RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

Laboratory Course Astronomy and Astrophysics

Stellar Statistics

Seminar Report

Marco Bischoff (Matr. No. 370222)

February 18, 2025

Contents

1 Introduction

Identifying stars in astronomical images and estimating their brightness are fundamental tasks in astronomy and are the first steps in many astronomical analyses. Night sky images are often contaminated by other sources of light and contain biases that must be corrected before the images can be analyzed. In this paper, I present approaches to correct for various sources of bias in astronomical images, identify stars in the images, estimate their magnitudes and use these magnitudes to do further analysis. The final goal is to estimate the source-count distribution of stars in the images and compare it to the literature.

1.1 Software Library

The methods presented in this paper are implemented in a software library called `brightest`. It is designed with a focus on reusability and extensibility, such that the methods are not optimized for the specific images used in this paper, but can be applied to any set of astronomical images. The code is written in Python 3.12 and is available on GitHub [2]. The most important libraries used, their versions and their purpose are listed in Table 1. This list represents the current state and may change in the future.

Library	Version	Purpose
`astrometry` [5]	4.1.2	Solving for position
`astropy` [1]	7.0.1	Pixel-to-sky mapping
`astroquery` [4]	0.4.9	Querying the SIMBAD database
`opencv-python` [3]	4.11.0.86	Image processing
`rawkit` [9]	0.6.0	Reading raw images (wrapper for LibRaw)
`scikit-learn` [6]	1.6.1	K-means clustering

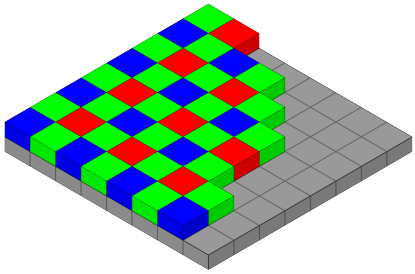
Table 1: Most notable dependencies of the `brightest` library.

2 Image Calibration

The first step in the analysis is to preprocess the raw images to correct for various sources of noise. This is necessary for the star identification, magnitude estimation and other analyses the user might want to perform to give accurate results. In this section, all steps of the preprocessing pipeline are described in detail. First, the observations that I used for testing are described in subsection 2.1. Reading the raw image data is covered in subsection 2.2. The subsequent processing steps are pixel correction (subsection 2.3), vignetting correction (subsection 2.4), interpolation (subsection 2.5) and skyglow correction (subsection 2.6). Finally, the full pipeline with some minor additional processing steps is described in subsection 2.7.

2.1 Observations

To test my processing pipeline and perform the analysis, I used a set of wide-field images taken with a Canon EOS 1200D camera and a 18-55 mm lens, mounted on a tripod. The images were



(a) An example Bayer arrangement of color filters on the pixel array of an image sensor.

(b) The night sky image used throughout this paper, converted from raw to RGB.

Figure 1

taken in the Stadtpark in Aachen and under good weather conditions. Even though images taken under perfect conditions would be ideal for estimating the source-count distribution, they would not be ideal for testing the robustness of the methods. The pipeline is able to handle different types of noise and bias, like faulty pixels (subsection 2.3) and skyglow (subsection 2.6). The camera has a 22.3×14.9 mm CMOS sensor with a raw image resolution of 5202×3465 pixels and a bit depth of 14 bits. I varied the exposure time and ISO settings to capture images with different levels of brightness and noise. All images were taken with an aperture of f/3.5 and a focal length of 18 mm and were saved in the Canon CR2 raw image format. An example image is shown in Figure 1b.

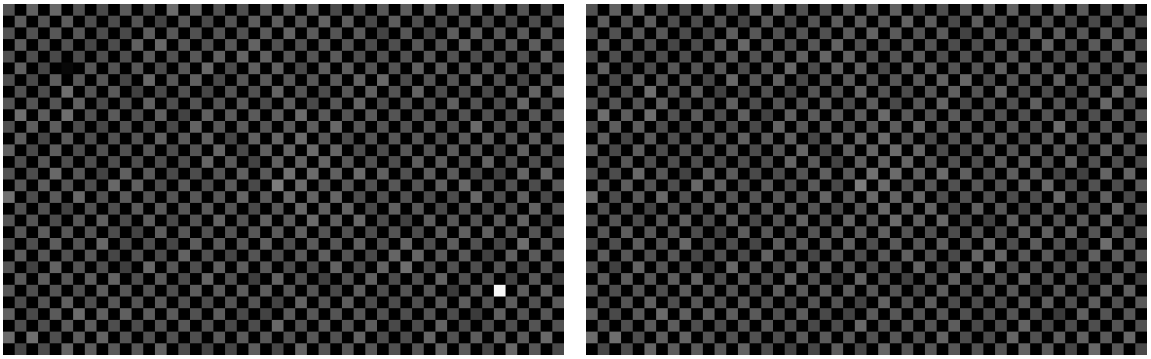
In addition to the night sky images, I also took black and white field images for calibration. The black images were taken with the lens cap on, before and after each set of night sky images, to match the conditions of the night sky images as closely as possible. The white images were taken out of focus, with a homogeneously illuminated white paper covering the entire field of view. To avoid saturation of the sensor, the shutter speed for the white images was set to 5 ms. The black images were used to reduce the noise in dark pixels (subsection 2.7), while the white images were used to correct for vignetting (subsection 2.4).

2.2 Raw Image Data

Raw images store the unprocessed data captured by the camera sensor without any modifications. For the Canon EOS 1200D, this is a matrix with values ranging from 0 to 16383, representing the intensity of light captured. Each pixel has a red, green or blue color filter, corresponding to the Bayer pattern used by the camera. An example of a Bayer arrangement is shown in Figure 1a. The camera that I used has the following pattern:

$$\begin{bmatrix} R & G \\ G & B \end{bmatrix}$$

This pattern is repeated for each 2×2 pixel block in the image. For my analysis, I am only interested in the intensity of light, instead of the color. As 50% of the pixels in the image are green, I only use the green channel for further processing. All methods of my software library can be used equally with the red and blue channels, but this will not be discussed in this paper. I masked the red



(a) Before correction **(b)** After correction

Figure 2: A cutout of the input image with a dead pixel in the top left and a hot pixel in the bottom right, before and after applying the method described in subsection 2.3.

and blue pixels, which leaves a checkerboard pattern of green pixels, as shown in Figure 5. The missing pixels will be filled in using interpolation (subsection 2.5), because the methods from the OpenCV library require a complete image. However, it is important to interpolate at the latest possible stage, to avoid introducing artifacts and additional bias into the image. I used the `rawkit` library [9] to read the raw images, which is a Python wrapper for the LibRaw library and also provides metadata about the images.

I selected a night sky image with a high number of stars and a low level of noise to test the processing pipeline and perform the analysis. The image was taken with an exposure time of 10 s and an ISO setting of 6400 and is shown in Figure 1b. In the following sections, I will refer to this image with the red and blue channels masked as the *input image*.

2.3 Faulty Pixels

A pixel is considered faulty if it does not respond to light as expected. This needs to be corrected before further analysis, as it can reduce the accuracy of finding stars or predicting their magnitudes. I considered two types of faulty pixels. First, a pixel can be permanently damaged due to a malfunction in the sensor, which can cause it to have an incorrect sensitivity to light. Second, a pixel can be faulty due to cosmic rays or other sources of radiation, which can cause it to receive more light than expected. Both types result in a pixel that has a higher or lower intensity than its neighbors.

To identify faulty pixels, I compared each pixel to its neighbors in the night sky input image. Let $I \in \mathbb{R}^{m \times n}$ be the input image and I_{ij} be the intensity of pixel (i, j). I define the convolution kernel $K \in \mathbb{R}^{3 \times 3}$ as

$$K = \frac{1}{4} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Then, the mean and the standard deviation of the neighboring pixels are given by

$$\mu(I) = I * K,$$
$$\sigma(I) = \sqrt{\mu(I^2) - \mu(I)^2},$$

where $*$ denotes the convolution operation. Let t_{hot} and t_{dead} be thresholds for hot and dead pixels, respectively. A pixel (i, j) is considered hot if

$$I_{ij} > \mu(I)_{ij} + t_{\text{hot}} \cdot \sigma(I)_{ij},$$

and dead if

$$I_{ij} < \mu(I)_{ij} - t_{\text{dead}} \cdot \sigma(I)_{ij}.$$

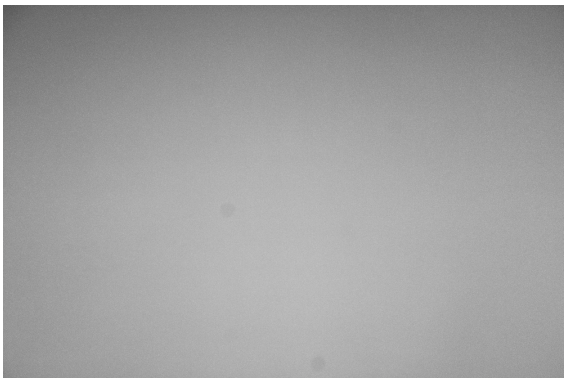
The thresholds can be set by the user. In my analysis, I set $t_{\text{hot}} = 3$ and $t_{\text{dead}} = 1.2$. I chose these values by visually inspecting the images to verify that the method correctly identifies pixels that I would consider faulty. For the input image, this method identified 42 hot pixels and 107 dead pixels. The detected pixels are replaced by the mean of the neighboring pixels. An example of the correction of a dead and a hot pixel is shown in Figure 2.

An alternative approach to identifying the first type of faulty pixels is to recognize bright pixels in the black images and dark pixels in the white images. However, my camera did not have any permanently damaged pixels, so I could not test this method. Also, the method described above should be able to identify both types of faulty pixels, so I did not implement this alternative approach.

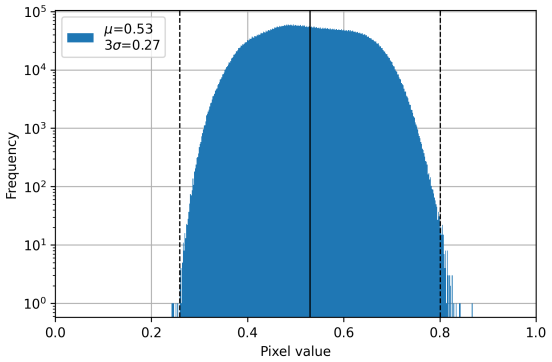
2.4 Vignetting Correction

As light rays from the edges of the lens have to travel a longer distance to reach the sensor, the intensity of light is lower at the edges of the image. This effect is called vignetting and can be corrected by dividing the input image by the white field image. Because the white field image is homogeneously illuminated, the vignetting effect is the main feature of the image. The white field image is shown in Figure 3a, and its green channel histogram is shown in Figure 3b. It is important that the brightest pixels are not saturated to cover the full range of intensities. We need to normalize the white field image, as the center of the vignette should correspond to a correction factor of 1 and the mean and standard deviation of the distribution depend on camera settings, like the exposure time. Therefore, we divide the white image by its 99.999th percentile, which, in this case, corresponds to the 18025th brightest pixel. This is similar to the maximum, but more robust to outliers.

Finally, we divide the input image by the normalized white field image. The result is shown in Figure 4. The vignetting effect is subtle, but visible. In particular, the light pollution is now more pronounced in the bottom right corner, because it was previously masked by the vignetting. I noticed that the white field image has multiple dark spots, which are also visible in the input image, if the contrast is increased. I cleaned the lens on the inside and outside and the mirror of the camera, but the spots remained. I suspect that they are caused by dust on a lens element inside the camera, which is difficult to clean without professional equipment. However, dividing by the white field image also corrects for these spots.

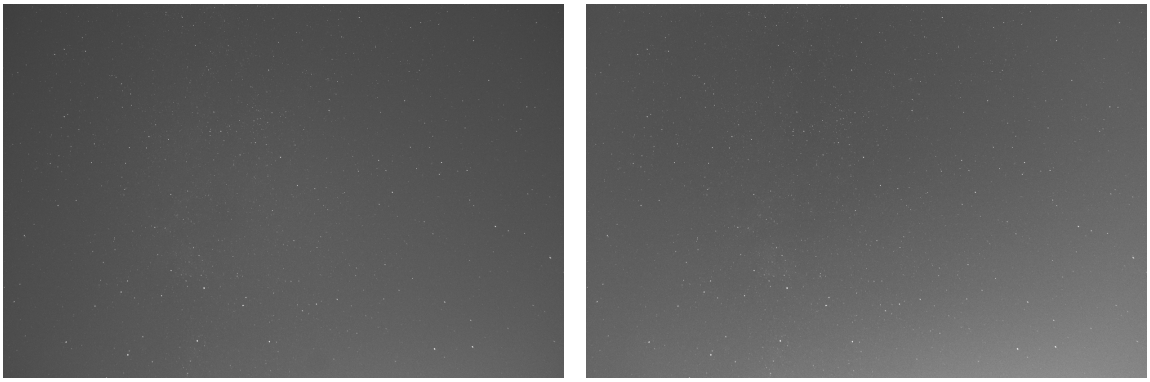


(a) The white field image with reduced brightness and increased contrast for better visualization.



(b) Histogram of the green channel of the white field image with relative pixel values.

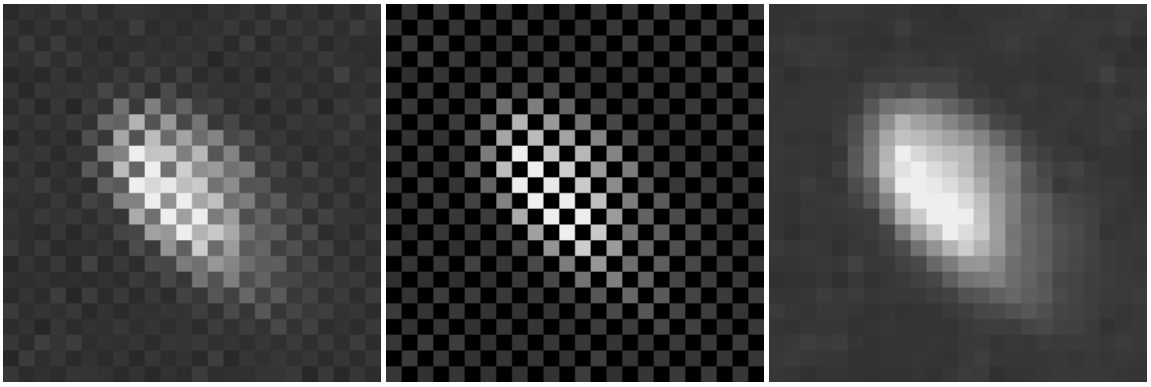
Figure 3



(a) Before vignetting correction

(b) After vignetting correction

Figure 4: The input image before and after applying the method described in subsection 2.4, interpolated for better visualization, as described in subsection 2.5.



(a) Raw image **(b)** After masking **(c)** After interpolation

Figure 5: A cutout of the input image, comparing it at different stages of the pipeline.

2.5 Interpolation

As the next steps in the pipeline require a complete image, we need to interpolate the missing pixels. Iterating over the pixels separately would take several seconds for an image of this size, so I used a convolution with the following kernel instead.

$$K = \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

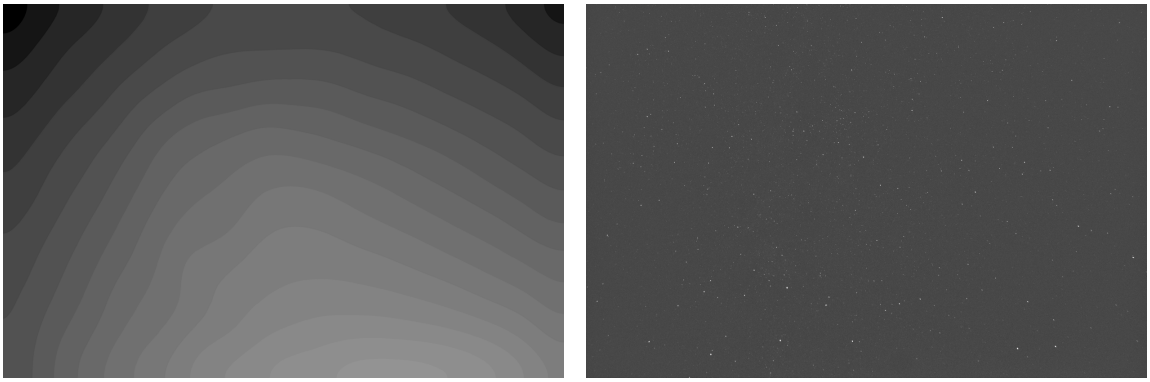
Each missing pixel is replaced by the corresponding pixel in the convolution of the input image with the kernel. The result is shown in Figure 5.

2.6 Skyglow Correction

Another common source of noise in night sky images is light pollution from artificial sources, like street lamps or buildings. This light is reflected by particles in the atmosphere and causes a bright background in the image. I refer to this effect as skyglow. It appears as a smooth gradient in the image with no hard edges or localized features. Stars, on the other hand, are point sources of light and have a sharp intensity profile. Therefore, we can correct for skyglow by subtracting a smoothed version of the image from the original image.

To smooth the image, I used a Gaussian filter with a kernel size of 700×700 pixels and a standard deviation of 200 pixels. The kernel size is chosen to be large, so that all stars are fully smoothed out. The smoothed image is shifted by subtracting its minimum pixel value to avoid negative values. The corrected image is then obtained by subtracting the smoothed image from the input image and clipping the result to the range $[0, 16383]$.

The blurred image and the result of the skyglow correction are shown in Figure 6. The gradient in the blurred image shows lines instead of a smooth gradient. This is because the original blurred image is very dim and, to show it in this paper, I converted it to 8-bit and increased the contrast and brightness. As the original image is 14-bit, the lines are not visible there. After correction, the background of the processed image is much more uniform. However, this method is only beneficial



(a) Input image after blurring, adjusted for better visualization.

(b) The processed image after skyglow correction.

Figure 6: Skyglow correction, see subsection 2.6

for wide-field images with no obstructing objects. If there are trees, buildings or the moon in the image, subtracting the blurred image might worsen the magnitude estimation near the objects. Also, in images with a narrow field of view, the skyglow is much more uniform, making the method unnecessary. The user can disable the skyglow correction in these cases.

2.7 Full Pre-Processing Pipeline

The previous sections describe the major steps of the pre-processing pipeline. An overview of the order of the steps and how they are applied to the input image is shown in Figure 7. The pipeline takes a raw night sky image, a black field image and a white field image as input. The red and blue channels of all images are masked (not shown in the figure for simplicity). After the faulty pixels are corrected, the mean of the black image is subtracted from the input image and the result is clipped to the range $[0, 16383]$. This reduces the noise in the dark pixels because every pixel with a value below the mean of the black image is set to 0. The next steps are the vignetting correction, interpolation and skyglow correction, as described in the previous sections. Lastly, the image is normalized by dividing it by its maximum pixel value. I assume that at least some pixels in the image are saturated. However, different cameras saturate at different values and the previous steps might have reduced the values of the brightest pixels. Also, the bit depth would affect the analysis results, if it would use absolute pixel values. Therefore, I normalize the image to the range $[0, 1]$ to get consistent results across different cameras and bit depths. The output of the pipeline is a pre-processed image that can be used for further analysis. I will refer to this image as the *calibrated image* in the following sections. Histograms of the pixel values of the input and output images are shown in Figure 8. Each step of the pipeline can be disabled by the user, if it is not necessary for the specific image.

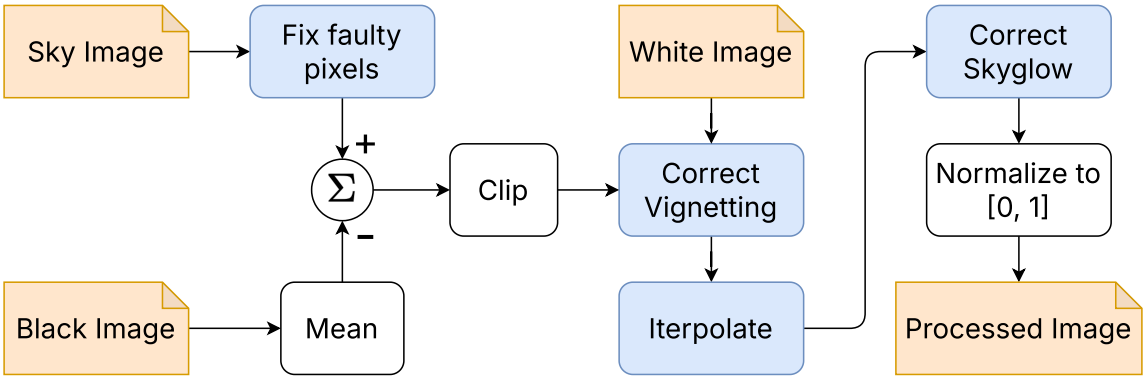
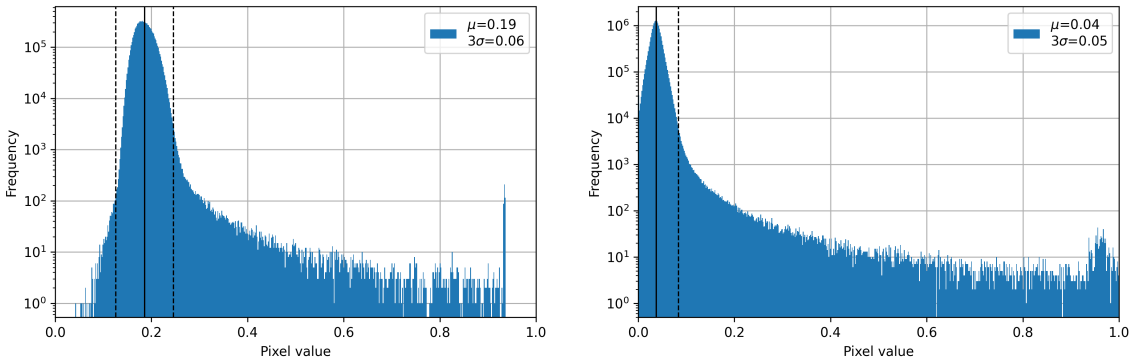
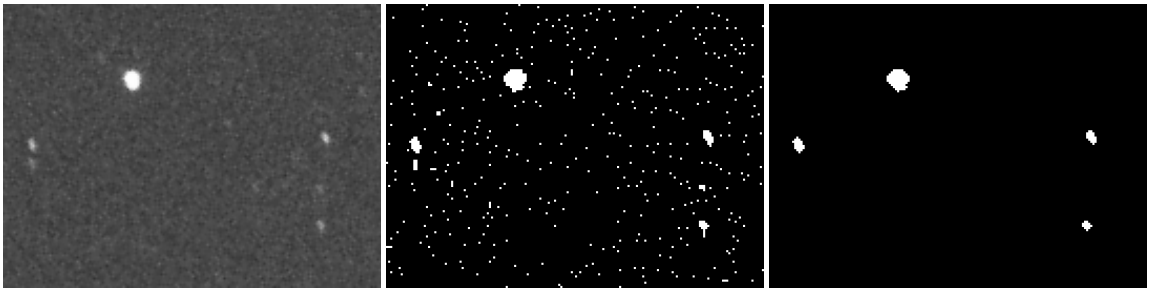


Figure 7: The full pre-processing pipeline for the input image.



(a) Input image **(b)** Output image

Figure 8: Histograms of the input and output images of the full pre-processing pipeline.



(a) Cutout of the input image **(b)** Mask after `floodFill` method **(c)** Mask after Opening operation

Figure 9

3 Analysis

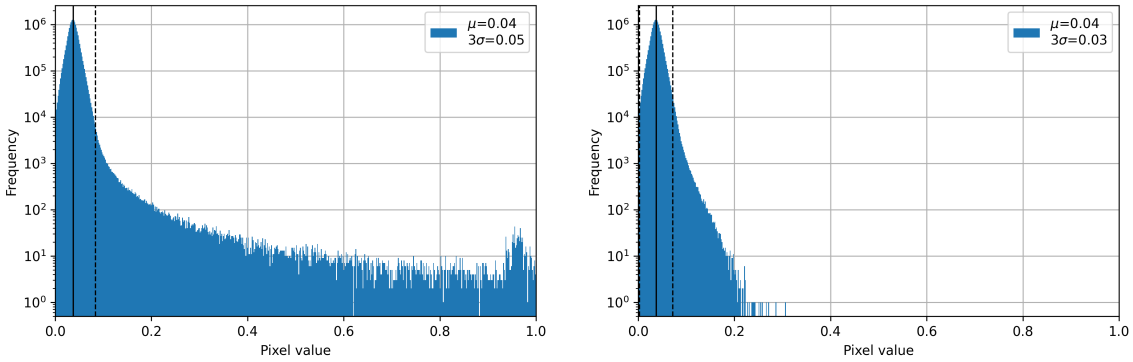
The main goal of the analysis is to estimate the source-count distribution of stars in the images by predicting their magnitudes. I fitted a model to the input image, which can be used by the user to predict the magnitudes of stars in other images. To train the predictor, I needed to obtain the true magnitudes of stars in the input image. This process involves multiple steps, that can be used independently. The first step finds the pixel coordinates of each star in the image (subsection 3.1). These are mapped to the celestial sphere to find the corresponding right ascension and declination (subsection 3.2). The true magnitudes are obtained by matching the sky coordinates to stars in the SIMBAD database (subsection 3.3). The model is then fitted to the labeled list of stars (subsection 3.4). Finally, the distribution of predicted magnitudes is compared to the literature (subsection 3.5).

3.1 Star Identification

Starting from the calibrated image, we need to identify the stars in the image. First, we classify each pixel as belonging to a star or the background. A fast and simple approach is the `cv2.floodFill` method from the OpenCV library [3]. It starts at a seed pixel, fills all connected pixels with a similar intensity and returns the mask of the filled area. A pixel (x, y) with a value $\texttt{src}(x, y)$ is considered to have a similar intensity iff

$$\texttt{src}(x', y') - \texttt{loDiff} \leq \texttt{src}(x, y) \leq \texttt{src}(x', y') + \texttt{upDiff},$$

where $\texttt{src}(x', y')$ is the value of one of the neighboring pixels that is already filled and `loDiff` and `upDiff` are thresholds set by the user. This rule is applied iteratively to all pixels. To fill the background of the input image, I chose the darkest pixel as the seed and set $\texttt{loDiff} = \infty$ and $\texttt{upDiff} = 5$ (for an 8-bit image). A lower value of `upDiff` would result in too much noise being classified as stars, while a higher value would miss faint stars. The result is shown in Figure 9b. We can see that there are many single pixels of noise not being classified as background. To remove them, I applied the morphological operation Opening with a 3×3 kernel using the `cv2.morphologyEx` method from OpenCV. This operation erodes the image and then dilates it, which removes objects that are at most two pixels wide. All other components are preserved. The result is shown in Figure 9c. Finally, I used the `cv2.connectedComponents` method to separate the objects in the



(a) Calibrated image

(b) After removing all star-pixels found by the method in subsection 3.1

Figure 10

mask. I defined the center of the bounding box of each object as the pixel coordinates of the star it corresponds to.

In the calibrated image, the method identified 1001 stars. According to the SIMBAD database, there are 746 stars with a visible magnitude of less than 7.0 and 2330 stars with a magnitude less than 8.0 in the field of view of the image. The ADQL query used to obtain this number is given in Listing 1. If two stars are very close together, the method might classify them as one object. During visual inspection however, I only found two cases of this happening. Stars not being recognized is more commonly due to them being too faint to distinguish them from noise. Figure 10 shows histograms of the calibrated image before and after removing the stars identified by this method. In Figure 10b we can see that almost all pixels identified as background have a relative value of at most 20%.

I also tested other approaches, like template matching. The main problem was to define a general template for stars, that would fit different sizes and brightnesses. This drawback made template matching much more complex and less reliable than the flood fill method.

3.2 Pixel to Sky Mapping

To find the true magnitude of a star in the input image, we need to know its right ascension and declination. This requires mapping the pixel coordinates of the star to the celestial sphere. Therefore, we need to know the orientation of the camera and the field of view of the lens. The website `astrometry.net` [5] allows users to upload an image and returns the world coordinates of the center of the image and other helpful information. An upload of the input image is linked and shown in Figure 16. However, as the `brightest` library is meant to be used without manual intervention or an internet connection, I used the `astrometry` Python library, which only includes the algorithm for astrometric plate solving used by the website. Given a set of pixel coordinates and limits for the field of view, the `astrometry` library returns an `astropy.WCS` (World Coordinate System) object, which can be used to map pixel to world coordinates.

The solver tries to find the best match between the input coordinates and a catalog of stars. I used the recommended built-in dataset based on a subset of the Tycho-2 catalog, available at

`data.astrometry.net`. Additionally, the solver is given a lower and upper limit for the scale of the image, expressed in arcseconds per pixel. My camera has a sensor size of $x \times y = 22.3 \times 14.9$ mm and a focal length of $f = 18$. Then field of view is given by

$$\mathrm{FOV}_x = 2 \arctan\left(\frac{x}{2f}\right) = 63.55^\circ$$
$$\mathrm{FOV}_y = 2 \arctan\left(\frac{y}{2f}\right) = 44.97^\circ.$$

With a horizontal resolution of 5202 pixels, we get scale $= \mathrm{FOV}_x/5202 = 43.98$ arcsec/pixel. For the input image (Figure 1b), the solver located the center of the image at right ascension $20^\mathrm{h}\ 50^\mathrm{m}\ 49.81^\mathrm{s}$ and declination $+60^\circ\ 10'\ 15.02''$. An annotated version of the image, showing the constellations, is shown in Figure 16. Using the returned `astropy.WCS` object, the pixel coordinates of the stars are transformed to sky coordinates.

3.3 Catalog Matching

Now that we know the sky coordinates of the stars in the input image, we can match them to stars in a catalog provided by the user. As mentioned in subsection 3.2, the `astrometry` library already provides a built-in catalog. However, it is only a small subset of the Tycho-2 data so that the solver can run quickly.

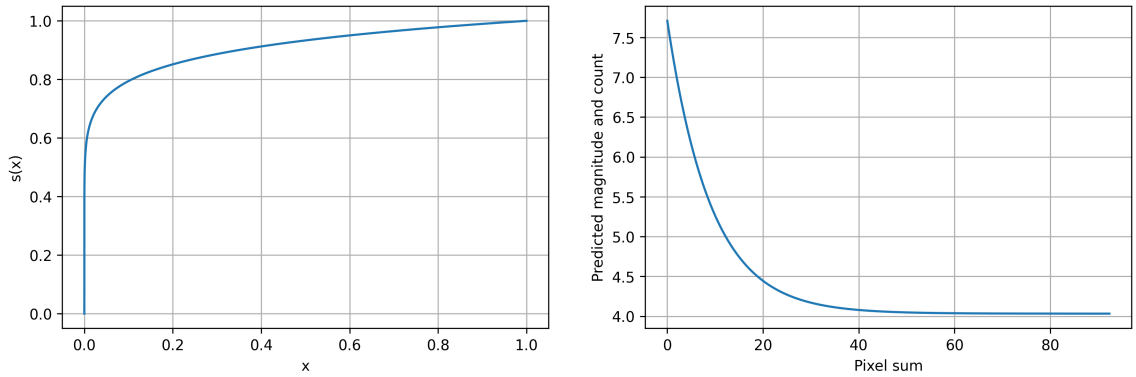
Instead, I queried the SIMBAD database [8]. It is one of the most comprehensive astronomical databases and presently contains information on about 8.9 million stars and 9.3 million non-stellar objects. I queried the database for stars with a visible magnitude of less than 8.0 in the field of view of the input image. Visual inspection showed that stars with a higher magnitude were not distinguishable from noise in the calibrated image. For other images, the maximum magnitude can be set by the user.

Matching the star positions is done by calling the `SkyCoord.match_to_catalog_sky` method from the `astropy` library [1] with the sky coordinates of the stars in the image and the catalog. It constructs a KD-Tree from the catalog and performs a nearest-neighbor search for each star in the image. The result is an index map that assigns a catalog star to each star in the image. The accuracy of this method depends mostly on the accuracy of the solution of the astrometric plate solver (subsection 3.2). The more the coordinates are off, the higher the chance that the nearest neighbor is not the correct match. Therefore, it is important to use a conservative maximum magnitude for the SIMBAD query to keep the size of the catalog close to the number of stars in the image.

3.4 Magnitude Estimation

The magnitude of a star is a measure of the intensity of light it emits on a reverse logarithmic scale. We want to predict the magnitude of the stars in the image for further analysis. The pixel mask that was created in subsection 3.1 gives us a set of pixels with values in $[0, 1]$ for each star. This serves as the basis for the estimation and will be summed up to get the total intensity of the star. As the `floodFill` method selects pixels based only on the intensity difference to their direct neighbors, the selection of pixels with a medium value is inconsistent.

To get a more accurate selection, I first scaled the image using the function $s(x) = x^{0.1}$ (see Figure 11a). This spreads out the values of the dimmer pixels to make them more distinguishable.



(a) Scaling function $s(x) = x^{0.1}$

(b) Fitted exponential function $f(x)$

Figure 11

Then, I applied k-means clustering with $k = 2$ to the pixels within the bounding box of each star. Pixels belonging to the cluster with the higher mean are considered star-pixels, and background-pixels otherwise. The clustering separates bright from dark pixels more consistently without any predefined factor or shape. I chose centroid-based clustering, because it can guarantee two clusters. For the implementation, I used the `KMeans` class from the `scikit-learn` library [6].

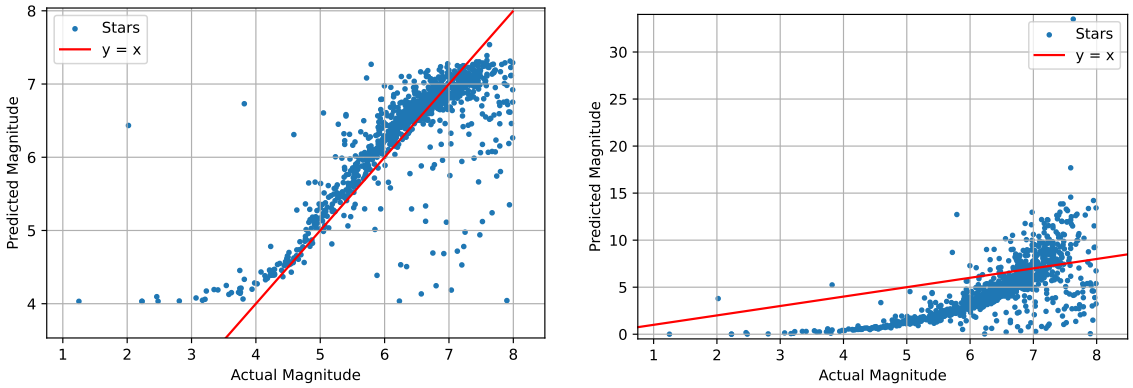
After refining the mask, I fitted an exponential function to the data. It takes the sum of pixels, that were classified as star-pixels, as input and returns the magnitude. The function is defined as

$$f(x) = a \cdot \exp\left(-b\left(\frac{x}{30} - c\right)\right) + d,$$

where a, b, c and d are the parameters to be fitted. The factor 30 is used to ensure convergence of the optimization algorithm. The parameters are fitted using the `curve_fit` method from the `scipy` library [7]. The optimal values of the parameters are $a = 0.47$, $b = 3.28$, $c = 0.63$ and $d = 4.03$ and the resulting function is shown in Figure 11b.

I tested other functions, like a polynomial, hyperbolic, logarithmic and variations of trigonometric functions. The function $g(x) = a \cdot \log(1/x + b)$ would be the most appropriate in the physical sense, as it is a convex function with $\lim_{0 \leftarrow x} g(x) = \infty$ which can produce negative values. However, having many points with a low pixel sum meant that the fitted function assigned the same magnitude to all brighter stars. A comparison of the magnitude predicted by the logarithmic function to the true magnitudes is shown in Figure 12b. I settled on the exponential function (Figure 11b) mentioned above, because it produced the lowest mean squared error. For a pixel sum of 0, the function returns a magnitude of 7.5, which is the maximum magnitude of the stars in the image. Any star with a higher magnitude is not distinguishable from noise. The function stagnates towards a magnitude of 4 from a pixel sum of 50 onwards. This is likely due to a lack of training data and saturation of the sensor for a majority of the pixels.

The mean squared error of the fit without the clustering step was 0.366. With the clustering, but without the scaling by $s(x)$, the error was 0.349. The final error after applying both mask refinement steps was 0.334, which is a reduction of 9.6% compared to the initial error. The MSE of the logarithmic function $g(x)$ was 0.452. A comparison of the predicted magnitudes to the



(a) Prediction with the exponential function $f(x)$ **(b)** Prediction with the logarithmic function $g(x)$

Figure 12: Comparison of the predicted magnitudes to the true magnitudes.

true magnitudes is shown in Figure 12. The true magnitudes are on the x-axis and the predicted magnitudes on the y-axis, with the red line $y = x$ for reference. The function is not perfect, but it is a good approximation for the majority of stars. The outliers could be due to the stars being matched to the wrong catalog star or two stars being recognized as one object. Also, very bright stars are not estimated well, likely because they are saturated in the image.

3.5 Source-Count Distribution

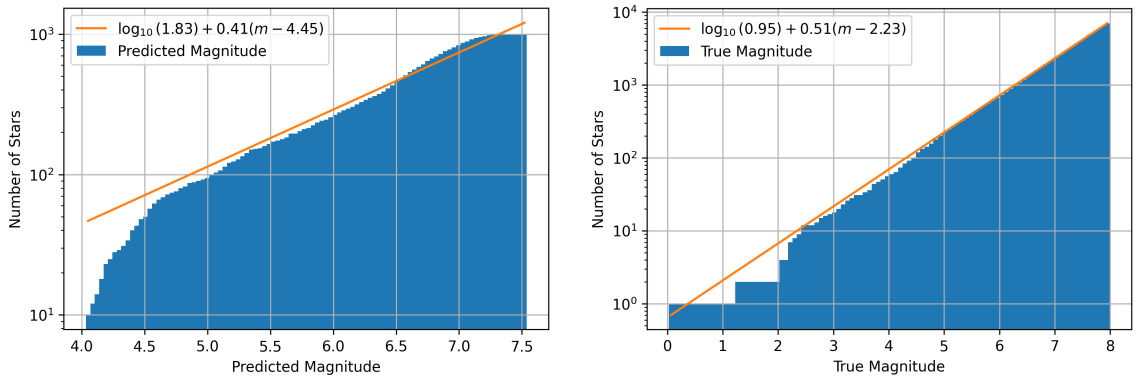
A source-count distribution is a cumulative distribution of the number of sources (N) brighter than a given flux density (S). It corresponds to the volume integration of those stars which generate a received radiation flux of $S > S_0$ for a given flux density S_0. Using the distance-luminosity relation $L(S) = 4\pi r^2 S$, where r is the distance to the source, and integrating over the luminosity function $\Phi(L)$, we can derive the general form of the source-count distribution as

$$N(> S) = N_0 \cdot \left(\frac{S}{S_0}\right)^{-\frac{3}{2}},$$

where N_0 and S_0 are normalization constants. The slope of the distribution is determined by the power-law index -3/2. With the definition of the magnitude $m - m_0 = -2.5 \log_{10}(S/S_0)$, we can express the source-count distribution in terms of the magnitude as

$$\log_{10}(N(< m)) = \log_{10}(N_0) + 0.6 \cdot (m - m_0). \tag{1}$$

We can use this relation to verify the accuracy of the magnitude estimation. The distribution of the magnitudes of the stars in the calibrated image, estimated by the approach in subsection 3.4, is shown in Figure 13a. For comparison, the same plot is shown for the stars obtained from the SIMBAD database using the query in Listing 1 in Figure 13b. I fitted the function in Equation 1 to the cumulative distribution, with the slope, N_0 and m_0 as free parameters. From the fit, I obtained a slope of 0.41 for the predicted magnitudes and 0.51 for the true magnitudes, which is close to the expected value of 0.6. One of the main reasons for the deviation is likely the low accuracy of



(a) Prediction with the exponential function $f(x)$ **(b)** Prediction with the logarithmic function $g(x)$

Figure 13

the predictor for bright stars. As no star receives a magnitude of less than 4, the distribution is inflated at the bright end, which contributes to the lower slope. Also, the distribution would likely approach a slope of 0.6 more closely, if the distribution would include stars with a magnitude higher than 8.0. I suspect this is the reason for the lower slope of the true magnitudes as well.

4 Discussion

The goal of this project was to develop a program for the analysis of night sky images and use it to estimate the source-count distribution of stars. The program consists of the calibration of the input image, the identification of stars, matching them to a catalog and estimating their magnitudes.

Judging by the results, the calibration of the image was successful. The faulty pixels were corrected, the vignetting effect was removed, and the skyglow was reduced. I think the biggest improvement would be to verify that the calibration works reliably under different conditions. How accurate is the method for different cameras, lenses, ISO settings and exposure times? How well does the method handle diffraction spike patterns? What if objects obstruct the view or the image is taken in the presence of the moon? I was not able to test these questions, but I am confident that my approach performs well for wide-field images with no obstructing objects.

The identification of stars was also successful. The method was able to find the majority of the stars that are visible in the image. The main problem was the distinction between stars and noise. I manually found stars that were not recognized, but they were faint and hard to distinguish from the background. Also, if the image is very blurry or out of focus, the algorithm might not work well. It would be interesting to combine the flood fill method with the template matching approach for small objects. This could help to identify stars that are too faint for the current method. The subsequent steps were very reliant on the accuracy of solving for the camera orientation. It would be possible to improve the robustness of the solver by taking the brightness of the stars into account. Also, the solver could be combined with the catalog matching step to make the intermediate step unnecessary. When estimating the brightness using the pixel sum, a big reason for inaccuracy was that the sensor was saturated for many of the bright stars. This introduced a

different slope in the data, which the exponential function could not capture. It could be modeled by a more complex function, but it would need to avoid overfitting the dimmer stars. The magnitude estimation does not work for stars with a magnitude lower than 4.0 or higher than 8.0, due to the lack of training data. The source-count distribution was estimated by fitting a power-law to the cumulative distribution of the magnitudes. The slope of the distribution was close to the expected value of 0.6, but the distribution was inflated at the bright end. I suspect this was due to the low accuracy of the predictor. However, the shape of the distribution does fit the expected form.

In conclusion, the program is a good starting point for the analysis of night sky images. It is able to calibrate images and give a solid estimate of how bright the stars are. The code is modular and can be used as a library or as a basis for further development.

References

[1] Astropy Collaboration, Adrian M. Price-Whelan, Pey Lian Lim, et al. The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. *APJ*, 935(2):167, August 2022.

[2] Marco Bischoff. Brightest: A processing and analysis tool for astronomical images. `https://github.com/bischoff-m/brightest`, 2025.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] A. Ginsburg, B. M. Sipocz, C. E. Brasseur, P. S. Cowperthwaite, et al. astroquery: An Astronomical Web-querying Package in Python. *AJ*, 157:98, March 2019.

[5] D. Lang, D. W. Hogg, K. Mierle, M. Blanton, and S. Roweis. Astrometry.net: Blind astrometric calibration of arbitrary astronomical images. *AJ*, 137:1782–2800, 2010. arXiv:0910.2233.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[8] M. Wenger, F. Ochsenbein, D. Egret, P. Dubois, F. Bonnarel, S. Borde, F. Genova, G. Jasniewicz, S. Laloë, S. Lesteven, and R. Monier. The simbad astronomical database: The cds reference database for astronomical objects. *Astronomy and Astrophysics Supplement Series*, 143(1):9–22, April 2000.

[9] Sam Whited, Cameron Paul, et al. Rawkit: Ctypes based libraw bindings. `https://github.com/photoshell/rawkit`, 2025.

A Appendix

A.1 ADQL Query for the SIMBAD Database

```
SELECT COUNT(*)
FROM basic JOIN flux ON oidref = oid
WHERE otype = 'Star..'
      AND filter = 'V'
      AND flux < 8.0
      AND CONTAINS(POINT('ICRS', RA, DEC), BOX('ICRS', 312.708, 60.171, 64, 45)) = 1;
```

Listing 1: ADQL query for the SIMBAD database to find stars in the field of view of the input image. "Star.." stands for "star or any sub category of star". The query can be executed at `https://simbad.cds.unistra.fr/simbad/sim-tap/`.

A.2 Histograms of Predicted Magnitudes

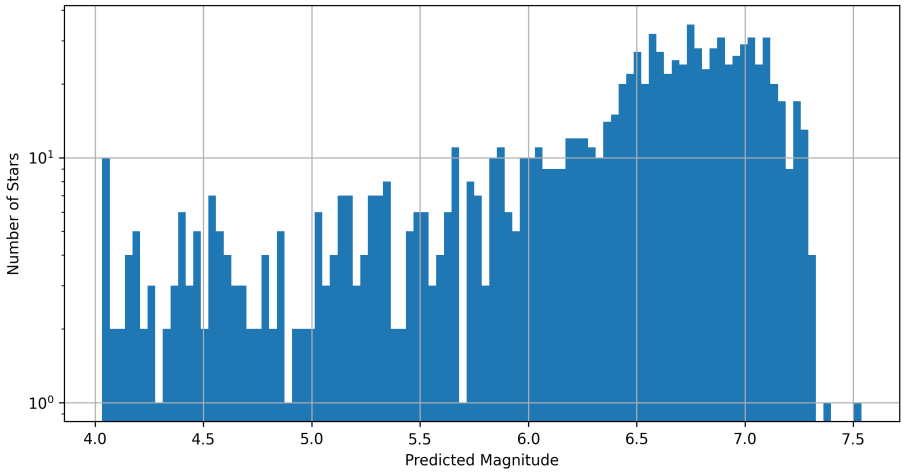


Figure 14: Histogram of the predicted magnitudes of stars in the input image.

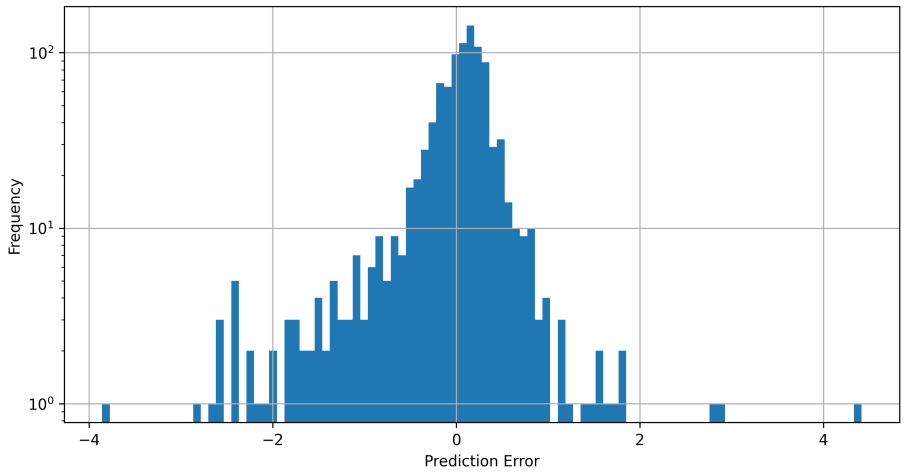


Figure 15: Histogram of the prediction error of the magnitudes of stars in the input image.

A.3 Output of the Astrometry.net Service

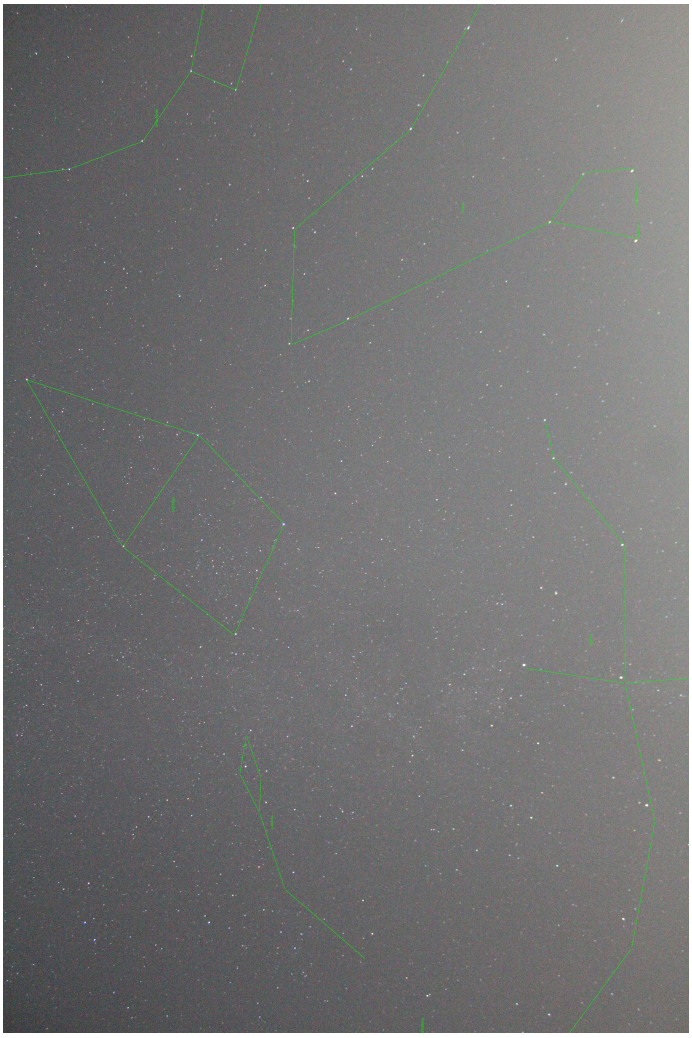


Figure 16: Output of the Astrometry.net service for the input image. Find the results online at `https://nova.astrometry.net/user_images/11597239#annotated`.