there is lots of documentation for it and has proven its worth, but its implementation for creating web-applications is not yet finished.

The other option is Python, which is a general-purpose programming language, created by Guido van Rossum in the 90s (Python Software Foundation 2019b). Python is relatively easy to program with and has many libraries which add a lot of functionality to the language. Even though it is generally slower than low-level languages, like C/C++, it is easier to create programs with and it has many libraries which help with manipulating (astronomical) data. Although it is inherently not a web-application programming language, like JavaScript, it does have libraries that make this possible.

In the end, we chose to use Python and its libraries, instead of Qt. Even though we could create applications in Qt with Python, we want to create a web-application and Qt is just not quite ready for this. There are also many great Python libraries for interacting with astronomical data, communicating with databases and plotting images. Furthermore, we have experience from previous courses with Python and some of its packages, which should allow us to create a better application.

3.2 ADQL

Before we talk about Python and the libraries that we will use, we will first discuss ADQL, which is the query language specified by the International Virtual Observatory Alliance (IVOA) and is needed to communicate with the Blaauw TAP service (Ortiz et al. 2008; International Virtual Observatory Alliance 2022; Dowler, Rixon, et al. 2019). ADQL is a dialect of SQL (Structured Query Language), which is a programming language used to communicate with relational databases. SQL was developed in the 70s by Donald Chamberlin and Raymond Boyce and was standardized in 1986 by the American National Standards Institute (ANSI) (Chamberlin 2012; Silberschatz, Korth, and Sudarshan 2010).

ADQL is based on the 1992 version of SQL, but is not a complete extension, because it uses only a subset of SQL-92 Ortiz et al. 2008. The advantage of using SQL as a basis is that it is used by many people, so to use ADQL, people need only be introduced to the ADQL specific functions. The functions specified by ADQL are of mathematical and geometrical nature. For example, it can convert angles between degrees and radians, calculate the sine of an angle, it has the constant π and many more functions.

To see how these functions can be used, we first have to look at the query syntax for ADQL. This can be seen in Figure 3.1 below.

```
SELECT [ ALL |  DISTINCT ]
      [ TOP unsigned_integer ]
      { * | { value_expression [ [AS] column_name ] }, …  }
      FROM {
       { table_name [ [AS] identifier ]      |
         ( SELECT ….) [ [AS] identifier ]  |
         table_name [NATURAL] [ INNER | { LEFT | RIGHT | FULL
           [OUTER] } ] JOIN table_name
           [ON search_condition | USING ( column_name,…) ] }
      , …}
      [ WHERE search_condition ]
      [ GROUP BY column_name, … ]
      [ HAVING search_condition  ]
      [ ORDER BY { column_name | unsigned_integer } [ ASC | DESC
          ],…]
```

Figure 3.1: This figure shows how a very general query is built up using ADQL. This is much more complex than we will use in our dashboard, because we will only use the `SELECT`, `TOP`, `FROM`, `WHERE` and `ORDER BY` clauses. Taken from Ortiz et al. 2008.

The figure shows the most general query that we can make with ADQL and we can make very complex and precise queries using all these clauses. We will not use all of them and we will only discuss the ones we do use. In the `SELECT` clause, we can select what columns we want the query to return. We could choose to return all columns, but we do not need all that information and handling all the columns takes longer, so we always specify the columns we want. With the `TOP` clause we specify how many entries we want to have returned. Using the `FROM` clause we specify what table we want to look in. Currently, the dashboard uses only one table, because the other tables are not yet available, but the dashboard could be easily extended to use those as well.

Then we have the `WHERE` clause, where we can use Boolean expressions to filter the data. This is where we can use the mathematical and geometrical functions, which we will talk about in the next paragraph. Finally, we have the `ORDER BY` clause. With this clause we can specify a column by which we want to order the data, for example, by observation date.

As said, the geometrical functions are used in the `WHERE` clause. There are three different topics which the geometrical functions can be put in: data type, predicate and utility functions (Ortiz et al. 2008). We will only use the data type and predicate functions. With the data type functions, we can specify what type of geometry we want to look at, such as a box, point, circle or polygon. We can combine these with the predicate functions: `CONTAINS` and `INTERSECT`.

The syntax for these functions is as follows: `SELECT * FROM <table_name>`
`WHERE CONTAINS(<data_type>, <data_type>)=1`. Here we have used the `CONTAINS` predicate functions, which checks if the first data type is contained in the seconds data type and returns 1 if this is true and 0 otherwise, which is why we have to add `=1` at the end. In Chapter 5 we will look at how this can be used in our dashboard.

3.3 Python Libraries

As said, Python has a great amount of useful libraries, most of which are available at PyPI; a package manager for Python (Python Software Foundation 2019a). All of the packages we used were available at PyPI and we will now discuss all of them. Some we used much more than others, but we needed all of them to create the dashboard.

3.3.1 PyVO

The database at Kapteyn has been set up in such a way that is accessible through the TAP service, created by the IVOA (Dowler, Rixon, et al. 2019; International Virtual Observatory Alliance 2022). To connect with the database through the TAP service, we used PyVO; a Python package which allows you to retrieve data from databases accessible through IVOA standards, of which TAP is one (Becker et al. 2022). PyVO is a package affiliated with astropy, created and maintained by the Astropy Project, from which we will use more packages (The Astropy Collaboration, Robitaille, et al. 2013; The Astropy Collaboration, Price-Whelan, et al. 2018).

With PyVO the dashboard can fetch entries from the database through the use of queries using ADQL (Ortiz et al. 2008). We only need the URL to the TAP service of the Blaauw Archive[1] and then we can query the database. Once the query is executed, it returns all of the entries it could find, which can then be used by the dashboard. These entries can be transformed to different formats for ease of use, such as a pandas DataFrame, which we will talk about later.

To execute queries, PyVO uses the Data Access Layer (DAL), which is a collective name for all kinds of IVOA data access standards. The most important one of these is the Data Access Layer Interface (DALI), which specifies a set of requirements all other data access standards have to adhere to (Dowler, Demleitner, et al. 2017). The query is generated by PyPika (see Section 3.3.2) and gets passed to PyVO. This will check if the query adheres to the standards of the server we are trying to access and if the query is correct, it will run the query through the TAP service and return the result. The result gets transformed to an astropy Table object (see Section 3.3.5), which in turn gets transformed to a pandas DataFrame (see Section 3.3.8), because that is very easy to manipulate and display.

3.3.2 PyPika

PyVO is the way to communicate with the database, but it needs a query to work. A very basic query could look like this: `SELECT TOP 100 * FROM observations.raw`, where the ADQL exclusive `TOP` clause is used and `observations.raw` is the name of the table we want to query. This query will return a list of 100 entries from the 'observations.raw' table and will show all the available columns. This works and is an easy query, but users will want to be much more specific in the entries they want to see, so we need queries that are more complex. We also want everyone to be able to use it, so people should not even have to enter a query manually.

PyPika is the perfect package for this, because it is a Python query builder, which means that it allows one to create dynamic queries based on input by a user (PyPika 2022). It allows the use of Boolean logic for filtering data, such as AND and OR, and you can chain these options together. It is also possible to create functions yourself, so it is a very flexible package and is more than good enough for what is needed.

To create the queries with PyPika, the query will have to be split up as shown in Figure 3.2 below.

[1] `https://vo.astro.rug.nl/tap`

Figure 3.2: These are the different clauses needed by PyPika to compile a query. The boxes with the solid lines show parts of the query that are always needed. The thick-dashed box is generally not used in SQL, but it is compulsory for ADQL. The dotted boxes show optional parts of the query. Queries are not limited to these options, but this is all we will use for the dashboard.

A query will always need at least a `SELECT` and `FROM` clause. With `SELECT` you can specify what columns have to be returned or if you simply want to see all columns. Whit `FROM` you can specify what table you want the query to act on. Because we work with ADQL, we also have to always include a `TOP` clause with our query. This ensures that we do not request thousands of entries, which might slow the system down.

Sadly, SQL is not a standardized language and there are many different dialects and only a few of those are supported by PyPika. ADQL is not one of the supported dialects and thus we had to extend PyPika to make it work with ADQL. This was done by using some of the base classes and extending those, which allowed us to add ADQL support. This gives us the ability to use the `TOP` clause, which was not generally supported by PyPika, and we have added some extra functionality in the form of geometrical functions. It is now possible to specify a point in right ascension and declination and make ADQL search for that point in a box or cone, whose sizes are specified by the user. This allows us to enter the coordinates of a specific object, say the M1 galaxy, and ADQL will search for entries whose image might contain the object.

SQL has more functions, such as, using aggregate functions, creating or deleting entries, using subqueries and much more, which would allow us to create very complex and precise queries. For now, we will not use that for our dashboard, but with PyPika we could easily extend it and use those functions if we wanted to. We could also add more ADQL specific functions using the extended classes we created. This allows others to extend the PyPika functionality in combination with ADQL to their needs, if they would want to.

3.3.3 Panel

Once the server has returned some entries from the database, the user needs to be able to see those entries. Normally, using tables and plots in a notebook would be good enough, but we want to display it in an interactive and dynamic web-application. We found that the easiest way to do this was by using the Python library Panel (Rudiger et al. 2022). It is part of HoloViz, which hosts a collection of libraries to visualize data (Bednar et al. 2020).

With Panel, you can create interactive, web-based dashboards that allow users to gain more insight into their data. It also allows for static display of data, but in conjunction with its many widgets, it allows users to change the view of the dashboard. It is built on another Python library: Bokeh (Bokeh Development Team 2022). It uses Bokeh's basis and adds more functionality on top by allowing bidirectional communication between Python and JavaScript, which allows for the interactivity. It also uses the server provided by Bokeh, allowing it to run applications outside of Jupyter notebooks (Rudiger et al. 2022).

Panel can be broken down into three main different objects: Pane, Widget and Panel. The Pane object is the simplest of them all and is basically just a container for whatever a developer wants to display, such as plots and tables. The Widget object is an interactive object and allows

users to provide input, which can be used to update what the user sees. Examples of widgets are a date picker, text input and check boxes. Finally, the Panel object brings everything together where all the other objects are placed in. The Pane and Widgets objects are placed in a Panel object and a Panel object can even contain other Panel objects. With Panel objects the developer can decide what the application they are creating will look like. Luckily, the Panel object is very flexible and can be made to resize based on its contents or the size of the screen it is displayed on.

To change the look of the dashboard, templates can be used. We could create our own custom template and specify our own JavaScript and CSS, but Panel also has a few standard templates. These templates are built around standard CSS/JavaScript frameworks, such as Bootstrap and React, and they allow the developer to spend less time on the looks of the dashboard.

3.3.4 Param

When projects get bigger, it is always useful to be able to separate code by its functionality. With our dashboard, we would like to separate the code that handles the data and interactivity from the way the dashboard looks. Param is a really helpful tool for doing this. With Param we are able to create widgets and then style those widgets with Panel. Param widgets are also very powerful, because we can change the value of a Param widget in our code and we can make functions depend on parameterized widgets, using the `param.depends` decorator. Python decorators are basically functions that call other functions and by using this, users can change widgets values, which in turn will trigger some function to be called.

While Param is a very useful library, it does require the use of classes, which can be cumbersome, especially when creating small programs. For us it does not matter, because the dashboard requires quite a bit of code, so we were going to use classes anyway. Something else that could be a disadvantage is the lack of customization of the widgets Param provides, but as said earlier, we use Panel to change the way the widgets look, so it is not a problem for us (Rudiger et al. 2022). Below in Figure 3.3 we can see one of the many ways a simple button can be customized. The button in Figure 3.3a is purely from Param, while the buttons in Figure 3.3b are Param buttons customized with Panel.

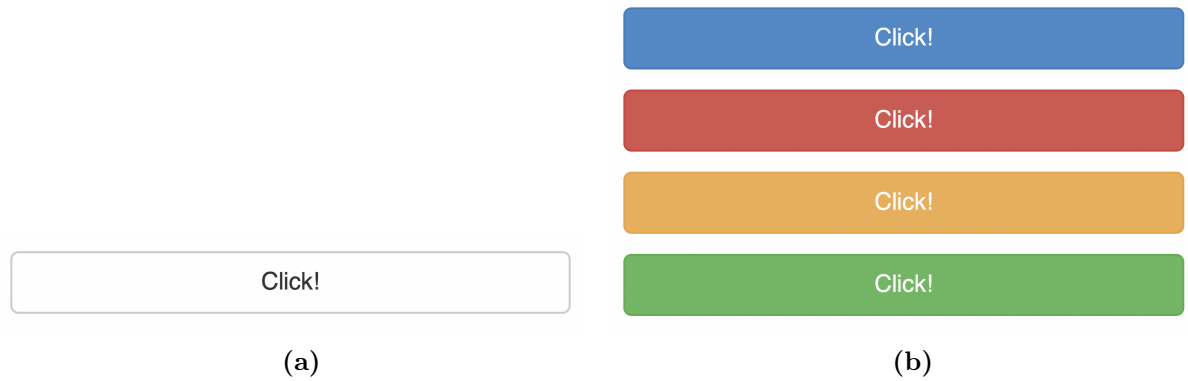


Figure 3.3: This figure displays one of the ways Panel can be used to customize parameterized widgets. The button in a) was created using only Param, while the buttons in b) were created by using Panel to stylize a Param button.

In the dashboard, we have used Param as the basis for every widget and used Panel to style the

widgets. This is especially useful for something like a table, because Param is fairly limited in that regard.

Param also allows us full control over the order in which the widgets are displayed on screen, which is slightly lacking for Panel. Every parameterized widget has a so-called 'precedence' attribute. The widgets will be displayed in order of precedence, with high precedence widgets being at the end of the list and low precedence widgets at the beginning of the list. Widgets can also be hidden by giving them a negative precedence. Furthermore, the precedence can be changed programmatically, allowing us to let widgets be hidden, whenever a user changes an option which might not use some of the available widgets. This allows us to give the user more choice, without having a cluttered dashboard.

3.3.5 astropy

When working with astronomical data in Python it is almost inevitable that one will use astropy (The Astropy Collaboration, Robitaille, et al. 2013; The Astropy Collaboration, Price-Whelan, et al. 2018). Astropy is a general astronomy Python library, which houses many standard astronomy related functions. For example, it has many constants and units and allows users to easily transform a variable from one unit to another. For our dashboard, the most important part of astropy is the FITS File Handling.

Generally, astronomical data is saved in FITS files, where FITS stands for Flexible Image Transport Systems. This file standard was first created in 1979 by Don Wells, Ron Harten and Eric Greisen (IAU FITS Working Group 2016). The first papers that described the standard were from Wells, Greisen, and Harten 1981 and Greisen and Harten 1981. In 1982 the International Astronomical Union endorsed the standard (IAU FITS Working Group 2016). Later, it was extended by Grosbol et al. 1988 and Harten et al. 1988 and from then on a special panel was established to really standardize the format (IAU FITS Working Group 2016).

With astropy, we are able to open the FITS files, look through the header and retrieve the image data. This is not yet important when querying the database, but we do need it for displaying the images and gathering some basic statistics about them. When we open a FITS file, astropy will return a Header Data Unit (HDU) list, which contains all the information from the file (The Astropy Collaboration, Robitaille, et al. 2013; The Astropy Collaboration, Price-Whelan, et al. 2018). Normally, this list has at least two entries; the first entry contains the header of the file, where all kinds of metadata is stored, while the second entry contains the data of the image itself. A file can contain multiple entries where image data is stored, but this is not true for the data at the Kapteyn server.

We also use astropy for some basic coordinates and unit conversions. From the coordinates package we use the SkyCoord and Angle classes. We need the SkyCoord class to transform the units of right ascensions and declinations. These are often given in hours, minutes and seconds and degrees, minutes and seconds respectively, but the geometric ADQL functions accept only degrees. Even though ADQL wants degrees, humans often prefer the hours/degrees, minutes and seconds system and that is what the Angle class is very useful for. We use this class on the data returned from the server, so all right ascensions and declinations are displayed in a way that is easier to understand.

The final part of astropy that we need is the time package, which contains the Time class. In the Blaauw database, we have observation dates in the Julian date system, which is not particularly readable for humans. So, our widgets use the Gregorian calendar and we need the Time class to convert between these two calendars.

3.3.6 Astroquery

While we expect most users to use the dashboard with a given date, such as the night they observed, some users might want to see data from specific objects. Searching for images that contain specific coordinate points is easy with the geometric functions of ADQL, but we first have to know what the coordinates of the object are. To find out what the coordinates of the specific object are, we use Astroquery (Ginsburg et al. 2019). Astroquery is a package affiliated with The Astropy Project, just as PyVO and astropy (The Astropy Collaboration, Robitaille, et al. 2013; The Astropy Collaboration, Price-Whelan, et al. 2018).

With Astroquery, we are able to query numerous astronomical databases containing lots of information about all kinds of objects. We can query the archives of ALMA, HST, Gaia and many more, but we are mostly interested in SIMBAD (Wenger et al. 2000). SIMBAD is a database containing information of millions of objects, such as right ascension and declination, aliases and fluxes (where applicable) and much more. For our dashboard, we will use SIMBAD for the coordinates and the aliases of objects.

When a user enters the name of an object, Astroquery will look for the object in the SIMBAD database, using the `query_object` method from the SIMBAD class. If SIMBAD contains an object with such a name, it will return information about it, such as the right ascension and declination. Astroquery then translates this information to an astropy Table object, which can be read by the application. It reads the coordinates, transforms them to degrees with the SkyCoord class from astropy and those coordinates can then be used for queries. We will talk more about this in Chapter 5.

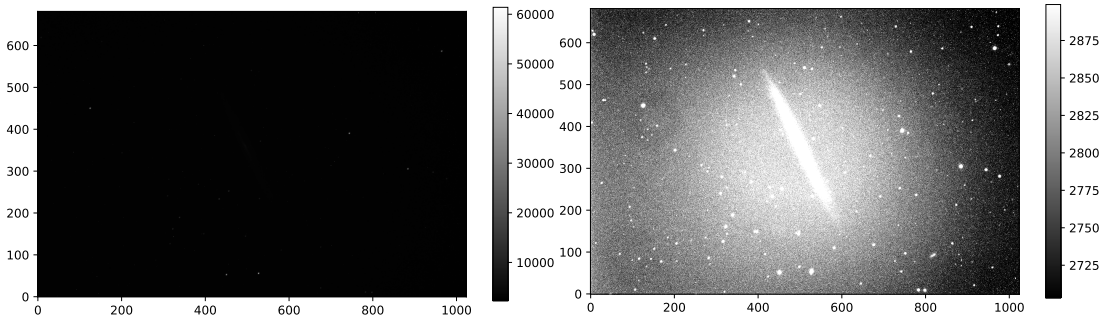
3.3.7 Matplotlib

Because we do not only want to display a table of entries, but also images of those entries, we need a plotting library. Panel supports quite a few plotting libraries: Bokeh, HoloViews, Matplotlib, Perspective, Plotly and Altair/Vega. From previous courses we are only familiar with Matplotlib, so we chose to use this as our plotting libraries (Hunter 2007). Some of the other libraries do have more functions, but we only have to display a static image from FITS data and Matplotlib is good enough for this.

Matplotlib is a Python plotting library with which you can create everything from 2D static plots, to 3D animation. It supports many different plot types, such as line plots, scatter plots and histograms, but we will only plot images and histograms. To plot the data from the FITS files, we will use the `imshow` method of the Axes class. With `imshow` we are able to plot an image made up of 2D data, which is exactly what the data from a FITS file is. The method returns an object, which can be placed in a Panel Matplotlib pane.

When we plot the image, we can also generate a colorbar next to our image. From the colorbar, users can see what the minimum and maximum values are within the plot, which can help when manipulating the data. FITS files often contain a few very low and high values, which can cause the plot to look like it is only one color, so we want to cut those values off when plotting. We can see this in Figure 3.4 below, where the values in Figure 3.4a were not cut off, while the values in Figure 3.4b were cut off at the 5th and 95th percentile.
We calculate these points by using the `percentile` function from NumPy (see Section 3.3.9). The cut-off points can then be seen in the colorbar and the user can use those values as a starting point when working on the data.

When using `imshow`, there are a few things we have to take into account. By default, Matplotlib will use an interpolation method to make the image look smoother, but because we are working with scientific data, we do not want that. We also have to make sure that we orient our

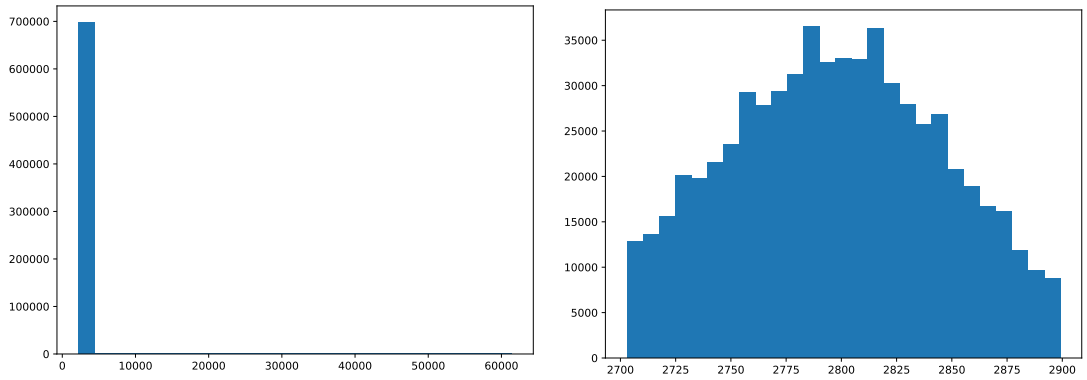


(a) **(b)**

Figure 3.4: The image in a) is the original data and the image in b) shows the data cut off at the 5th and 95th percentile. The difference is also clearly visible in the accompanying colorbars.

data correctly, because Matplotlib assumes that the origin of the data is in the upper left corner, but for FITS data the origin is in the lower left corner.

To plot some basic statistics, we use histograms. For the histograms we have the same problem as with the image; there are a few very low and high values, which can result in a useless histogram. This can be seen in Figure 3.5 below, where once again the values in Figure 3.5a were not cut off, while the values in Figure 3.5b were cut off at the 5th and 95th percentile.



(a) **(b)**

Figure 3.5: The image in a) shows the histogram of the original data and the image in b) shows the histogram cut off at the 5th and 95th percentile.

To counter this problem, we once again use the `percentile` function and this gives a histogram that is actually useful. We also have to be careful with the amount of bins we use, because too few bins gives little insight, while too many bins can give an almost flat histogram. For the amount of bins, we use the square root of the amount of data points and round up to an integer. There are other methods of determining the amount of bins, but those are often much more computationally expensive and we are just using the histograms for a first look, which our method is good for.

3.3.8 pandas

When querying the database, PyVO returns a list of results. This list is not particularly useful for manipulating the data, so we transform it to an astropy Table object, but we can go one step further and transform it into a pandas DataFrame. pandas is a library specifically for easily analyzing and manipulating data by using its DataFrame object (The pandas development team 2020; McKinney 2010). It allows us to easily apply functions to certain columns, filter the data and display it using Panel. It is also very fast, because it uses Cython, a library to use C code in Python, and C itself (Behnel et al. 2011), which allows us to manipulate the data, without slowing the dashboard down.

3.3.9 NumPy

The final major library we will use is NumPy (Harris et al. 2020). NumPy is a general-purpose package for scientific computing and is mostly famous for its arrays, which are written in C, so they are very fast. Generally, when manipulating data, it is thus recommended to use NumPy arrays, but in some cases these can actually be slower than normal Python lists. When appending data to an array or a list, using the list is actually quite a bit faster and should be used over the NumPy array. On our computer, we tested this by appending five thousand integers to both a list and an array and timed it. This resulted in the `append` method for the Python list being about 80 times faster than the `append` method for the NumPy array. So, even though NumPy arrays can be very fast, we will exclusively use Python lists when appending data, to make the dashboard just a bit faster.

We do need NumPy for other functions, such as calculating the median of many data points. We need to do this quite often and using `numpy.median` is about three times faster than using Python's `statistics.median` when calculating the median of five thousand random numbers. Even though this is not as big of a difference as with the `append` methods, it still gains us time, which should result in a slightly more responsive application.

NumPy also has a function to calculate percentiles. This is useful for us, because we generally want to cut off the lowest and highest 5% of an image to get the best result. Otherwise we get images that do not tell us anything, as in Figure 3.4 and histograms which are useless, as shown in Figure 3.5.

Chapter 4

Process

In the previous chapters we discussed the Blaauw Archive, its limitations and what we would like to see in a solution to these limitations. We also discussed all of the tools we will use, how they work and why we need them. Now that we have all this information, we can start working on the Blaauw Dashboard and in this chapter we will take you through this process. The code for the dashboard is on GitHub, if you want to look at the completed process[1].

First of all, we looked at the already available software. Sipma (2021) had set up the TAP service, which our dashboard could hook in to. When deciding on what tool to use for the dashboard, we chose Python (see Section 3.1), but there were many different libraries that could be used for creating such an application.

We looked at the HoloViz family of packages and began to follow some of the tutorial provided by these packages (Bednar et al. 2020). This allowed us to gain more understanding of what each of these packages was capable of. The most notable of these were Bokeh and Panel, which are the main packages for creating a dashboard (Bokeh Development Team 2022; Rudiger et al. 2022). Working with these two libraries, we found the documentation of Panel easier to follow, as well as having an easier to use syntax, so we chose for Panel (see Section 3.3.3). This is of course a subjective choice and one could probably get the same results with Bokeh.

Once we had decided on Panel, we made some small stand-alone test apps, to see how everything works, such as creating widgets, adding interactivity and displaying the dashboard. We wanted to be able to allow these widgets to interact with the database immediately, but we had no proper and extensible manner to do this. Querying the database with manually written queries was a possibility and this is what we used at first, but we want to have an interactive dashboard that does not depend on the user writing their own queries. We tried to create our own, very primitive query builder. A query builder is a way to compose a query dynamically, so the query can be changed based on input by the user. Our own query builder was based on many `if`-statements and was rather limiting. It worked when having only a few very basic widgets, but it fell apart rather quickly once we wanted more complicated queries. So, we looked for an already existing query builder and we found it in PyPika (see Section 3.3.2) (PyPika 2022).

PyPika is a library for composing queries in Python. It supports a few different dialects of SQL, but it was not compatible with ADQL. This forced us to extend PyPika ourselves, by adding ADQL as a new SQL dialect and implementing the ADQL functions. While ADQL is based on SQL, it requires the `TOP` clause and the real strenght of it is in the geometric functions, which are ADQL-specific. So, we had to extend a few of the base classes used by PyPika and added ADQL functionality. Now we are able to look for coordinates using a box or cone search. If we ever want to add more ADQL-specific functionality, we can use the classes we created and seamlessly integrate those new functions.

Now that we had chosen a library to create the dashboard and a way to query the database and return information from it, we could start creating the application. With PyPika we can return the information in tabular form and this is also how we chose to display it. Users can immediately see the most important information in a clear way and with a table we are able to show lots of information at once. We used the Tabulator widget from Panel for this, which is based on Tabulator, a JavaScript library for creating tables (Folkerd 2022). With the Tabulator

[1] `https://github.com/PracticalAstronomyCrew/Blaauw_Dashboard`

widget we are able to create a table containing all information and let other widgets communicate with it.

To implement being able to search for objects, we used SIMBAD (Wenger et al. 2000). The service for querying SIMBAD is build into Astroquery, in the form of the `query_object` function (Ginsburg et al. 2019). We combined this with the box and cone search functions specified by ADQL (see Section 3.2) and implemented this in PyPika. Whenever a user types in a name that is recognized by SIMBAD, SIMBAD will return its coordinates and the dashboard uses those coordinates to look for the object. If the name specified by the user is not known to SIMBAD, Astroquery generates an error, which would stop the whole dashboard. We solved this by handling the exception using Python's built-in `try ... except` syntax. This allows the dashboard to search for the object in the SIMBAD database and automatically displays an error whenever the object cannot be found and ensures that the dashboard still works correctly.

Users could now search the whole database by specifying a night, an object or a set of coordinates, but they could not yet inspect the returned files. We decided to show the plots on a different tab, because the main tab was already filled with the Tabulator widget. We decided on allowing a user to plot at most three images at the same time. More images would not all fit on the same screen, so users could not compare the images to each other. We created a second Tabulator widget, in which the selected files from the main tab are shown. In this second widget, only the shortened file name, the filter and the object are shown. More information would have required a larger table, but we wanted as much space available as possible for the plots. We tried to use one single function for all three plots, but this did not work. When given a list of Matplotlib objects, Panel did not display them, but we do not know why. We solved this by creating one small function for each plot, which calls a general plotting function that can generate plots based on a file name given by each of these smaller functions. It is not the most elegant solution and introduces some redundancy, but it works for what we need it to do.

To give the user some insight in to their calibration files, we created another tab where a user can choose a specific night, get some statistics for that night and compare it to previous data. To display the statistics of the chosen night, we used the median of the bias frames, read noise, dark frames and flat fields. For the comparison, we wanted to show the previous data as histograms, with a vertical line showing the median of the chosen night. Calculating the medians of all previous nights during runtime took too long. We would have to go through all calibration files for all years with a certain binning, open them, calculate the median, close the file, append the median to a list and return it. This takes tens of minutes, during which the user would have to wait. Instead of this, we opted to do these calculations preemptively and save everything to text files. These can be read in during runtime, which takes only a fraction of a second.

Finally, we wanted to make the dashboard look more presentable, because with adding all the widgets, the UI had gotten a bit cluttered. Panel allows you to specify a custom layout, which allowed us to organise our widgets however we want and thus solved our problem. We also wanted to make sure that color blind people could use our application and we checked this with a color blindness simulator[2]. When using this simulator, we found that the buttons and different colors on the plots were distinguishable, so a user with color blindness should still be able to use the Blaauw Dashboard.

[2] `https://www.color-blindness.com/coblis-color-blindness-simulator/`

Chapter 5

Program showcase

Now that we have laid out all the tools and the process, we can start showing the application itself. In this chapter we will show the Blaauw Dashboard and all of its features. In Section 5.1 we will talk about the opening view users see when they start the program. In Section 5.2 we will show the tab where users can plot the images they selected and in Section 5.3 we will discuss the tab where some basic statistics are shown for a specific night.

A user guide explaining more in-depth how to get started on working with the dashboard can be found in Chapter 8.

5.1 Querying the database

When a user first opens the application, Figure 5.1 is what they will see. Here we can see the table, which contains all of the information returned from the query, and the settings, which contains all the widgets of which the user can change the value. Once a user has dialed in the settings to their liking, they can click the 'Search' button and the application will do its work.

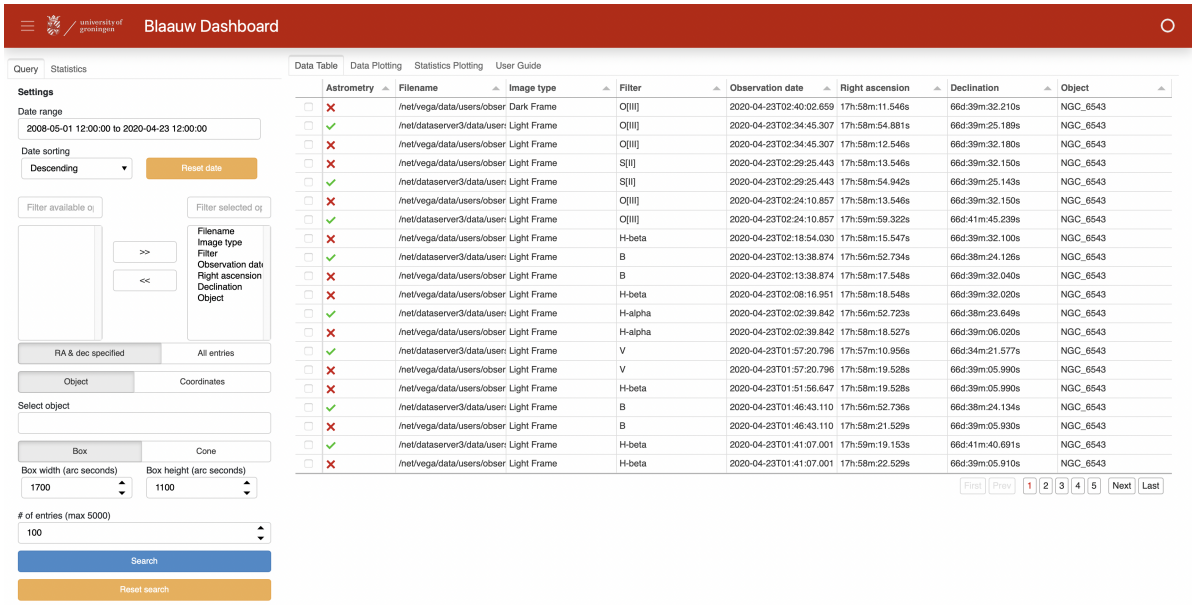
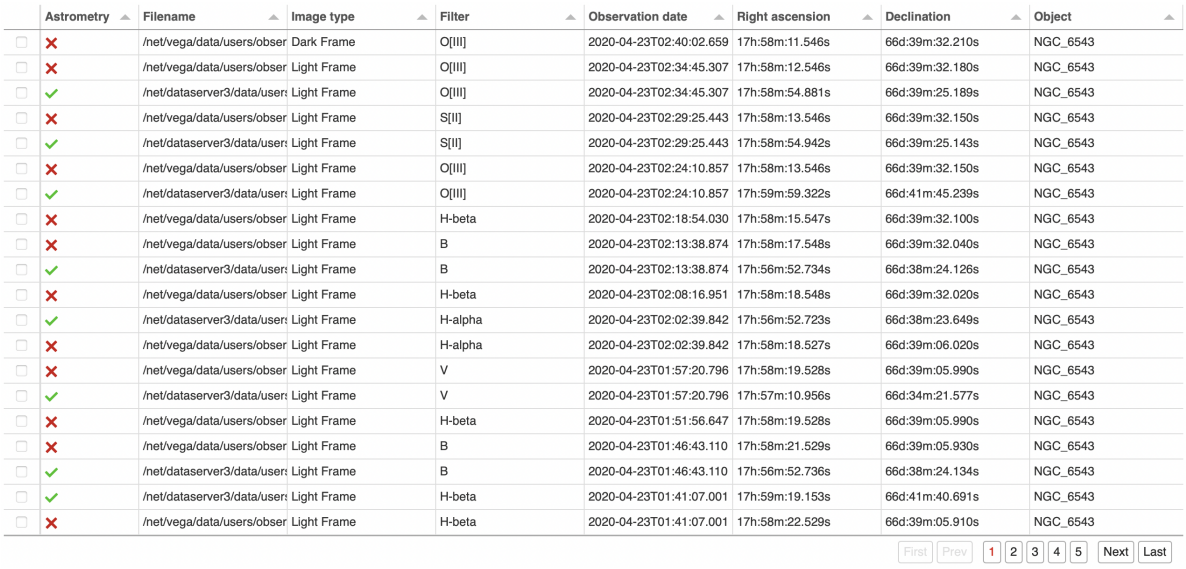


Figure 5.1: This is the main opening screen of the dashboard. It shows the widgets, whose values can be changed by the user, on the left and the table with all the information returned from the query in the middle.

First, we will discuss the table containing all of the information and then we will discuss the different settings the user can change.

A larger scale image of the Tabulator can be seen in Figure 5.2 below. This is just one view and will of course change depending on the settings chosen by the user. In the figure we can see the columns and their corresponding data. The first column shows checkboxes. These are part of the widget and allow the user to select the files they want to plot on the plotting tab.



Astrometry	Filename	Image type	Filter	Observation date	Right ascension	Declination	Object
✗	/net/vega/data/users/obser	Dark Frame	O[III]	2020-04-23T02:40:02.659	17h:58m:11.546s	66d:39m:32.210s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	O[III]	2020-04-23T02:34:45.307	17h:58m:12.546s	66d:39m:32.180s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	O[III]	2020-04-23T02:34:45.307	17h:58m:54.881s	66d:39m:25.189s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	S[II]	2020-04-23T02:29:25.443	17h:58m:13.546s	66d:39m:32.150s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	S[II]	2020-04-23T02:29:25.443	17h:58m:54.942s	66d:39m:25.143s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	O[III]	2020-04-23T02:24:10.857	17h:58m:13.546s	66d:39m:32.150s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	O[III]	2020-04-23T02:24:10.857	17h:59m:59.322s	66d:41m:45.239s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	H-beta	2020-04-23T02:18:54.030	17h:58m:15.547s	66d:39m:32.100s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	B	2020-04-23T02:13:38.874	17h:58m:17.548s	66d:39m:32.040s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	B	2020-04-23T02:13:38.874	17h:56m:52.734s	66d:38m:24.126s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	H-beta	2020-04-23T02:08:16.951	17h:58m:18.548s	66d:39m:32.020s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	H-alpha	2020-04-23T02:02:39.842	17h:56m:52.723s	66d:38m:23.649s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	H-alpha	2020-04-23T02:02:39.842	17h:58m:18.527s	66d:39m:06.020s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	V	2020-04-23T01:57:20.796	17h:58m:19.528s	66d:39m:05.990s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	V	2020-04-23T01:57:20.796	17h:57m:10.956s	66d:34m:21.577s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	H-beta	2020-04-23T01:51:56.647	17h:58m:19.528s	66d:39m:05.990s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	B	2020-04-23T01:46:43.110	17h:58m:21.529s	66d:39m:05.930s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	B	2020-04-23T01:46:43.110	17h:56m:52.736s	66d:38m:24.134s	NGC_6543
✓	/net/dataserver3/data/user	Light Frame	H-beta	2020-04-23T01:41:07.001	17h:59m:19.153s	66d:41m:40.691s	NGC_6543
✗	/net/vega/data/users/obser	Light Frame	H-beta	2020-04-23T01:41:07.001	17h:58m:22.529s	66d:39m:05.910s	NGC_6543

First Prev 1 2 3 4 5 Next Last

Figure 5.2: A close-up view of the Tabulator widget. The files shown here are just the files shown whenever the dashboard is started. Whenever a user makes changes to the widgets, the files shown will also change.

Next, we see an Astrometry column, which checks if a filename contains the word 'astrom'. This is how files put through astrometry.net in the pipeline are identified in the server and they are often preferred. The rest of the columns contain information that was already in the database, such as the filename, image type, filter, observation date, coordinates and object. In the best case scenario, all of these columns are filled, but as we saw already in Section 2.3, this is not always true, especially for the coordinates. From these columns, the user can identify the files they were looking for and inspect them more closely. When needed, they can always download the files from the server, which should be easier with the complete path to the file displayed in the filename column.

The columns are resizable for when the information in a column cannot be read when the column is at its standard size. We opted for this instead of having columns display all information by default, because this can obscure the rest of the information. For example, the data in the filename column is generally very long and extending it by default would hinder the view of the rest of the data. For this reason, we also let the widget create pages, instead of displaying all entries on one page. Furthermore, the widget allows the user to sort by column, by clicking on the name of the column. This will only sort the already available data and will not send another query to the database.

Finally, the Right Ascension and declination are displayed differently then how they are saved in the database. The database gives these coordinates in degrees, but it is generally preferred to have the Right Ascension in hours, minutes and seconds and the declination in degrees, arc minutes and arc seconds.

Next, we will look at the settings tab, which contains all of the widgets. This can be seen in Figure 5.3 below.

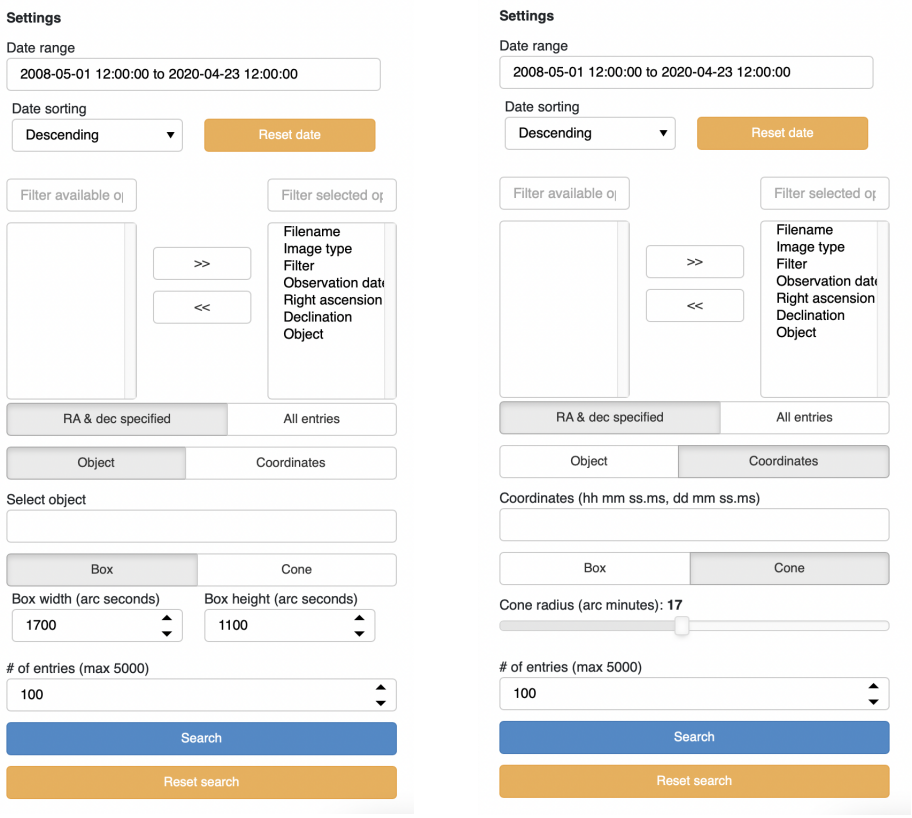


Figure 5.3: This figure displays the settings tab for the Main Screen. It contains all the widget a user can use to control what data they want to retrieve. Figure a) shows the settings with the object and box options selected, while Figure b) shows the settings with the coordinates and cone options selected.

At the top of the tab, a user can select the date range they want to search in. When looking for objects or coordinates, searching the whole date range is generally preferred, to get as many entries as possible. Below the date range, one can choose how to order the data; ascending or descending dates. A reset date button is also available, which resets the date range to encompass all of the data. Next, we have a column selector, with which the user can select which columns they want to see in the table. There are three columns whose information are always needed for the rest of the dashboard, namely filename, filter and object. These columns can be deselected, but the dashboard still includes them in the query, they will just be hidden from the user. The user can select whether they want to only include files that have a Right Ascension and declination or simply include all files.

Below this, search for an object or search using coordinates and the dashboard automatically includes only files with a Right Ascension and declination. When searching for an object, the name has to be a name that has been registered by SIMBAD. For the coordinates, the dashboard assumes the Right Ascension is given in hours, minutes and seconds of time, with spaces in between, while the declination is assumed to be in degrees, arc minutes and arc seconds, also with spaces in between. Furthermore, Right Ascension and declination should be separated using a comma. If the first number of the Right Ascension is 24 or higher, the dashboard will assume degrees were used. If the first number is lower than 24, the dashboard will assume hours were used. For example, a Right Ascension of 20 30 00 is assumed to be in hours, minutes and seconds

and will be transformed to 307.5 degrees to use in calculations. Right Ascension can also be given in decimal hours, so that 20.5 also gives us 307.5 degrees. If a Right Ascension of, for example, 50.6 is given, then this will simply be assumed to be degrees. The declination can also be entered in degrees only by using a single floating-point number. If the field is left empty, the dashboard will not look for a specific object or for specific coordinates.

If an object or a set of coordinates has been entered, the user must specify whether they want to do a box or cone search around the object or coordinates. For the box search the width and height have to be entered in arcseconds, while the cone search requires a radius in arcminutes. The reason for using arcseconds for the box search, is that the platescale is generally given in units of arcsec/pixel. The object or coordinates are put at the center of the box and the width and height are the complete lengths of the sides of the box, so they are not the width and height as seen from the object or coordinates. For the cone search, the object or coordinates are also put at the center and the user can specify the radius of the cone around the center.

Finally, the user can select the number of entries they want to have returned. Currently, the limit is set to a maximum of five thousand returned entries. Returning thousands of entries takes longer and is generally not necessary, so it is advised to first use the default and increase the amount when needed. Once done with setting all the widgets, the search button can be clicked and the dashboard will retrieve the matching data. A button to return all fields to their defaults is also available.

Whenever the dashboard cannot find any entries, it will display an error in the bottom right, telling the user what went wrong. This happens whenever SIMBAD cannot find the object specified or whenever the database simply has no entries for the settings chosen by the user. The error is displayed until it is dismissed by the user, so they can take their time reading it.

5.2 Plotting the data

Once the user is satisfied with the entries returned, they can select and plot them on the 'Data Plotting' tab. The result of this can be seen in Figure 5.4. These plots can be used as a quick inspection of the wanted data. It will allow the user to check if the data might be good enough for their research, without having to download the images. On the left we see the same settings as for the Main Screen, but they are not needed here and the settings tab can be collapsed by clicking the three horizontal lines in the top left. In the center, we have a table and three plots. In the table, the files selected by the user on the main tab are displayed. Here they can be selected once again to plot them. We have restricted the number of simultaneous plots to three, because if we were to show more plots, they could not all be viewed at the same time.

As we saw in Section 3.3.7, plots of FITS files are generally completely black and do not show any objects, unless a portion of the minimum and maximum values are cut off. We chose to cut the values off at the 5th and 95th percentile, which results in images that can be inspected. When doing research with the images, one might want to use different values, but for a quick inspection, this is sufficient. We display the colorbar, from which the user can see the minimum and maximum values displayed in the images, which can be used as a starting point when doing research. The title of each plot shows the file name and the filter.

Just as for the Main Screen, the Dashboard will display errors when something goes wrong. It will display errors when the user clicks the plotting button and no files have been selected or when more than three files have been selected and if one of the selected files could not be found. We tried displaying the filename in the error, whenever a selected file could not be found, but the filenames can be very long and this resulted in the error going off the screen and making it illegible.

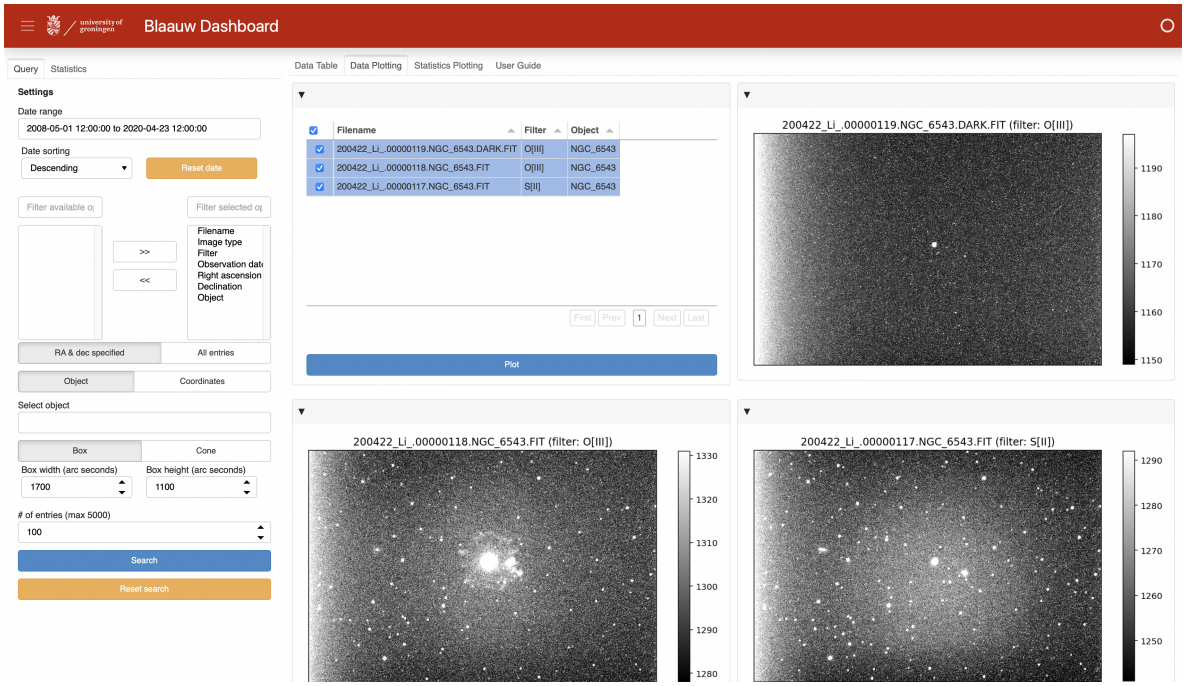


Figure 5.4: This is the appearance of the tool when a few figures have been plotted. Three files were selected from the main tab and then selected on the plotting tab and plotted. When no files are selected, this screen will just show empty plots.

5.3 Nightly statistics

The final tab available is for the nightly statistics. The screen and its settings can be seen below in Figure 5.5

We see here that the settings tab has changed with respect to the other two tabs. The user can specify a minimum and maximum percentile for the histograms, choose a night and click the plotting button. The percentile has the same use as when plotting the files (see Section 3.3.7). This will generate four histograms, displaying different statistics for the chosen night and all previous data. In the top left we have the median of the bias frames, in the top right the median of the read noise, in the bottom left the median of the dark frames and finally in the bottom right the median of the flat fields. All of the histograms are grouped by binning, because two different binnings will result in different values and thus cannot be compared. Furthermore, we corrected the dark frames with their exposure times and bias frames. Most of the value in a pixel for the dark frame is made up of the bias and each dark frame can have a different exposure, so we had to correct them to be able to compare them. This does result in the user having to wait longer for the dark frames plot, because correcting each dark frame and then plotting them takes longer than just plotting them.

With these statistics, the user can identify if their calibration might be problematic, compared to previous calibration frames. This knowledge can be used to resort to using calibration frames of other nights for research.

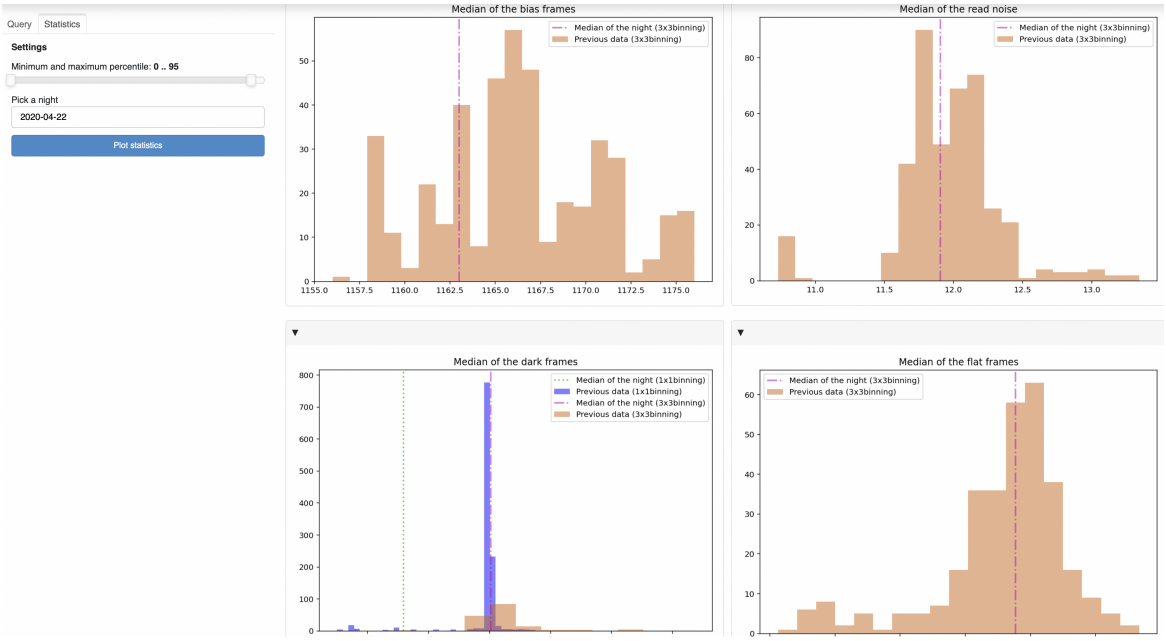


Figure 5.5: This figure displays the appearance of the dashboard when plotting nightly statistics after the user has chosen a night. The exact plots will depend on the night chosen. The image has been cut off slightly, to fit everything into it.

Chapter 6

Conclusion

In this thesis we looked for a way to allow users easier interaction with the data at the Blaauw archive. Our solution was to use Python and a few of its libraries to create a dashboard web-application. With this application, users can query the database in an easy and intuitive manner. Users can search for observations of specific nights or they can search for an object or a set of coordinates. The returned images can then be plotted, which allows for a quick inspection of the data and can help the user decide whether it is what they need. Furthermore, some statistics for a specific night can be displayed and compared to previous data. From this comparison the user can judge whether they are satisfied with the taken calibration frames or if they might want to use calibration frames from previous nights.

In the introduction we discussed some of the problems users currently face with the data that we want to solve with our application. We will now discuss whether our application manages to solve these problems. We will also look at other features we might want to add in the future, to make the Blaauw Dashboard more useful or user-friendly.

6.1 Solved and unsolved problems

In the introduction we stated that inspecting observation data from the Blaauw Observatory can be a confusing process. Files are hidden in folders at about nine folders deep and many folders contain many other folders, making the path to the files not straightforward. Once the user has reached the files they want, to need to either download them through the terminal or an FTP-program. Only then can the files be inspected and used for research. While downloading files through the dashboard is not possible, inspecting files has become much easier. Users only have to start the application and they can immediately query the database and inspect their files.

With the Blaauw Dashboard, looking for files containing specific objects has become possible. Earlier, users would have to look through observation logs to check what object was observed on a specific night. Since a few years these logs have become mostly digital, making going through them easier, but there are still many handwritten logs. When observing, the computer is supposed to add the object to the header of the FITS files, but this does not always works and thus the header cannot always be trusted. Now, users can type in the name of the object in the dashboard and it will return the entries containing the object. This is useful for users looking to research a specific object or specific type of object and for whenever the weather forbids observing.

Another request we had for the dashboard, was that it had to be user-friendly and intuitive. We gave all widgets self-explanatory names and entered default values that are a good starting point for most users. Users can use the dashboard without needing any knowledge of ADQL or Python, which will open up the application to more people. The dashboard is also color blind friendly. Furthermore, a user guide is available which contains examples to explain all of the features.

We did not solve the problem of making the data easier to download. The dashboard can be used to inspect files, but if one wants to work with those files, they still have to go through the hassle of downloading them from the server.

The dashboard is not yet available on the Kapteyn computers, so users will have to download the application from GitHub and start it via the command line. Even though starting the application is a short and simple command, this does hurt user-friendliness.

6.2 For the future

There are some features we would like to see added to the dashboard in the future. Most importantly, it should be made available on the Kapteyn computers for everyone to use. We know the dashboard works on the Kapteyn computers, but we want to install it on the server. This way, everyone will be able to use the application, without having to download the files from GitHub and starting it via the command line. Having the dashboard in one central location also allows for easier updates whenever new features are added.

Downloading files from the dashboard would make it the all-in-one application for discovering, inspecting and downloading one's files. We are not yet sure what would be the best way to implement this, because having to select each file individually is not desirable. Being able to download complete folders might solve this, but sometimes a user might only need a few files and downloading a folder with many unwanted files would be a waste of time and disk space.

Some images have a better quality than others and if we could inform the user about the quality of the images, they could choose the best images for their research. We looked into doing this for the dashboard, by using machine learning, but ultimately decided not to do it. We would not have had enough time to properly develop and train the algorithm, but it could be a great addition for the future.

The nightly statistics could be made more useful, by expanding it. Adding a seeing estimate for a few images through the night gives the user an approximation of the seeing through the whole night. If we combine the seeing estimate with things such as temperature, humidity and cloudiness, we could get an idea of the quality observation. If we record this over a longer stretch of time, we could maybe give crude predictions of the observation quality for a future night, given the expected weather conditions.

Finally, the dashboard can be a bit slow when plotting the statistics histograms. The reason for this is that we correct the dark frames. This is necessary, because most of the values in those pixels come from the bias frames, but it does slow the dashboard down. One possible solution might be to create a C++ program which does the calculations and import it into our Python code. Generally, writing code in C++ does take more time, but C++ is much faster than Python (Tamimi 2021). We are not sure how big the difference would be, because many of the Python packages utilize C already, so more research is needed to determine if it would be useful for our purposes.

Chapter 7

References

Becker, Stefan et al. (Feb. 2022). *astropy/pyvo: v1.3*. Version v1.3. DOI: 10.5281/zenodo.6168226. URL: https://doi.org/10.5281/zenodo.6168226.

Bednar, James A. et al. (June 2020). *holoviz/holoviz: Version 0.11.4*. Version v0.11.4. DOI: 10.5281/zenodo.3873665. URL: https://doi.org/10.5281/zenodo.3873665.

Behnel, Stefan et al. (Mar. 2011). "Cython: the Best of Both Worlds". In: *Computing in Science & Engineering* 13, pp. 31–39. DOI: 10.1109/mcse.2010.118.

Bokeh Development Team (2022). *Bokeh: Python library for interactive visualization*. URL: https://bokeh.pydata.org/en/latest/.

Chamberlin, Donald D. (Oct. 2012). "Early History of SQL". In: *IEEE Annals of the History of Computing* 34, pp. 78–82. DOI: 10.1109/mahc.2012.61. URL: https://ieeexplore.ieee.org/abstract/document/6359709 (visited on 06/11/2022).

Dowler, Patrick, Markus Demleitner, et al. (May 2017). *International Virtual Observatory Alliance: Data Access Layer Interface Version 1.1*. URL: https://www.ivoa.net/documents/DALI/20170517/REC-DALI-1.1.pdf (visited on 06/10/2022).

Dowler, Patrick, Guy Rixon, et al. (Aug. 2019). *International Virtual Observatory Alliance: Table Access Protocol Version 1.1*. URL: https://www.ivoa.net/documents/TAP/20190927/REC-TAP-1.1.pdf.

Folkerd, Oli (2022). *Tabulator - Interactive JavaScript Tables*. tabulator.info. URL: http://tabulator.info/docs/5.2 (visited on 06/19/2022).

Ginsburg, Adam et al. (Mar. 2019). "astroquery: An Astronomical Web-querying Package in Python". In: *The Astronomical Journal* 157.3, 98, p. 98. DOI: 10.3847/1538-3881/aafc33. arXiv: 1901.04520 [astro-ph.IM].

Greisen, E. W. and R. H. Harten (June 1981). "An Extension of FITS for Groups of Small Arrays of Data". In: *Astronomy & Astrophysics Supplement Series* 44, p. 371.

Grosbol, P. et al. (June 1988). "Generalized extensions and blocking factors for FITS". In: *Astronomy & Astrophysics Supplement Series* 73.3, pp. 359–364.

Harris, Charles R. et al. (Sept. 2020). "Array Programming with NumPy". In: *Nature* 585, pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

Harten, R. H. et al. (June 1988). "The FITS tables extension". In: *Astronomy & Astrophysics Supplement Series* 73.3, pp. 365–372.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

IAU FITS Working Group (July 2016). *Definition of the Flexible Image Transport System (FITS)*. URL: https://fits.gsfc.nasa.gov/standard40/fits_standard40aa.pdf (visited on 06/10/2022).

International Virtual Observatory Alliance (2022). *IVOA.net*. www.ivoa.net. URL: https://www.ivoa.net (visited on 06/09/2022).

Lang, Dustin et al. (May 2010). "Astrometry.net: Blind Astrometric Calibration of Arbitrary Astronomical Images". In: *The Astronomical Journal* 139.5, pp. 1782–1800. DOI: 10.1088/0004-6256/139/5/1782. arXiv: 0910.2233 [astro-ph.IM].

McKinney, Wes (2010). "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

Ortiz, Inaki et al. (Sept. 2008). *IVOA Astronomical Data Query Language Version 2.0 IVOA Recommendation 30 Oct 2008 Previous version(s): Editor(s)*. URL: https://www.ivoa.net/documents/REC/ADQL/ADQL-20081030.pdf.

PyPika (2022). *PyPika - Python Query Builder - Documentation*. pypika.readthedocs.io. URL: https://pypika.readthedocs.io/en/latest/ (visited on 06/09/2022).

Python Software Foundation (2019a). *PyPI – The Python Package Index*. PyPI. URL: https://pypi.org (visited on 06/09/2022).

Python Software Foundation (2019b). *Python 3.9 Documentation*. Python.org. URL: https://docs.python.org/3/ (visited on 06/09/2022).

Rudiger, Philipp et al. (May 2022). *holoviz/panel: Version 0.13.1*. Version v0.13.1. DOI: 10.5281/zenodo.6576241. URL: https://doi.org/10.5281/zenodo.6576241.

Silberschatz, Abraham, Henry F. Korth, and S. Sudarshan (Jan. 2010). *Database System Concepts*. 6th ed. McGraw-Hill.

Tamimi, Naser (Jan. 2021). *How Fast Is C++ Compared to Python?* Medium. URL: https://towardsdatascience.com/how-fast-is-c-compared-to-python-978f18f474c7 (visited on 06/26/2022).

The Astropy Collaboration, A. M. Price-Whelan, et al. (Aug. 2018). "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package". In: *The Astronomical Journal* 156, p. 123. DOI: 10.3847/1538-3881/aabc4f.

The Astropy Collaboration, Thomas P. Robitaille, et al. (Sept. 2013). "Astropy: a Community Python Package for Astronomy". In: *Astronomy & Astrophysics* 558, A33. DOI: 10.1051/0004-6361/201322068.

The pandas development team (Mar. 2020). *pandas-dev/pandas: Pandas 1.0.3*. Version v1.0.3. DOI: 10.5281/zenodo.3715232. URL: https://doi.org/10.5281/zenodo.3715232.

The Qt Company (2019). *Qt | Cross-platform Software Development for Embedded & Desktop*. www.qt.io. URL: https://www.qt.io (visited on 06/08/2022).

University of Groningen (Oct. 2011). *Blaauw Observatory*. University of Groningen. URL: https://www.rug.nl/research/kapteyn/sterrenwacht/ (visited on 06/06/2022).

Wells, D. C., E. W. Greisen, and R. H. Harten (June 1981). "FITS - a Flexible Image Transport System". In: *Astronomy & Astrophysics Supplement Series* 44, p. 363.

Wenger, M. et al. (Apr. 2000). "The SIMBAD astronomical database. The CDS reference database for astronomical objects". In: *Astronomy & Astrophysics Supplement Series* 143, pp. 9–22. DOI: 10.1051/aas:2000332. arXiv: astro-ph/0002110 [astro-ph].

Chapter 8

Appendix

8.1 User guide

8.1.1 Introduction to searching the database

When you first open the Blaauw Dashboard, the view in Figure 8.1 is what you will see.

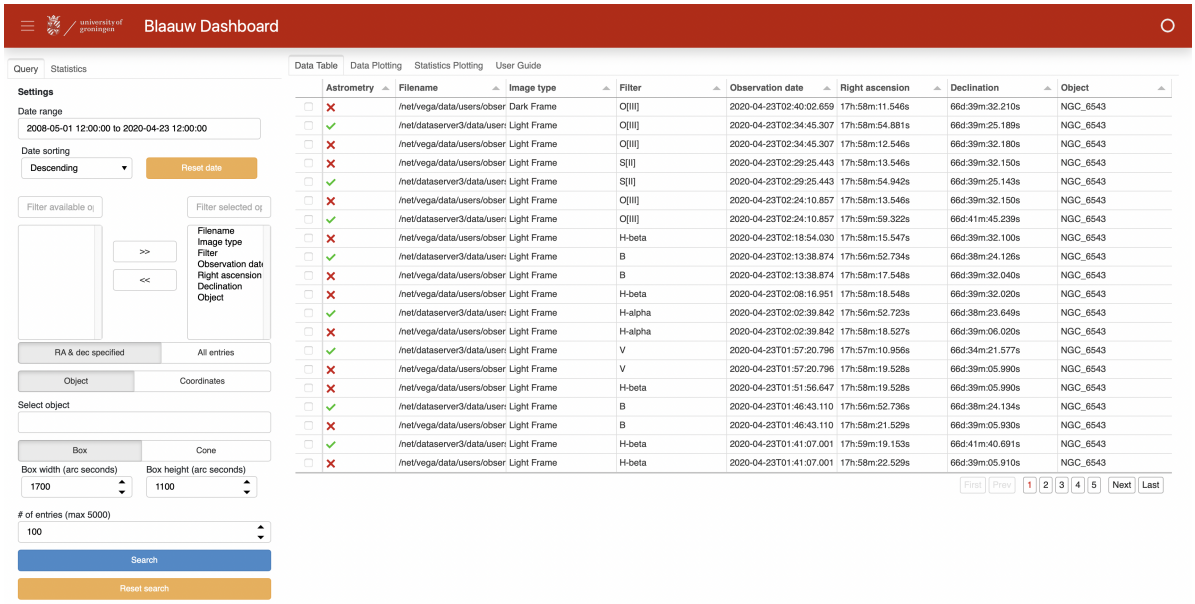


Figure 8.1: This is the main screen of the Blaauw Dashboard and is what you will see when you first open it. In the center we have the table containing all information and the interactive widgets on the left. Once you have changed some settings, the view will be different.

In the center it shows the table containing all the entries returned by the database. You can click on the names of the columns to sort them. You can also click the check boxes in front of the entries to select them and use them on the next tab for plotting. On the left side we have the widgets of which you can adjust the values and this is what we will focus on in the user guide.

Below in Figure 8.2 we have two figures showing a zoomed-in view of the available settings. The one on the left is the default one, with the object and box search, while the one on the right shows the coordinates and cone search.

Picking a date

Now we can really start searching the database. To begin with, you can choose a date range over which you want to search. If you are looking for data from a specific night, just choose that night, but if you want data from a certain object or set of coordinates, you should probably use the whole date range. This way you will have a bigger chance of actually finding the data. You can also choose how you want the data sorted. With descending you will get the latest data first

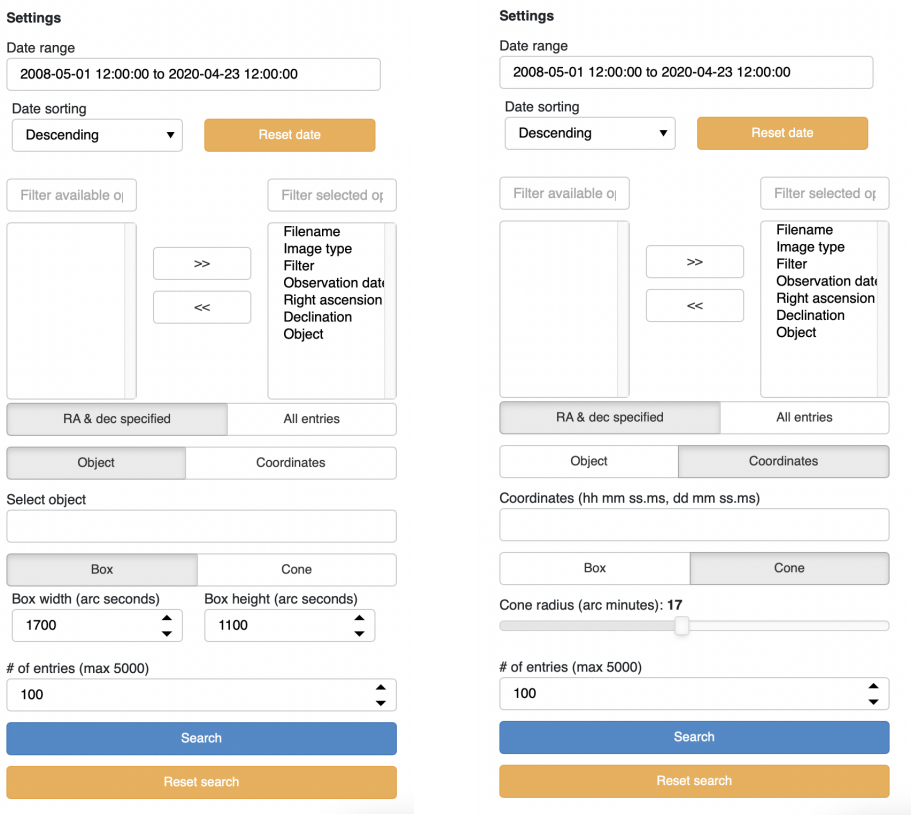


Figure 8.2: This figure displays the settings tab for the main screen. It contains all the widgets you can use to control what data you want to retrieve. Figure a) shows the settings with the object and box options selected, while Figure b) shows the settings with the coordinates and cone options selected.

and with ascending you will get the oldest data first. The reset date button will set the date range to the default date range, which can be useful when you first wanted to look at data for a specific night, but now want it for a bigger range.

Selecting columns

Then we have the column selector where you select which columns you want to see in the table. Currently these are the only columns available, but for quick inspections of data, this should be more than enough. The order in the column selector is also the order that the columns will be shown in the table.

Coordinates specified or all files

Next, you have to choose whether you only want files that have a Right Ascension or declination or simply all files. When you are looking for data from a specific night, you might want all files, because you just want all files from that night. When looking for objects or specific coordinates, the dashboard will automatically only use files that have a Right Ascension and declination.

Looking for objects and coordinates

Now we finally get to what is probably the most useful feature for looking through the Blaauw archive: searching for a specific object or set of coordinates. To search for an object, you just type in its name. For example, you could search for 'M1' and the dashboard will look through the database to find images that contain M1. It does this by using a box or cone search, which we will get to next. Many objects have multiple names and you can also use those. If we stay with the M1 example, you could also type in 'Crab Nebula' and it would still work! This can be especially useful for stars, which often have multiple names, because they appear in multiple catalogues. It will also warn you whenever you type in a name that cannot be found by SIMBAD.

To search through the database using a specific set of coordinates, you have to choose the 'Coordinates' option and then you can enter your right ascension and declination. The dashboard assumes that you use hours, minutes and seconds for the right ascensions and degrees, minutes and seconds for the declination. For the right ascension, you could also use degrees only, but this will only work if the value is 24 or higher. For the declination you could also use degrees only and this will work anytime.

Note that you do not have to choose an object or a set of coordinates. You can just leave it empty and then the dashboard will only look at the date range you chose.

Using the box and cone search

If you do fill in an object or a set of coordinates, you can choose whether you want to use a box or cone search around it. In Figure 8.3 you can see what using the box search means.

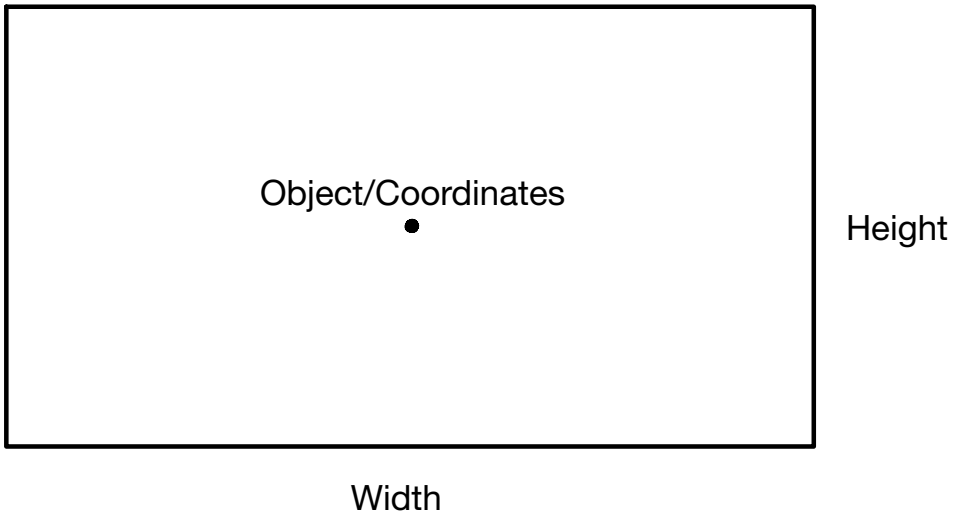


Figure 8.3: This figure displays the way the box search is used.

The box search needs a set of coordinates, which you either provide, or is provided by SIMBAD when you type in the name of an object. Those coordinates are then put in the middle of a box and the width and height of the box are provided by you. They have to be entered in arc seconds and the dashboard provides you with default values which should be good enough. The dashboard will then look at each image and return a table with all of the images in which your coordinates are contained.

The cone search works on the same principle as the box search, except that it uses a cone instead of a box. For the cone search you have to enter a radius in arc minutes and then the

dashboard will return a table with all images that contain your coordinates.

Searching the database

Once you have set everything the way you like, you can choose how many entries you want to see returned. The default value of 100 is probably good enough for a first try and you can always set it higher when needed.

Now you just have to click the 'Search' button and the Blaauw Dashboard will do its job! If it cannot find anything, it will give you an error in the bottom right corner. This means you might have to choose a different date range or that there are no entries in the database for your chosen object or set of coordinates.

8.1.2 Plotting data

Once you have found some data you want to inspect more closely, you can select files by clicking the checkbox and front and then head over to the next tab. The files you selected will show up here in another, smaller table. From here you can select up to three files and plot them, as you can see in Figure 8.4. You can click the three horizontal lines at the top left to collapse the settings, which we will not need here, and the rest of the dashboard will automatically resize the fill up the screen.

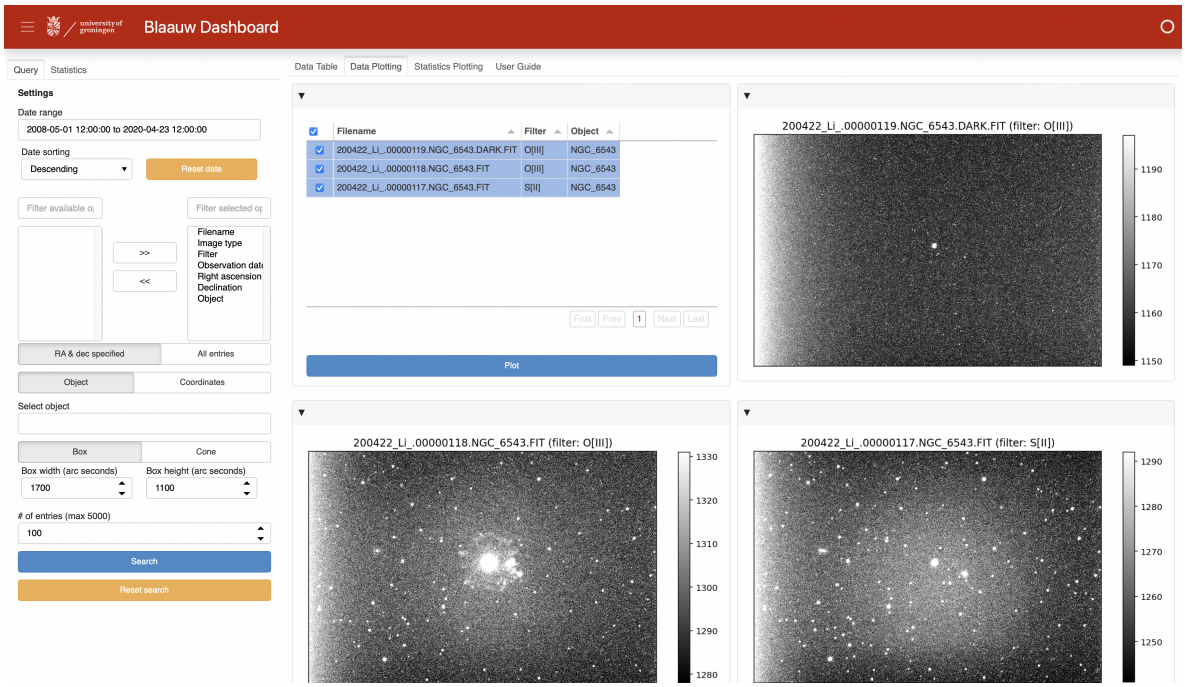


Figure 8.4: This is what the plotting screen of the Blaauw Dashboard looks like, once you have chosen some files to plot.

The title of each plot contains the filename and the filter, so distinguishing them is a bit easier. You can also see a colorbar next to each plot and from this you can get an idea of the minimum and maximum values in the plot. When you want to do more research on the images, you can use these values as a starting point.

8.1.3 Nightly statistics

Finally, on the last tab you can compare some statistics of one night to all previous data. The result of this can be seen in Figure 8.5

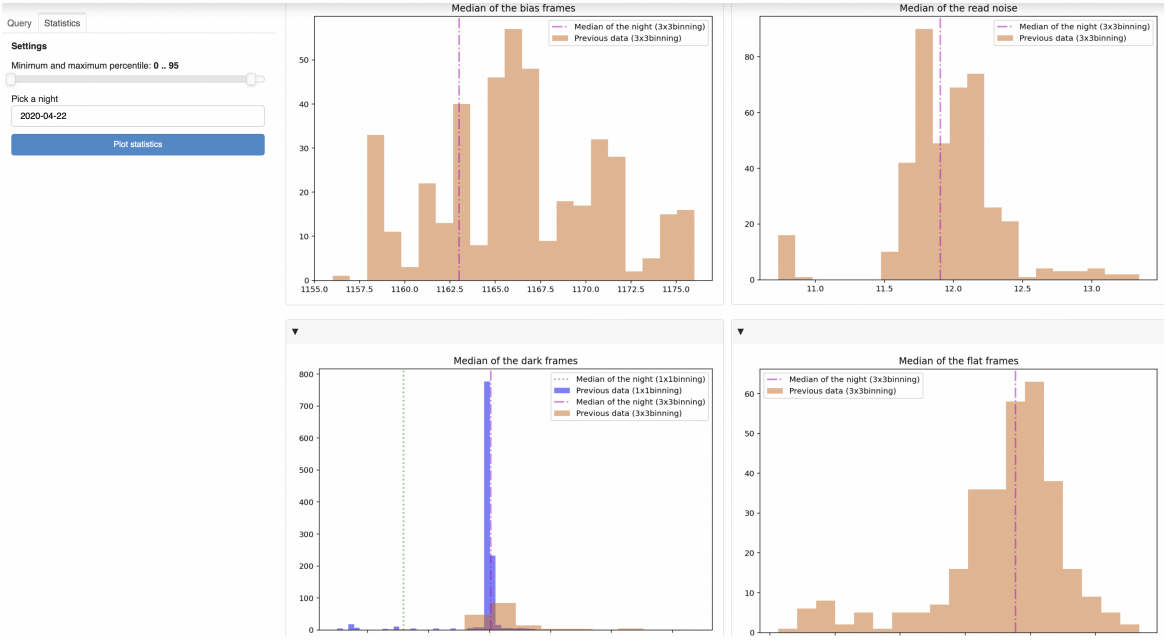


Figure 8.5: The view of the final tab, showing some nightly statistics for the night that was chosen on the left.

To produce the plots just pick a night, choose a percentile range and click the 'Plot statistics' button. This percentile range cuts off the lowest and highest values, because you would end up with illegible histograms otherwise. The default is from the 0th percentile to the 95th percentile, but you can change it to what you want. In the histograms, each binning has a different color, so you can differentiate between them more easily. The median of the chosen night is shown as a vertical line and those also have different colors and different linestyles.

You can use this to compare your calibration files to all previous calibration files. If your median seems to be within the normal range of the previous data, you can probably use it safely. If your calibration files are outside of the normal range, you might want to give them a closer inspection or even use different calibration files altogether.