university of groningen / faculty of science and engineering / bernoulli institute

Star Removal in Astronomical Images

Bachelor's Project Computing Science

Sander Zeeman, s3808777, s.zeeman@student.rug.nl
First supervisor: dr. M.H.F. Wilkinson
Second supervisor: Caroline Haigh MSc

August 31, 2021

Acknowledgements

I would first like to extend my gratitude to everyone who played a part in the completion of this project.
First, this thesis would not have been possible without the excellent guidance of my supervisor: dr. Michael Wilkinson. Our weekly meetings greatly helped me in understanding the theory that was required for the development of this project. Additionally, many great suggestions were made during these meetings, which I greatly appreciate.
Next, I would like to thank Caroline Haigh for acting as my second supervisor in this project.
Finally, I want to extend my gratitude to my family and friends, who supported me in my efforts to make this project reach a successful conclusion.

Abstract:

As increasingly many images get taken of outer space, astronomers are facing great difficulties when trying to identify any significant objects in these images. Contrast stretching is used to highlight these objects. However, stars are very bright point sources, which nearly nullifies the effectiveness of contrast stretching. In this thesis, we explored the idea of building a star removal software to solve this issue. We applied state of the art object detectors to identify all objects in an image, then attempted to filter all stars from this set of objects by thresholding on numerous attributes of an object. We determined that our software, while not ready to be adopted in the standard image processing pipeline used by astronomers, in combination with some manual adjustments, will allow for contrast stretching to have improved results. Additionally, the software we developed can be used as a framework, which future research could build upon.

Contents

1 Introduction

In the past decades, great advancements have been made in the field of astronomical imaging. While many barriers have been overcome, many issues still remain to this day. Before describing one of the main issues in detail, let us first take a step back and display how much the field of astronomy has grown since the creation of the very first telescope.

The following quote is from an article that is part of the Library of Congress (2014).

"The first telescopes were created in the Netherlands in 1608. Spectacle makers Hans Lippershey & Zacharias Janssen and Jacob Metius independently created telescopes at this time. These telescopes were created for uses on Earth itself, rather than for observing outer space. These initial telescope were capable of magnifying objects three times."

It shows the beginnings of the instruments that astronomers use on a daily basis. Today, telescopes are capable of taking high quality pictures of outer space in bulk. This means astronomers work with large databases full of these images.

Astronomers are always working to make new discoveries regarding our universe. After all, discovering a new galaxy, star or nebula could lead us to even greater discoveries surrounding our universe. However, this is where astronomers run into an issue. Nebulae or galaxies might be difficult to spot, as they have a relatively low surface brightness. This issue could be solved by stretching the contrast of the image. As described by Sahidan et al. (2008): "Contrast stretching (often called normalization) is a simple image enhancement technique that attempts to improve the contrast in an image by 'stretching' the range of intensity values it contains to span a desired range of values, the full range of pixel values that the image type concerned allows." This would make our celestial objects with low surface brightness much more clear.

However, there is an issue with this solution. Stars have a much higher surface brightness than nebulae or galaxies. Because there are a lot of stars in outer space, it is extremely likely that any astronomical image will contain multiple bright stars. This will cause the dynamic range of our image to become very large. To demonstrate this issue, a histogram of all pixel values from Figure 3.1 can be seen in Figure 1.1.

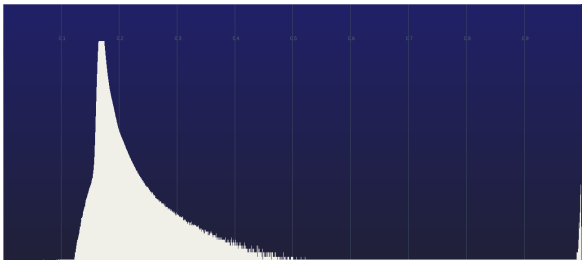

Figure 1.1: A histogram of all pixel values in Figure 3.1.

Apart from the constant background value of ± 0.1 that could be removed, linear contrast stretching would not improve the image as much as we had hoped. After all, there are many pixel values close to 1.0. This large dynamic range, caused by mostly stars, will nearly nullify the improvement that contrast stretching could bring. This is demonstrated in Figure 1.2, where logarithmic contrast stretching was performed on the original image in Figure 3.1. The concept of logarithmic contrast stretching will be explained in detail in Section 2.1.

The main reason that stars are so harmful to logarithmic contrast stretching, is that this method aims to make the faint objects more noticeable, while keeping the bright sections of the image from taking over. However, stars are not captured as exact point in the image. Stars will be become more dim around the edges and merge into the background. The point spread function (PSF) describes this effect. This has

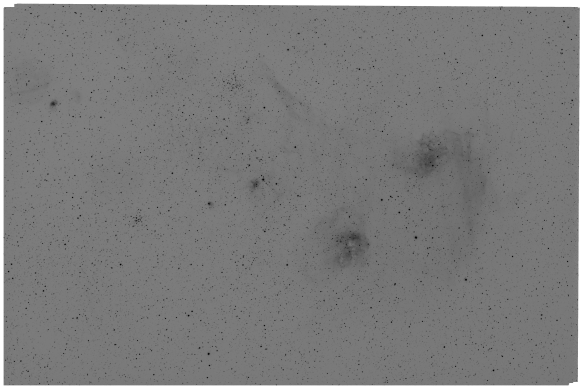

Figure 1.2: Figure 3.1, after performing logarithmic contrast stretching.

as a result that these dim sections around each star will also be stretched in the result image. This means the edges of the star will become far more noticeable, which will cause stars to grow.

This is the problem we will be solving throughout this thesis. We hypothesise that removing the stars from an image will make contrast stretching noticeably more effective. As such, we will be providing an answer to the following research question:

How can we improve the current state of available "star removal" software?

In order to answer this question, we will have to answer multiple sub-questions, related to the steps we will be taking.

- How do connected morphological image processing techniques compare to the existing tools, which use machine learning, in terms of effectiveness, speed and adaptability?
- Which image inpainting methods are best suited for astronomical images?

In this thesis, a solution to the formerly described issue will be proposed. Existing object detection algorithms will be rewritten in C++ and extended with the ability to filter the stars from all detected objects. These stars will then be removed from our image. Finally, the missing areas will be filled in by a variety of image inpainting algorithms. These results will be showcased, after which a recommendation will be made, based on which algorithm produced the optimal results.

Let us start by introducing the works that this research has built upon. In section 2, we will go through the concepts that we applied in this thesis. Next, we will discuss our implementation of the star removal software in section 3. After this, we will display our results in section 4. In section 5, we will discuss the results we obtained, as well as compare our solution to existing solutions. Additionally, we will mention any shortcomings of the software we developed. We can conclude our thesis by forming a single, compact conclusion in section 6 and describing any possible future work on this topic in section 7. Finally, this thesis will be concluded by our bibliography.

2 Related Work

Before we can discuss our implementation, we have to introduce the related research that this thesis is based on. Let us start by describing contrast stretching, as mentioned in Section 1. Afterwards, we will discuss the related work in the order in which it will be brought up in the implementation section, starting with the MaxTree.

2.1 Contrast stretching

Before defining contrast stretching and explaining why it is as useful as it is, we must first define contrast itself. We will be defining this concept similarly to Anand (2017):

"Contrast is the difference in pixel values of two different objects at the same wavelength. It can be defined as the ratio of maximum intensity to the minimum intensity over an image.

$$C = \frac{L_{max}}{L_{min}}$$

where C is the contrast, L_{max} is the maximum intensity value present in an image and L_{min} is the minimum intensity value present in an image."

Astronomical images often don't use the full range of pixel values. Instead, many pixel values surround some pixel value, with the outliers being very close to 0 or 1, this can also be noticed in Figure 1.1.

To combat this, contrast stretching is often used to redistribute the pixel values to make use of the full range. This could be done linearly, however, astronomers are more interested in non-linear stretches. Specifically, logarithmic contrast stretching is often used.

The benefit of logarithmic contrast stretching is that it increases the contrast between darker and brighter sections of an image. This allows astronomers to more easily separate objects from the background. The lower brightness values are given a greater relative contrast than the brighter values, this allows the darker features of the images to be easily separated from the rest of the image, while bright values are made less apparent, while still being recognisable (Anand, 2017).

As was described in the introduction, logarithmic contrast stretching will cause stars to grow, by stretching the faint edges of a star. This tends to do more harm than good, as can be seen in Figure 1.2. Because of this effect, we hypothesise that the absence of stars would cause contrast stretching to have more effective results. This is why we are interested in removing the stars from astronomical images.

2.2 The MaxTree data structure

To introduce the MaxTree data structure, we must first define "connected components". Connected components are sets of pixels with the exact same pixel values, where all pixels are connected through either 4-pixel or 8-pixel connectivity. Let us now define the MaxTree, similarly to Salembier et al. (1998).

A MaxTree consists of a set of nodes. Each node represents a connected component, with links between parent and child nodes. Consider the example given in Figure 2.1.

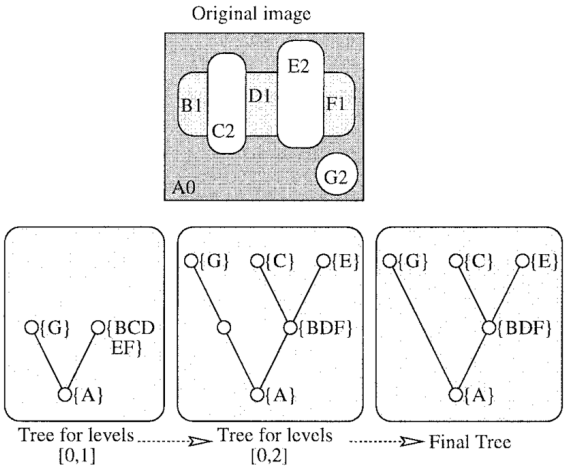

Figure 2.1: MaxTree representations. (Figure retrieved from Salembier et al. (1998))

Three layers of pixel values are represented in this figure, represented by the digit after the letter. All objects on the same layer will consist of the same pixel values. Below the example image, we see how the MaxTree is built, let us focus on the section labelled "Final Tree".

As one can see, there is only one object on layer 0, namely the background, node A. From here, we notice two separated objects, node $G2$ and a combination of the remaining nodes. Node $G2$ is on layer 2, so we can skip past the layer 1 node on this branch, this node will remain empty and can be ignored. (Note that this differentiates the MaxTree from the similar data structure: "Component tree" (Jones, 1999), which does not ignore these nodes.) We will then create a child node for object $G2$. On the remaining object, we perform the same operations. We notice that layer 1 is the lowest layer inside this object, so we consider this the root of this "sub-tree". Nodes $C2$ and $E2$ are the only objects on a higher layer, so objects $B1$, $D1$ and $F1$ are combined into the root node. Finally, separate child nodes are created for objects $C2$ and $E2$. Adding this "sub-tree" as a child to the main tree will then leave us with the structure labelled "Final Tree" in Figure 2.1.

If we were to represent an image as a MaxTree, we would gain knowledge on the connectivity of pixels inside the image. As one can imagine, this knowledge greatly assists us when attempting to detect objects in an image.

Next, let us discuss MTObjects [1], which implemented a MaxTree as part of an object detector for astronomical images.

2.3 MTObjects

MTObjects, the MaxTree-based faint object detector, has been updated and improved many times throughout its lifetime. We will discuss the implementation by Caroline Haigh, which was based on the work done by Teeninga et al. (2015).

Our extension to this program will be described in full detail, from start to finish, in Section 3. This will include the algorithms implemented in the existing codebase. Therefore, we will choose not to extensively discuss these algorithms here.

Instead, let us consider this code a blackbox for now and look at its input and resulting outputs, while discussing why this program would assist us in reaching our goal.

Consider Figure 2.2. The image on the top is the original, where the colours have been inverted to increase visibility of the astronomical objects. MTObjects has performed an object detection, then marked all pixels belonging to an object a different colour. The result can be seen on the bottom of Figure 2.2. One can

[1] `https://github.com/CarolineHaigh/mtobjects`

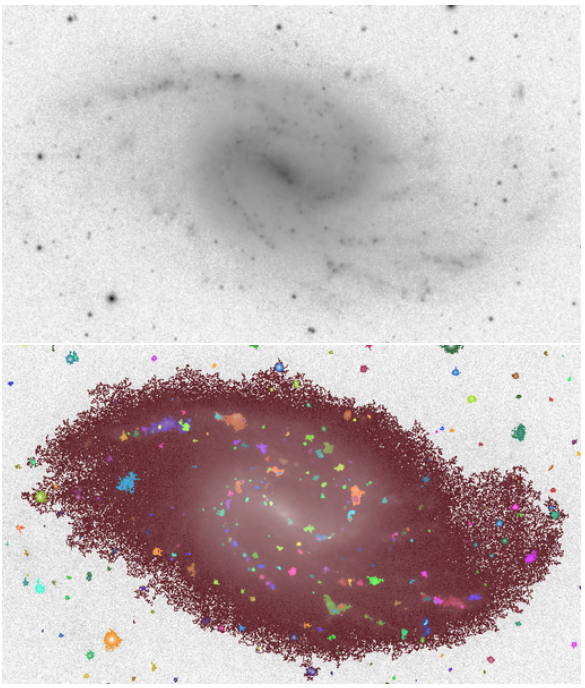

Figure 2.2: MTObjects applied on an astronomical image. (Retrieved from Teeninga et al. (2015))

immediately tell that many tiny, faint objects are being detected by this program.

If we could identify a method of separating the stars from all other objects that were detected by this program, this would benefit us greatly in the pursuit of a star removal algorithm. As described by Haigh et al. (2021): "MTObjects achieves the highest scores on all tests for all four quality measures, whilst SExtractor obtains the highest speeds.". From this, we can tell MTObjects has sacrificed some execution speed, in favour of a higher accuracy, which reassures us that this program will be suitable for our project.

2.4 Image inpainting techniques

We have now discussed the main research that we will be building upon in this thesis. However, we have not yet mentioned any techniques that will be useful to our project after we have removed the stars.

When we have successfully removed a section of an image, we want to restore this section to the best of our ability. In order to do this, we need to know what the space behind the star looked like. This is an impossible task with the information we are given. Therefore, our best option is estimation.

Specifically, image inpainting algorithms. Inpainting algorithms come in all shapes and sizes. Some are extremely fast, yet also imprecise. While others take a long time to run, but return very accurate results. We will be discussing two algorithms. One of which is a well known simple technique. The other is more advanced and shows great potential for noise-heavy images. Let us discuss them, one by one.

2.4.1 Linear interpolation

Interpolation is quite a simple technique. We choose to apply linear interpolation, as this should be an extremely fast operation. Because of this, it should give us a good comparison for the advanced method we will discuss afterwards.

Linear interpolation assumes the pixel values, between two known pixels, will follow a linear trend. An example can be seen in Figure 2.3

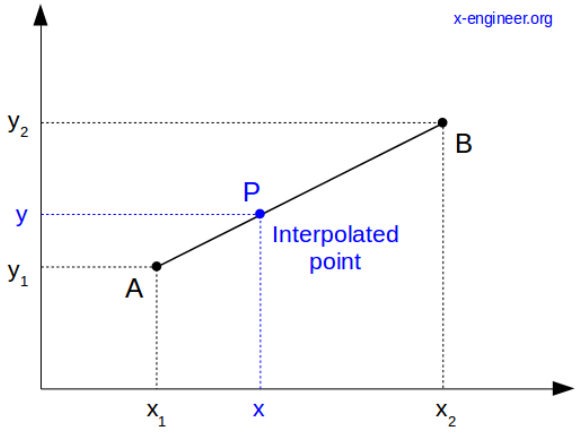

Figure 2.3: Linear interpolation (Retrieved from `https://x-engineer.org/undergraduate-engineering/advanced-mathematics/numerical-methods/linear-interpolation-and-extrapolation-with-calculator`)

Here, A at (x_1, y_1) and B at (x_2, y_2) are two points with a known value. Following linear interpolation, the value at x should be y.

Linear interpolation follows Equation 2.1:

$$y = y_1\left(1 - \frac{x - x_1}{x_2 - x_1}\right) + y_2\left(1 - \frac{x_2 - x}{x_2 - x_1}\right) \quad (2.1)$$

As one can tell, we simply assign weights to the values at points A and B. Specifically,

the weight of point A will be determined by $1 - \frac{x-x_1}{x_2-x_1}$, which is the distance on the x-axis from P to B, scaled down to a value in $[0, 1]$. Similarly, the weight of point B will be determined by $1 - \frac{x_2-x}{x_2-x_1}$, which is the distance on the x-axis from A to P, also scaled down to a value in $[0, 1]$.

2.4.2 Navier-Stokes based

Finally, let us discuss the inpainting method by Bertalmio et al. (2001), based on the Navier-Stokes equations. The Navier-Stokes equations govern incompressible Newtonian fluids. Because of this, we suspect that an inpainting method based on these equations should lead to a fluid transition between known pixel values and inpainted values.

As an example, see Figure 2.4, where four images have been filled with words in an attempt to showcase Navier-Stokes inpainting.

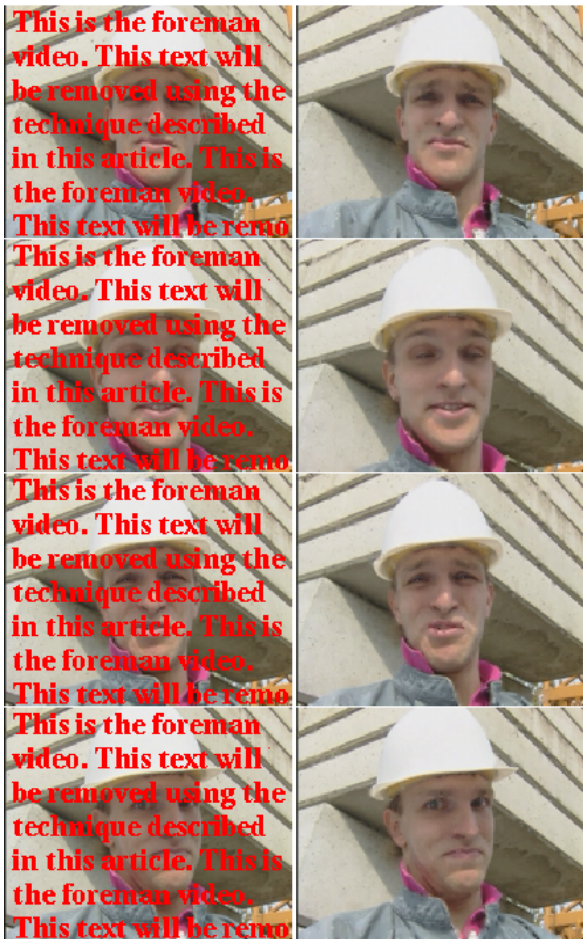

Figure 2.4: Four images where the lettering is removed by the NS inpainting algorithm. (Retrieved from Bertalmio et al. (2001))

As one can tell, the algorithm did an impressive job at reconstructing these images. This reassures us that this inpainting method will be a good choice of advanced inpainting algorithm.

The exact mathematics behind this algorithm are far beyond the scope of this thesis. However, we will describe the algorithm, following the OpenCV page[2], to give a clear idea of the parameters that the algorithm requires and how the algorithm reconstructs an image.

The algorithm accepts an image, in the form of a Mat[3] object of floats, a mask, in the form of a Mat object of booleans, an empty Mat object, which will be filled with the output and an integer radius, which describes the radius of the disk which is used to set up a stream function, as described by Bertalmio et al. (2001).

By combining the descriptions in Bertalmio et al. (2001) and the OpenCV documentation page, we can explain the algorithm as follows:

The Navier-Stokes algorithm utilises partial differential equations. It first travels along the edges of an area that has yet to be inpainted. Here, methods from fluid dynamics are used to match gradient vectors at the boundaries of this region. Additionally, the isophotes are continued. Isophotes are defined as follows, by Patrikalakis et al. (2009): "Isophotes are curves of constant light intensity on a surface". Once they are obtained, the removed pixels are filled in to minimise the variance in that area.

2.5 Starnet++

Now that we have discussed the research that we will be basing our research on, we can introduce Starnet++[4].

Starnet++ is an application, developed by Nikita Misiura. This application applies deep neural networks to remove stars from an astronomical image. We will not dive into the exact workings of this application in this thesis, as this would require us to give detailed explanations of neural networks, which is beyond the

[2] https://docs.opencv.org/4.5.2/df/d3d/tutorial_py_inpainting.html
[3] https://docs.opencv.org/4.5.2/d3/d63/classcv_1_1Mat.html
[4] https://github.com/nekitmm/starnet

scope of this thesis. However, readers with a background in deep learning are recommended to view the GitHub page in the footnotes, as the application is documented well, with references to other interesting papers.

Let us view the results Starnet++ achieves. As Starnet++ does not allow for the reading of 32 bit FITS images, we will be using a different image to the one we will use to test our application. The image that was used can be viewed in Figure 2.5.

Figure 2.5: The input image for Starnet++. (Retrieved from the Starnet++ documentation)

The associated output image can be seen in Figure 2.5.

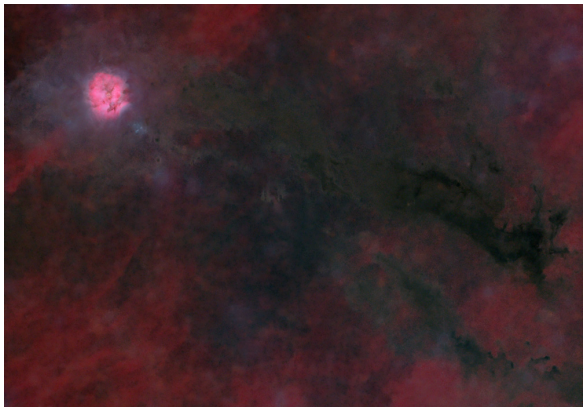

Figure 2.6: The output (starless) image for Starnet++. (Retrieved from the Starnet++ documentation)

It is clear that these results are quite impressive. However, there are a few downsides to a deep learning approach such as this.

First, we are restricted to 16 bits per channel TIF images. This means we must first stretch our 32 bits per channel images, which would result in a noticeable loss of detail.

Additionally, the execution time is quite high and will continue to grow quickly, as the size of the image increases. For example, on our local machine, Starnet++ ran for 103 seconds, for an 1048x712 image, which leads to a size of approximately 0.75 megapixels (MP). One can imagine how long the execution time would be for an astronomical image such as the one in Figure 3.1, which is approximately 28 times larger.

These execution times are based on a machine with the following processor:
Intel© Celeron© CPU B800 @ 1.50GHz × 2

We believe our implementation could improve on these two downsides. Let us now move into discussing our implementation. Afterwards, in Section 5, we will compare our application to Starnet++.

3 Implementation

The detection and subsequent removal of stars in an image can be reduced to the following five steps:

- Pre-processing
- Building a MaxTree
- Detecting objects
- Filtering stars
- Post-processing

3.1 Design choices

As was described in the introduction, one of the issues with the existing tools that are capable of detecting and removing stars, is that they take a long time to run. To combat this, I choose to write all of the code for this project in C++. C++ is known for being one of the fastest available languages (Tamimi, 2020). While C++ does not have as many built-in functionalities and easily accessible libraries as Python, I believe this is a sacrifice that would benefit the final product.

The final product can be found on Github [5]

[5] `https://github.com/Sander-Zeeman/StarRemoval`

3.2 Used dataset

A collection of images was made available by the first supervisor. This collection contains six FITS images, one JPG image and one TIFF image. Each of these images have a size of approximately 20 MP.

In figure 3.1, an example of one of these images can be seen. This images has the following dimensions: 5621x3718. This leads us to an image size of nearly 21 MP.

3.3 Reading and writing images

Currently, we only allow for greyscale FITS images to be processed. A simple future addition would be to allow for other file formats and/or multiple channels per image. Due to time constraints, this will not be considered in this thesis.

CFITSIO [6] is a library, developed by NASA, that allows the processing of FITS files. This library is responsible for reading the data from a FITS file and transforming it into an array of floats. In addition to this array, we will also store the width, height and total size of the image.

Additionally, this library will be used to write our processed image to a FITS file, as to remain consistent.

3.4 Pre-processing

Astronomical images tend to contain a lot of background noise. Because of this, it is important to perform pre-processing on the input images.

Closely following the algorithm described by (Teeninga et al., 2013), while referencing the MTObjects repository, I have implemented a function that will search the image for flat tiles.

Let the width and height of a flat tile be a power of 2, as this allows us to quickly find a suitable size for the given image. The starting size will be $2^6 = 64$. I check whether any flat tiles exist with a technique that will be discussed shortly. If any exist, we double the width and height of our tile and once again check whether any flat tiles exist. This process continues until no more flat tiles can be found, or the maximum size is reached, which is set to $2^7 = 128$. If no flat tiles were found in the first check, the operation is reversed. This means the size is then divided by 2 until either no more flat tiles are found, or the minimum size is reached, which is set to $2^4 = 16$. The last checked size for which at least one flat tile existed is then returned. This results in the largest size for which at least one flat tile exists, let us call this size n.

We can now divide our image into tiles of size $n \times n$ and determine whether they are flat tiles. We decide whether a tile is a flat tile based on the following criteria:

- The tile does not exceed the image boundaries
- The sum of all pixels in the tile are greater than 10^{-6}
- The tile is considered a normal distribution by the D'Agostino K-squared test
- The means of the top and bottom half and left and right half are considered equal by a T-test

Checking whether the tile exceeds image boundaries is trivial, we simply check whether the bottom right pixel exceeds either the height or the width of the image. There is no need to check the top left, as we start here and only move right and down. The boundaries are checked as to simplify the process of determining the flatness of tiles. Any tiles that exceed the boundaries are ignored. This may lead to slight inaccuracies, however, these are barely noticable.

The sum of pixels is equally as trivial. We work with greyscale images, so we simply sum the values of all pixels in the tile. If this sum does not exceed 10^{-6}, this tile is fully black and likely a result of faulty measurements, rather than actual data.

For the next two sections, we have to introduce the reject rates that were used. If the resulting p value of either test is lower than the corresponding reject rate, we have to reject our null hypothesis: "The values in this tile follow a normal distribution". This would imply that the current tile is not flat. Closely following MTObjects, we set the rejection rates as

[6] `https://heasarc.gsfc.nasa.gov/fitsio/`

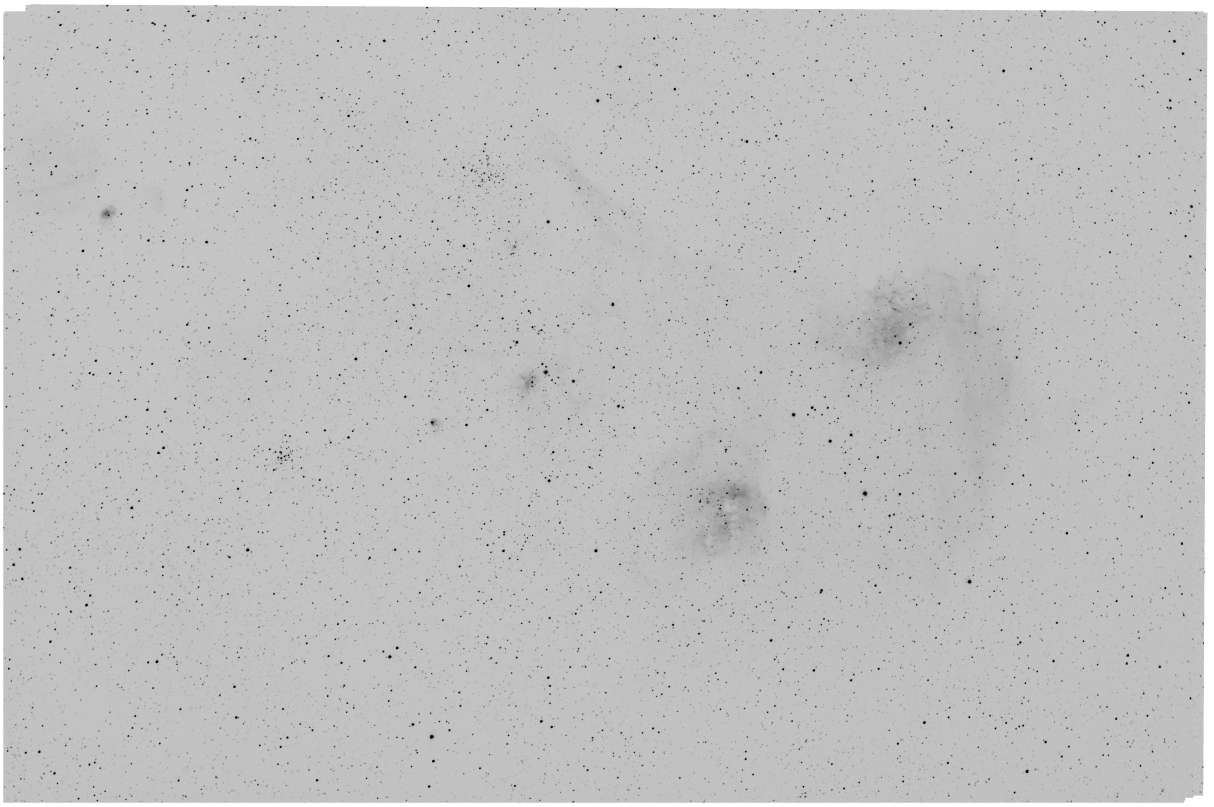

Figure 3.1: The image we will be testing on in our thesis

follows:

$$reject\_rate_1 = 1 - \sqrt{0.95}$$

$$reject\_rate_2 = 1 - \sqrt[4]{0.95}$$

Next, we perform a more interesting operation. We check whether the tile follows a normal distribution. The D'Agostino K-Squared test (D'Agostino et al., 1990) uses the skewness and the kurtosis of the distribution of a tile to determine whether it can be considered a normal distribution.

We extract the p-value of this operation by setting up a χ^2 distribution with 2 degrees of freedom. From here, the p-value is the probability that a score higher or equal to our score would occur in a normal distribution. We then check the resulting p-value against $reject\_rate_1$. If the p-value is greater, we can accept out null hypothesis, other wise, we will have to reject it.

If the data in this tile does not follow a normal distribution, it is very likely that this tile contains more than just background noise. Therefore, we can skip this tile if this is the case.

Finally, we wish to compare the mean of the left and right halves of the image, as well as the mean of the top and bottom halves of the image. If there is a statistically significant difference between the two means, this tile is likely not merely background noise and we can conclude this is not a flat tile. We use a T-test to decide whether the two means have a statistically significant difference. We extract the p-value of this operation by setting up a students_t distribution with $n * n - 2$ degrees of freedom. From here, the p-value is the probability that a score would end up as far or further into either tail side of the distribution. Once again, we check the resulting p-value against $reject\_rate_2$ and accept or reject accordingly.

After finding these areas, the mean, variance and gain will be calculated. The mean will be subtracted from the entire image, as to remove a large amount of background noise. The variance and gain are stored for later use in our significance test. Because the amount that is subtracted from the image is based on the average background value, this method will work on all images where we are able to find

flat tiles.

3.5 Building a MaxTree

After pre-processing the data, we are now ready to start building a MaxTree. Following (Wilkinson, 2011). We will start by finding the least bright pixel in our image, which is a trivial exercise.

We will insert this pixel into a heap. From here, we will enter a loop until the heap is empty. We first add the pixels above, below and to the sides of the pixel in the top of the heap, to the heap, given that these pixels have not entered the heap before. If any of these pixels have a higher value than the one at the top of the heap, we move up as many layers as needed, as to always remain on the highest known level. If any adjacent pixels have the same value, we merge them into a single node.

By the end of this loop, it is possible that the largest pixel in the heap is now on a lower layer than where we started. In this case, we descend to the lower layer, so we can continue our loop.

By the end of our loop, we link any loose nodes to the node below it, this will lead to a fully connected array of nodes.

3.6 Star Detection

We have now built a MaxTree structure, however, we can not infer a lot from this structure alone. We need to filter these nodes, such that we are left with only the nodes that represent stars. First, we will implement faint object detection, as described in (Teeninga et al., 2015). Afterwards, we will have a list of objects, which we will filter on attributes that are specific to star-like structures.

3.6.1 Determining the relevant indices

Currently, we have as many nodes as there are pixels in our image. It is easy to see that this is far more than we will actually need. As was stated in section 3.5, nodes are often merged into a single node.

However, these merged nodes are still present in the structure. Each node points to its parent, which is either the node it was merged with, or the main node of the layer below. Therefore, we can get rid of a lot of nodes by only keeping level roots and the nodes whose parent has a different pixel value from themselves.

Additionally, common sense tells us the least bright node in the image will never be an object, therefore the root of the structure is also ignored.

All remaining nodes are sorted from least bright to brightest, using a heapsort, and reversing it afterwards. We sort the data this way, as we already had access to a heap that can handle data of the Pixel class.

3.6.2 Significance testing

After reducing the size of our list of nodes by a large amount, we want to filter the nodes that are not significant. In order to do this, we run the significance test as described by (Teeninga et al., 2013). Our implementation is directly based on the implementation included in MTObjects.

This significance test is based on the "power" of a peak components. The power of a component P is described by the following equation: (Teeninga et al. (2013))

$$power(P) = \sum_{x \in P} (f(x) - f(parent(P)))^2$$

where function f returns the grey level of a pixel x.

Whether a peak component is caused by noise or by a significant object, is decided by the number of degrees of freedom of the χ^2 distribution of $\frac{f^2(x)}{\sigma_B^2}$ for x in P. If the distribution has 1 degree of freedom, P is likely due to noise. Otherwise, the number of degrees of freedom describes the significance level of this node. This significance level is later used in the object detection. Please see the original paper by (Teeninga et al., 2013) for further detail.

After each node that we classify as significant, we update its main branches for later use in the object detection. What this means is that, when the closest ancestor of a node already has a significant descendent, the area of this descendent is compared to the area of the current node. The node with the larger

area will then be the the node in the main branch of the ancestor. If the ancestor does not yet have a significant descendent, we insert the current node into its main branch.

3.6.3 General object detection

Now that we have decided which nodes are significant, we must find whether any nodes form a single object together. This is easily done by checking the parent of each node.

If the parent of the node is also a significant node, we know the current node is a part of the larger parent structure. Therefore, we can mark any significant node whose parent is not significant, as an object.

Additionally, we must account for nested objects, as one can imagine the scenario in which a star covers another celestial object that would be deemed significant. These object are caught by the main branches we introduced in the previous section, where we discussed significance testing (section 3.6.2). If the main branch of the parent of this node does not point to this node, we know this object is nested on the object below. Therefore, we know this is a separate object and set it as such.

3.6.4 Star filtering

Now, we have identified all objects within our image. Let us move into differentiating stars from other objects. First, we will remember the attributes of a star, that made us want to remove them in the first place. Stars are bright point sources, which means the highest pixel value in a star is likely to be high. Additionally, stars are quite small, and there are many of them. Given these attributes, brightness and area, we can tell we need a measure of how dense an object is. A natural solution is the following:

Let us define the highest pixel value in an object as p, the pixel value of the lowest layer as b and the area of the object as a. A test which determines whether an object is a star could have the format described in Formula 3.1

$$\frac{p-b}{a} > C \tag{3.1}$$

Where C is some constant that would differ from image to image.

It is possible for this test to miss some larger stars, so let us also threshold the objects, such that any object with a highest pixel value greater than M would also be classified as a star, as described by Formula 3.2.

$$p > M \tag{3.2}$$

Finally, this addition is likely to include some larger objects with high peak values as well, so we will conclude by setting a maximum area A, which will have to be adjusted for a new image, as in Formula 3.3

$$a < A \tag{3.3}$$

This leaves us with the system for star detection that is described in Equation 3.4.

$$\left(\frac{p-b}{a} > C \vee p > M\right) \wedge a < A \tag{3.4}$$

The A, C and M parameters will have to be tuned for a specific image in the current state of the application. Potential improvements to this method will be discussed in Section 7.

At this time, for the example image, given in Figure 3.1, the chosen parameters are:

$$\begin{aligned} A &= 2000 \\ C &= 0.3 * 10^{-4} \\ M &= 0.8 \end{aligned}$$

Now, we can mark all objects according to the result of this test. If the object is a star, all pixels that are a part of the object will be marked with the index of the root node of the star, if the object is not a star, all related pixels are marked with a negative value, such that we know to ignore it. This marking process will happen in a separate array, the 'mask', which we will use during Section 3.8.

3.7 Removing stars

After marking all nodes that specify stars, we want to return from our MaxTree interpretation of the data, to the image interpretation. Therefore, we create a mask of our original image. Every pixel that is not in any way a part of a star, is given a negative value, which tells us we can safely ignore this pixel when we get to removing the stars. Every pixel that is a part of a star, will be given the index

of the main node of this star. It would have been sufficient to use a boolean mask, where we only set the pixels that are part of a star to true. However, the decision to include this additional information was made to simplify future additions to this program, which might need to tell which pixels belong to separate stars.

From here, it would be easy to simply set all pixels that are part of a star to 0. However, while this would achieve our goal of removing stars from an image, it is not very pleasant to look at, as well as removing a large amount of context from the image. Because we know exactly where stars were located, we are able to resolve this issue through post-processing.

3.8 Post-processing

In the Related Work section (section 2.4), we introduced multiple image inpainting techniques. Next, we will discuss the techniques and how they integrate in our project.

3.8.1 Hierarchical inpainting

As mentioned before, the MaxTree data structure allows us to know the pixel value of the layers below an object. So a quick improvement to setting all star pixels to 0, would be to set them to the value of the lowest layer of the star instead. Because every pixel that should be inpainted is marked with the index of the significant node of this object, we can access the main node of the object from any pixel. Therefore, we can easily access the value of the lowest layer of this object from any pixel. Finally, we must remember that it is possible for stars to be nested in larger stars, in this case, we must take the lowest layer of the lowest star instead.

3.8.2 Linear interpolation

In order to perform linear interpolation, we require to know a pixel value, which has not been removed, on all sides of each removed pixels, as well as the distance to it. Through our mask, we can find the pixels that have not been removed quite easily. If we were to hit the boundaries of the image, we simply take the distance as the distance to the edge, while taking 0 as our value, otherwise, we take the value that this pixel had in our original image. We interpolate a value for both the row and column, as to make the result less dependant on which axis contains more information, then take the average of these as our new value.

3.8.3 Navier-Stokes based

As for the Navier-Stokes based inpainting method, we noticed the OpenCV[7] library, contained an implementation[8] of this algorithm.

After supplying this algorithm with the image, the mask and a radius, it returned an inpainted image. We selected a radius of 5, as this should give the algorithm a clear idea of the area surrounding a star.

Let us now move on to highlighting the results of our research.

4 Results

We have to note a few things before getting into our results.

First, while the images were originally, as one would expect, mainly black, with objects being white. We chose to invert the colours of the images, as black objects on a white background are easier to notice that white objects on a black background.

As can be seen in Figure 4.1, the full-sized images can get quite small in a two-column format, therefore, full sized copies will be included in Appendix A.

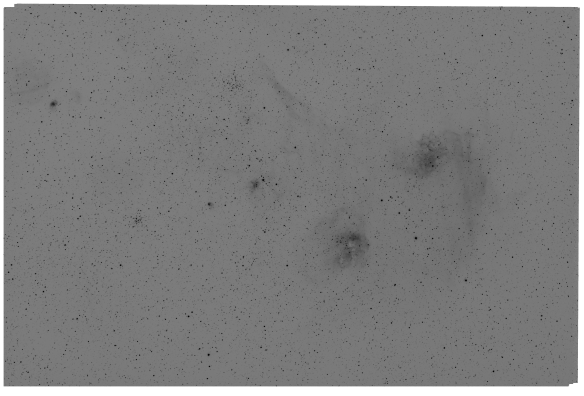

Figure 4.1: The original image. (See also the uninverted, unstretched image in Figure 3.1)

[7] https://opencv.org/

[8] https://docs.opencv.org/4.5.2/d7/d8b/group__photo__inpaint.html

Additionally, showing the results by only removing the stars from our original image does not do the results justice. In Section 3.4, we described how the background noise was removed from our images. Instead of showing the original image, we will use the image with its background noise removed, as this makes the results easier to interpret. Additionally, the inpainting can have some strange effects on the noisy images, as the pieces of background that remain after background removal are often considered stars. This has interesting effects when trying to rebuild a noisy image from the MaxTree of a noiseless image. These results are not representative of the effectiveness of our application, yet the results will be added to give an idea of the described issue. These full noisy images will be included in Appendix B.

In Figure 4.2, the full image, where the background has been removed, can be seen.

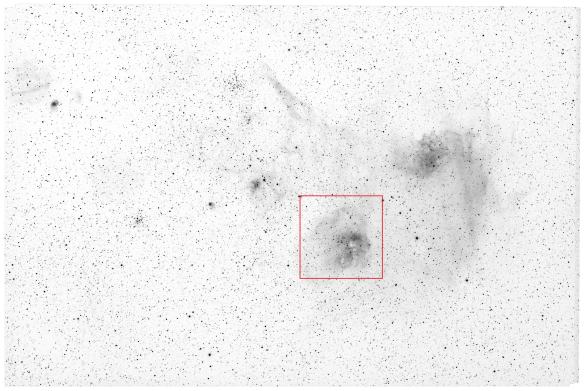

Figure 4.2: The example image with its background noise removed and the zoomed area marked.

As one can imagine, the full effects of inpainting can hardly be seen in an image of this size. Therefore, we will also be including an 800x800 cutout of a specific section of each image. The zoomed area has been marked in Figure 4.2 with a red box. The zoomed version of Figure 4.2 can be seen in Figure 4.3

You will find that, while the results are all given in this section, they will not yet be analyzed. We will save this process for the discussion in Section 5.

Note that contrast stretching was manually performed on these images where possible, for the sake of the reader. After all, images with all pixel values being in the range $[0, 0.1]$ can be quite difficult to analyze.

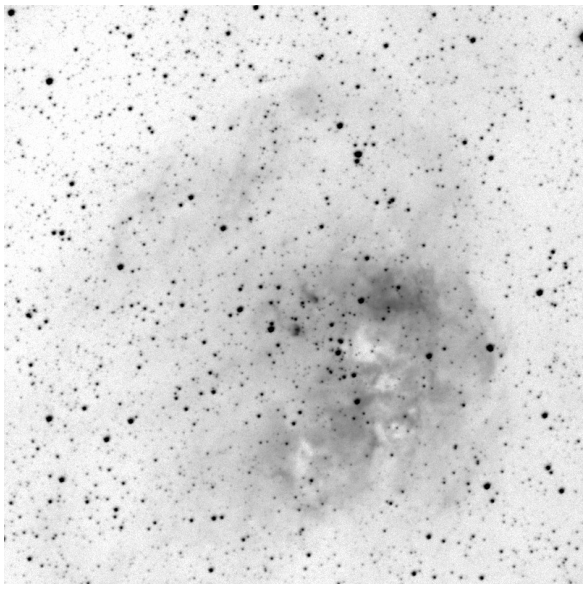

Figure 4.3: The zoomed section of Figure 4.2.

Finally, let us mention the average execution times of each section, besides the post-processing methods. (These will be discussed n the respective sections) Note the star detection is split into multiple subsections. Additionally, some input preparation happens in between sections, hence the higher total time than the sum of all sections.

As a final remark, these execution times are based on a machine with the following processor:
Intel© Celeron© CPU B800 @ 1.50GHz × 2

- Reading an image: $0.22s$
- Pre-processing: $1.58s$
- Building the MaxTree: $11.98s$
- Star detection: $19.67s$
  - Finding relevant indices: $10.36s$
  - Finding significant nodes: $4.30s$
  - Finding objects: $1.70s$
  - Finding stars: $1.93s$
  - Marking IDs: $1.38s$
- Writing an image: $0.19s$
- Writing a zoomed image: $0.01s$
- Total: $35.21s$

4.1 No inpainting

Let us start with the easiest post-processing method: No post-processing at all. The pixels that represent part of a star will have their pixel value set to 0. The resulting image can be seen in Figure 4.4.

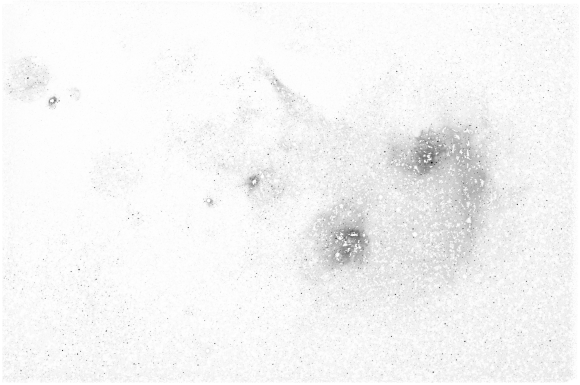

Figure 4.4: The image with its stars removed.

As before, the zoomed section can be seen in Figure 4.5

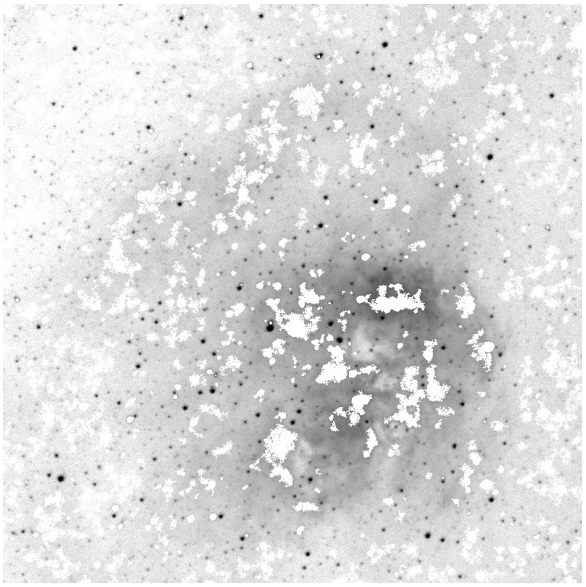

Figure 4.5: The zoomed section of Figure 4.4.

This process runs in linear time, as we access every pixel once. This lead to an execution time of $0.08s$.

4.2 Hierarchy-based inpainting

Next, let us discuss the results with the first post-processing method that we discussed. (see Section 3.8.1) The result from setting all pixels that represent a star, to the value of the lowest layer of the star, leads us the figure we display in Figure 4.6.

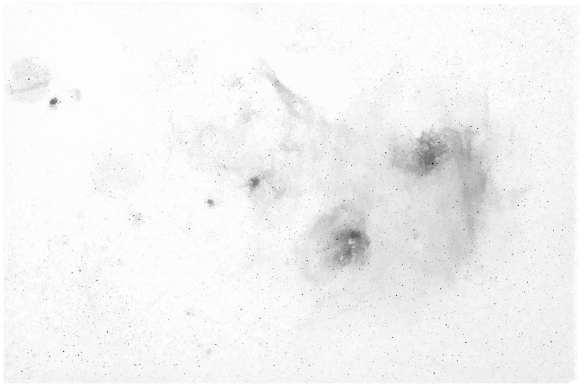

Figure 4.6: The image, inpainted through its hierarchical structure.

Once again, the zoomed section of this image can be seen in Figure 4.7.

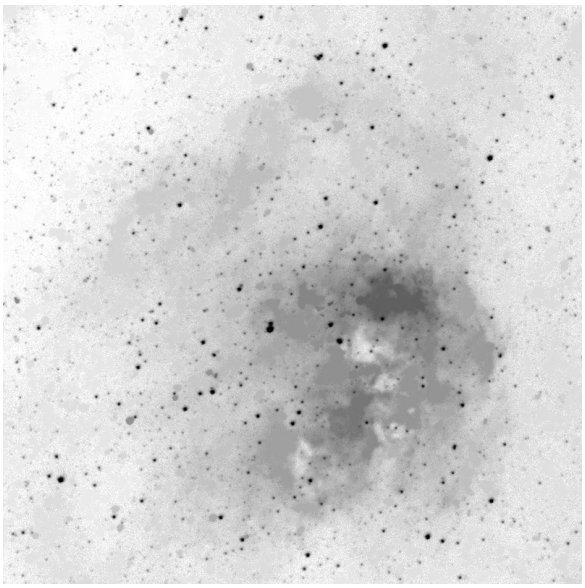

Figure 4.7: The zoomed section of Figure 4.6.

This process also runs in linear time, as we access most pixels once, only accessing pixels again when a star is nested in another star. Additionally, we must read an extra value from memory for each pixel, instead of setting each pixel to a constant. This leads to an execution time of $0.19s$.

4.3 Linear interpolation

Let us move on to the first post-processing techniques which uses surrounding pixel values to fill in the removed sections. (see Section 3.8.2) Applying linear interpolation to estimate the missing pixel values produces the image shown in Figure 4.8.

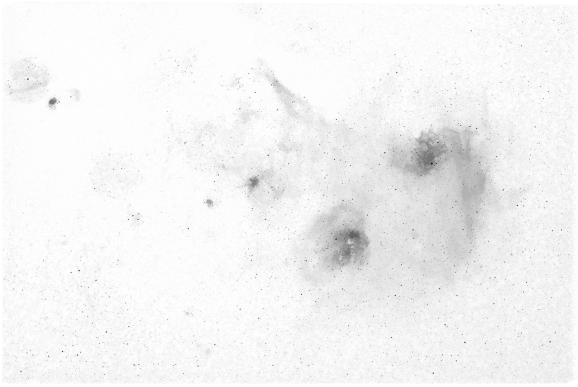

Figure 4.8: The image, inpainted through linear interpolation.

Once more, the zoomed section of this image can be seen in Figure 4.9.

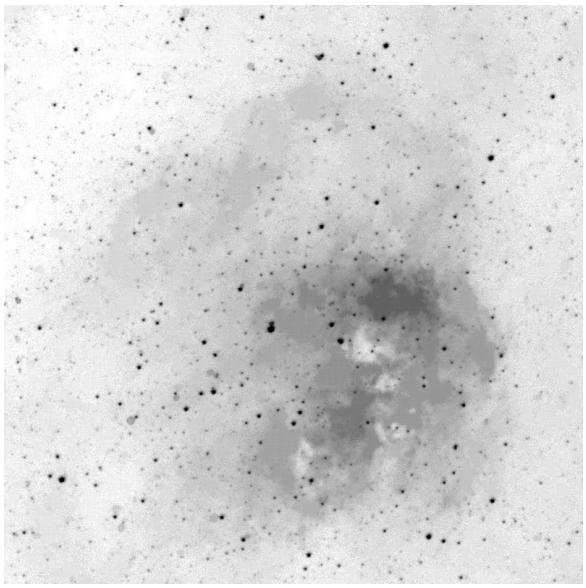

Figure 4.9: The zoomed section of Figure 4.8.

In a worst-case scenario, interpolation takes

$$\mathcal{O}((w+h)*n) \tag{4.1}$$

where w is the width, h is the height and n is the size of the image. This is still considered linear time. This lead to an execution time of $0.79s$.

4.4 Navier-Stokes based inpainting

Next, we will showcase the results of our final inpainting method. The Navier-Stokes based inpainting method was described in Section 3.8.3.

Unfortunately, we found that the Navier-Stokes inpainting method could not be completed in a reasonable amount of time. Details of this will be discussed in the Section 5.

We found this method could run in reasonable time on the smaller subsection of the image that we have been displaying in former sections. These results can be seen in Figure 4.10.

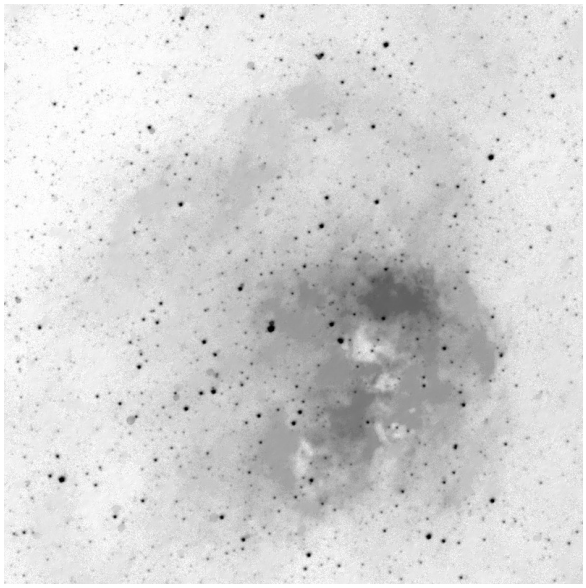

Figure 4.10: Navier-Stokes inpainting applied on a subsection of our image.

This subsection of the image is 800x800 in size. The execution time of this inpainting method was $35.16s$. From this, we can make an estimate of the execution time of the full 5621x3718 image. We make an estimate, as the program was unable to finish execution in a reasonable amount of time due to the large image size. This will become clear from the estimated execution time.

We know a 800x800 section led to an execution time of approximately 35 seconds. As this algorithm aims to fill in the stars, using only the closely surrounding pixels, we can estimate a minimum execution time by assuming the execution time will increase linearly, as filled

in stars will generally not be revisited.

$$\frac{5621 * 3718}{800 * 800} \approx 33$$

Therefore, the minimum execution time on the full image would be atleast 33 times larger than the execution time we noted from the 800x800 subsection. This leads us to a minimum execution time of $35 * 33 = 1155$ seconds, or 19 minutes and 15 seconds. Note that this estimate is based on the assumption that the execution time grows linearly based on image size alone, this means the actual execution time may be even larger.

Now, we have shown all the relevant results that we obtained. In the upcoming section, these results will be discussed and a comparison with Starnet++ (as described in Section 2.5) will be made.

5 Discussion

Before discussing the inpainting, let us discuss the effectiveness of our star removal.

5.1 Star Removal

In Figures 4.4 and 4.5, we see the images that results from removing all detected stars. By comparing these figures to Figures 4.2 and 4.3, one can clearly tell that a portion of stars have been detected and subsequently removed. However, it becomes very clear that not all stars are being detected. This is especially clear in the zoomed in figure.

A large number of stars remained because the parameters were not properly tuned to this image. Similarly, one could argue our choice of compactness measure (Formula 3.4) is suboptimal.

From this, we must conclude that our star removal algorithm does not yet do what we intended. The algorithm can be improved upon by means of different attribute filters, such that few to no stars remain after filtering.

Next, let us discuss the results of the inpainting methods we explored.

5.2 Inpainting methods

In general, the inpainting section of this project can be considered a success. As will be described in detail in the upcoming subsections, we have shown that image inpainting is an effective way of removing any marks left behind by star removal. However, for this to become as effective as possible, the star removal itself must be improved upon.

For every method, we will discuss multiple factors. Namely, the effectiveness of the inpainting algorithm in covering up the removal, the scalability to larger images and any remarkable downsides.

5.2.1 No inpainting

First, we explored what would happen if we were to not perform any post-processing whatsoever. The results of this experiment are visible in Figures 4.4 and 4.5.

One can imagine that these results would not be a good fit, when our goal was to hide the holes created by the removal.

While these results will naturally not be a contender for the optimal inpainting method, they were very useful in estimating the quality of our star removal, as we did in Section 5.1.

As this will not be a contender for optimal inpainting method, there is no need to discuss its scalability or quality. Both should become clear from realising the exact operation that was performed.

5.2.2 Hierarchy-based inpainting

In this inpainting method, we attempted to use the MaxTree structure to discover the lowest pixel value that is still part of the star. We hypothesised that this method would result in lighter grey areas appearing where stars used to be. These results can be viewed in Figures 4.6 and 4.7.

We can tell that the obtained results match our expectations. This method lead to images that seem more natural than the "holes" that appear when merely removing stars. However, the flat areas are noticeable, especially when considering the zoomed image in Figure 4.7.

This process was very efficient, running in linear time, though it has not hidden the inpainting as well as we would like.

We explored this method as a baseline for our inpainting algorithms. Our inpainting algorithms should improve on the results obtained through this hierarchy-based approach. Let

us now discuss the results of our first actual inpainting algorithm.

5.2.3 Linear interpolation

The first inpainting algorithm, from which we expected good results, applies linear interpolation to estimate the pixel values of all pixels that were removed by the star removal. Linear interpolation is often used to estimate a smooth transition between two know data points. There exists no way of knowing what was actually behind the stars, given only the image data. Because of this, the best we can do is estimate how the pixels should be filled in, according to the surrounding known data points. The results of applying linear interpolation to our image can be viewed in Figures 4.8 and 4.9.

This algorithm behaves as we would expect. Removed areas are filled in with with values similar to the surrounding pixels. While objective bad inpainting results do exist, good inpainting can be very subjective when a ground truth does not exist. We believe the linear interpolation is a good middle ground between inpainting quality and execution speed. The results look "whole" and the removed sections slightly blend into the surrounding area, unlike the previously discussed inpainting methods. Additionally, in the smaller removed sections, the fact that inpainting occurred is barely noticeable. While this fact is noticeable in the larger removed sections, we believe these results are acceptable.

Finally, the execution time is not an issue for this method, as linear interpolation runs in linear time.

5.2.4 Navier-Stokes based inpainting

As for our final experiment, we explored the possibility of applying a Navier-Stokes based inpainting method to fill up the artifacts that were created by the star removal process. As this method is based on fluid dynamics, we hypothesised that this method would perform especially well in a noisy environment, as we should notice quite a fluid transition from low to high pixel values. The result can be viewed in Figure 4.10.

We notice that the result looks quite nice, as we predicted. However, the difference between linear interpolation and Navier-Stokes based inpainting is not immediately clear. The results of these two methods look quite similar, which goes to show that linear interpolation would be an acceptable choice for inpainting algorithm.

However, as was mentioned in Section 4.4, the execution time of this method is its downfall. While this method is very effective at filling in any artifacts in an image, it is quite a computationally expensive algorithm. The algorithm was applied on a 0.64MP subsection of our image. This lead to an execution time of approximately 35 seconds. Naturally, the algorithm never finished on the full 21MP image. Because of this, the Navier-Stokes inpainting method cannot be considered for a final product.

5.3 The effect of contrast stretching

After discussing the effects of the various inpainting techniques that were applied in this thesis, we should discuss the how the star removal and inpainting affected the effectiveness of contrast stretching.

We decided to compare the original image to the image that was inpainted through linear interpolation, as this method was deemed the optimal method from the few methods we tested. These images can be found in Figures 5.1 and 5.2 respectively.

On both of these images, logarithmic contrast stretching, as described in Section 2.1, was applied.

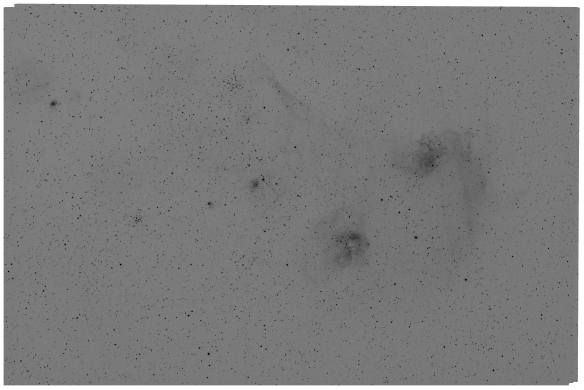

Figure 5.1: The "pre-removal" image after applying contrast stretching.

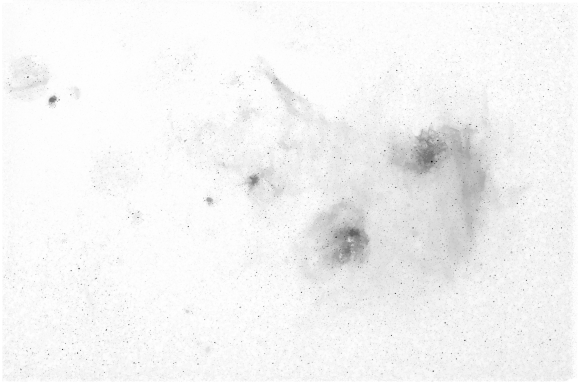

Figure 5.2: The "post-removal" image after applying contrast stretching.

As described in Section 2.1, the goal of logarithmic contrast stretching is to enhance the contrast between the bright and dark sections of the images. We can tell that the pixel values in Figure 5.1 tend to blend together; the distinction between nebulae, stars and background is not very clear. On the other hand, in Figure 5.2, even the faint sections of the nebulae are clearly visible, while the background is nearly fully white. Our original intent with this research was to decrease the brightness of the brightest parts of the image, as to allow for more effective contrast stretching. From these results, one can tell that the effectiveness of contrast stretching has indeed improved after applying our star removal process.

Next, let us move into our final point of discussion. How did we do in comparison to the state of the art star removal tool?

5.4 Comparing to Starnet++

Let us remind ourselves of the input and the result that were discussed in this thesis of both our application and starnet++. The input image and the resulting image from starnet++ can be found in Figures 2.5 and 2.6 respectively. Similarly, our input and resulting image can be found in Figures 4.1 and 4.8 respectively.

It becomes immediately clear that Starnet++ does not suffer the quality issues that our application raised, as discussed in Section 5.1. All stars were detected and removed accurately by Starnet++, while our application left a large number of stars untouched, as well as falsely identifying multiple background fragments as stars.

On the other hand, our application does not suffer from the disadvantages of Starnet++ either, as listed in Section 2.5. We are not limited to 16 bits per channel images, instead, we allow for full 32 bits per channel to be processed. Furthermore, our code requires minimal changes in order to support 64 bits per channel images. Additionally, as mentioned in Section 2.5, Starnet++ runs for 103 seconds to process an image of size 1048x712. In contrast, our application processes an image that is 28 times larger, in approximately a third of the time. From this, we conclude that our application is far more scalable to larger image sizes than existing deep learning models.

Therefore, it should be possible to create a tool that is able to compete with existing star removal software, if the star detection is improved by using other attribute filters.

6 Conclusion

The goal of this thesis was to explore the effectiveness of applying MaxTree-based object detectors to identify and remove stars from astronomical images. Additionally, we explored multiple types of image inpainting to fill in the "holes" that appear as a result of the star removal.

From our results, we can tell that star detection through MaxTree-based object detection has great potential. In its current state, larger stars and dim stars may be missed. However, even though our implementation is not ready to be applied in image processing pipelines, the effect of contest stretching has greatly improved as a result of our application.

We gained a lot of insight into the effectiveness of inpainting algorithms from our experiments. As one could expect, inpainting based on the MaxTree structure leaves some noticeable flat areas in the image. Furthermore, advanced methods such as the Navier-Stokes algorithm are not feasible for astronomical images, due to their astronomically large dimensions. These dimensions would require any algorithms to execute in (close to) linear time. Unfortunately, most advanced inpainting algorithms are not capable of adhering to this restriction. However, the image that resulted from applying interpolation had surprisingly similar re-

sults to the resulting image after applying the Navier-Stokes based method. Because of this, we believe advanced methods may not be necessary to produce acceptable, or even good results. Additionally, this method leaves areas that are slightly less flat compared to the results of the hierarchical inpainting. The flat areas are still noticeable, however, the linear interpolation is an improvement over hierarchical inpainting, as it considers all surrounding pixel values, rather than only the lowest value. Therefore, from the four explored methods, we would recommend linear interpolation.

In Figures 5.1 and 5.2, we can see the massive difference in how contrast stretching affects our image. It is clear that contrast stretching has indeed become more effective after star removal, which tells us this hypothesis was correct.

Finally, an important goal of this thesis was to compare our application to Starnet++, a deep learning based star removal tool.

Starnet++ is more thorough in the detection of stars than our application. However, our application has avoided the disadvantages that Starnet++ brought. Namely, our application allows for 32 bits per channel images (and is easily extendable to 64 bits per channel images), whereas Starnet++ requires images to be stretched to 16 bits per channel. Additionally, our application executes much faster than Starnet++. While Starnet++ runs for 103 seconds to process an image of size 1048x712, our application is capable of processing an image of size 5621x3718 in just 35 seconds.

We believe that star removal through connected morphological image processing has great promise to surpass deep learning methods. Unlike deep learning methods, connected morphological image processing easily scales to large images. Considering astronomical images can become larger than a terapixel, deep learning does not seem suitable for use in a professional image processing pipeline, however, the methods described in this thesis do seem suitable.

Finally, we will outline the ways in which the current state of the software could be improved.

7 Future Work

The future work on this project can be divided into two categories: "Improving the existing methods" and "New additions that would benefit this project".

First, we will list the points of improvement in the current state of our research.

- Our implementation of the MaxTree is not yet optimal. Parallel algorithms exist, which could significantly speed up the entire program, as flooding the MaxTree is currently is nearly a third of the total execution time.
- The formula in Equation 3.4 is likely suboptimal and requires the tuning of three separate parameters. State of the art compactness measures could be explored in future research.
- Ground truth images may be artificially created, this would allow us to quantify the results of our inpainting algorithms, which would allow us to make a definite recommendation.
- Finally, other inpainting algorithms may be explored. Consider, for example, quadratic or cubic interpolation, or even a machine learning method.

Next, we will list any additions that would benefit the project, that we can think of.

- Implementing FreeImage[9] support for the reading of images from multiple filetypes, such as PNG, TIFF or JPG, into an array of floats. The rest of the application does not depend on the filetype, so the star removal would work as it should.
- The current equations used to determine whether an object is a star (Equation 3.4), use constants that can differ greatly from image to image. An interesting technique that could be explored, is applying a clustering algorithm to certain attributes of an object (area, volume, power, brightest pixel). A centroid-based clustering algorithm seems optimal for this cause, as it will often classify close

[9] https://freeimage.sourceforge.io/

data points in the same way. Specifically KMeans is promising. This would automate the tuning process mostly, which would greatly increase user experience.

- We could further automate the star removal by automatically stretching the contrast right before saving our output image.
- Finally, as was mentioned in the discussion and conclusion, our application can easily be extended to support 64 bits per channel images. Additionally, as we currently only support greyscale images, introducing star removing on multiple channels would be an interesting addition. I hypothesise that simply applying our application on all channels separately should produce some interesting results, however, there is no evidence to support this claim.

It is clear that there are many ways in which the framework we built can be improved upon. Considering that the current results seem promising, yet not ready to be deployed as a stand-alone application, we believe that the use of a MaxTree, combined with statistical attribute filtering, could lead to huge improvements in the current state of star removal software.

References

Anand, A. (2017). Unit 12: Image enhancement and transformation. Retrieved from: `https://www.egyankosh.ac.in/bitstream/123456789/39542/1/Unit-12.pdf`. Accessed: 11.08.2021.

Bertalmio, M., Bertozzi, A., and Sapiro, G. (2001). Navier-stokes, fluid dynamics, and image and video inpainting. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, pages 355–362.

D'Agostino, R. B., Belanger, A., and D'Agostino Jr., R. B. (1990). A suggestion for using powerful and informative tests of normality. *The American Statistician*, 44(4):316–321.

Haigh, C., Chamba, N., Venhola, A., Peletier, R., Doorenbos, L., Watkins, M., and Wilkinson, M. H. F. (2021). Optimising and comparing source-extraction tools using objective segmentation quality criteria. *Astronomy & astrophysics*, 645:A107.

Jones, R. (1999). Connected filtering and segmentation using component trees. *Computer Vision and Image Understanding*, 75(3):215–228.

Library of Congress (2014). Galileo and the telescope. Retrieved from: `https://www.loc.gov/collections/finding-our-place-in-the-cosmos-with-carl-sagan/articles-and-essays/modeling-the-cosmos/galileo-and-the-telescope`. Accessed: 15.06.2021.

Patrikalakis, N. M., Maekawa, T., and Cho, W. (2009). 8.1.2.2 isophotes. Retrieved from: `https://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node148.html`. Accessed: 15.06.2021.

Sahidan, S., Mashor, M., Wahab, A., Salleh, Z., and Ja'afar, H. (2008). Local and global contrast stretching for color contrast enhancement on ziehl-neelsen tissue section slide images. In *4th Kuala Lumpur International Conference on Biomedical Engineering (BIOMED 2008)*, pages 583–586.

Salembier, P., Oliveras, A., and Garrido, L. (1998). Antiextensive connected operators for image and sequence processing. *IEEE transactions on image processing*, 7(4):555–570.

Tamimi, N. (2020). How fast is c++ compared to python? Retrieved from: `https://towardsdatascience.com/how-fast-is-c-compared-to-python-978f18f474c7`. Accessed: 15.06.2021.

Teeninga, P., Moschini, U., Trager, S. C., and Wilkinson, M. H. F. (2013). Bi-variate statistical attribute filtering: A tool for robust detection of faint objects. In *11th International Conference on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-11-2013)*, pages 746–749.

Teeninga, P., Moschini, U., Trager, S. C., and Wilkinson, M. H. F. (2015). Improved detection of feint extended astronomical objects through statistical attribute filtering. In *12th International Symposium on Mathematical Morphology (ISMM 2015)*, pages 157–168.

Wilkinson, M. H. F. (2011). A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP 2011)*, pages 1021–1024.

A

Full sized images

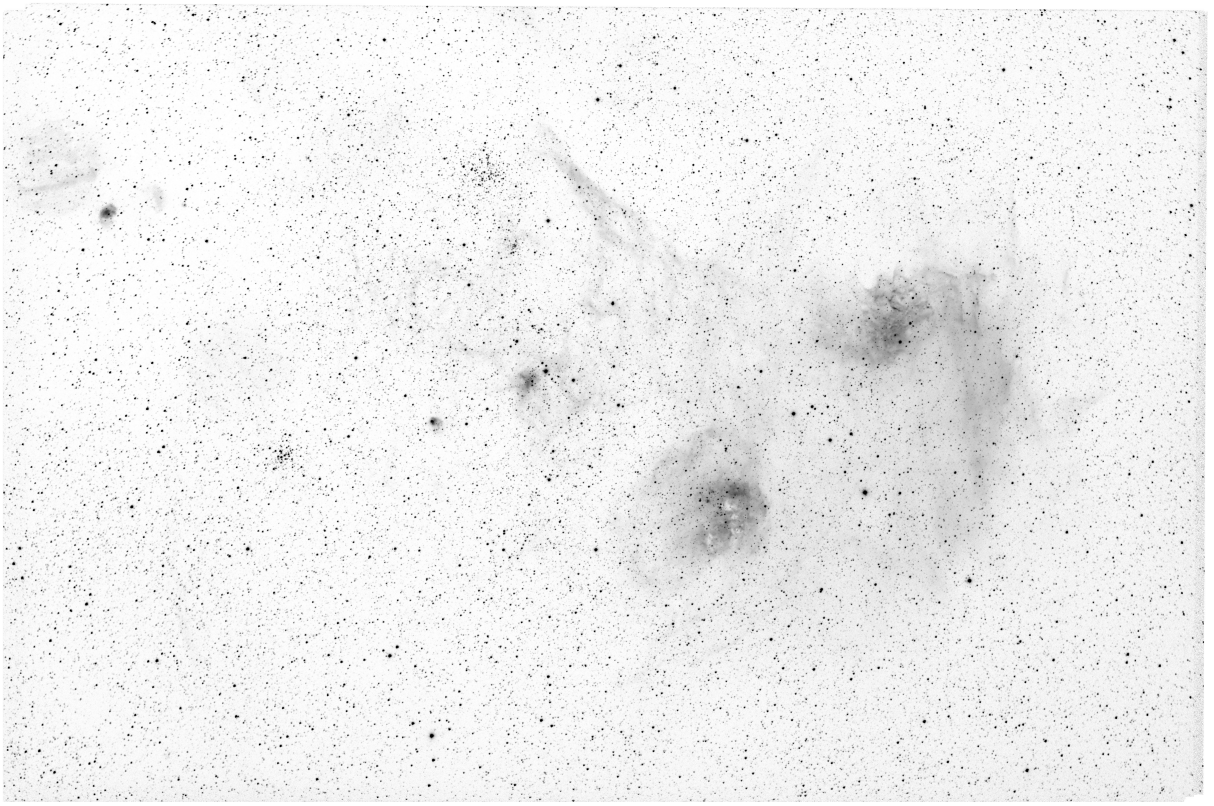

Figure A.1: The inverted image with background noise removed.

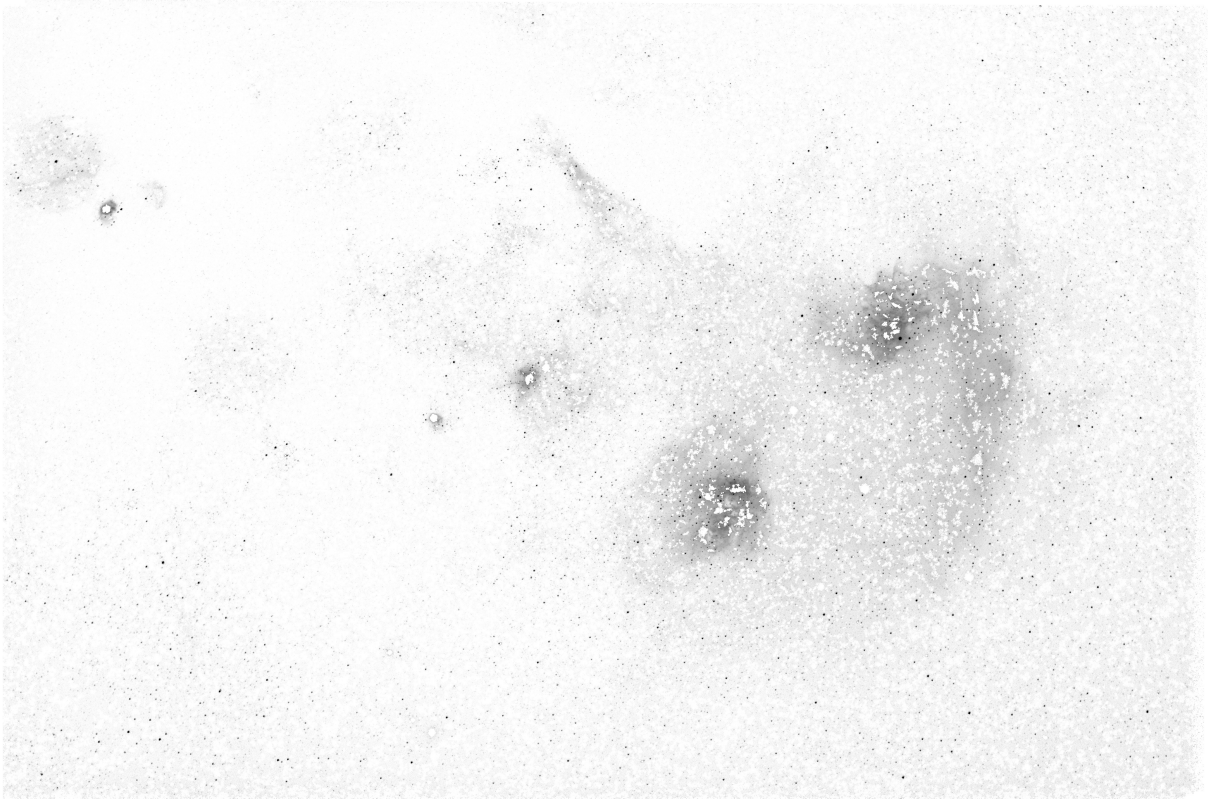

Figure A.2: The image with background noise and stars removed.

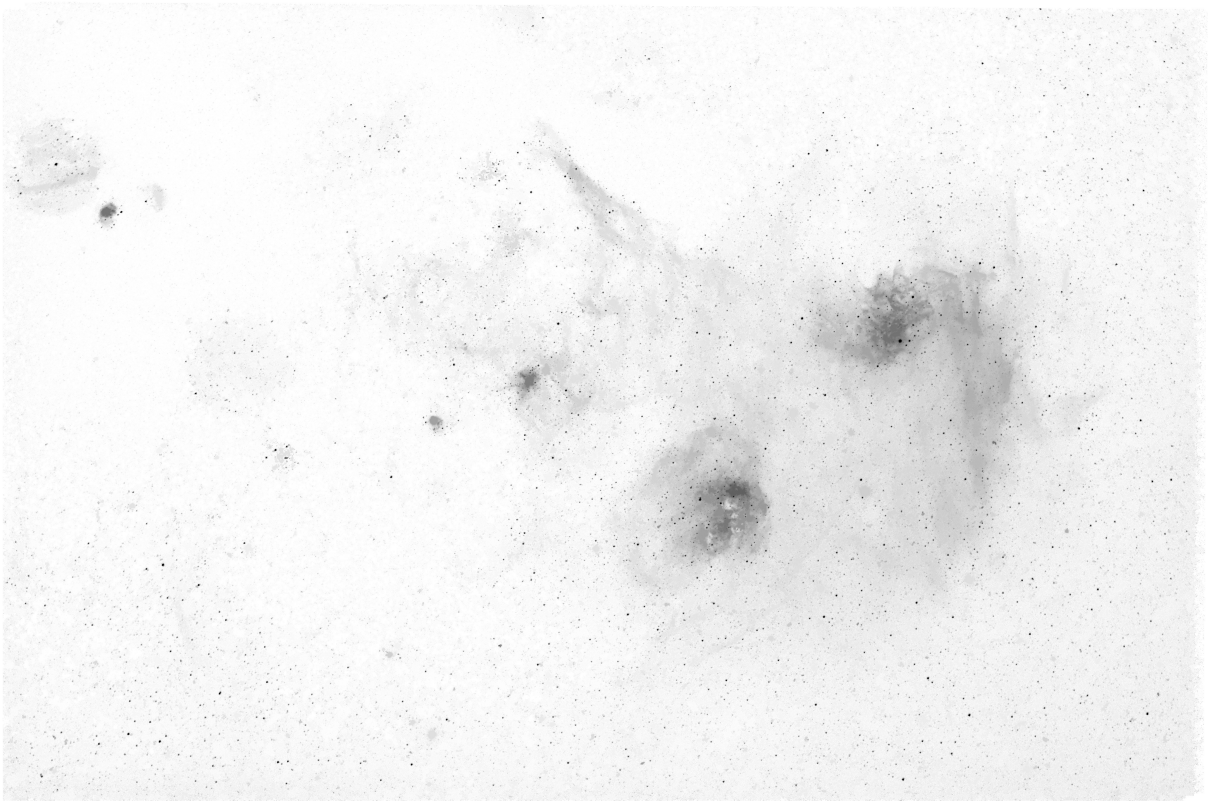

Figure A.3: The image with background noise removed and stars filled in through the MaxTree structure.

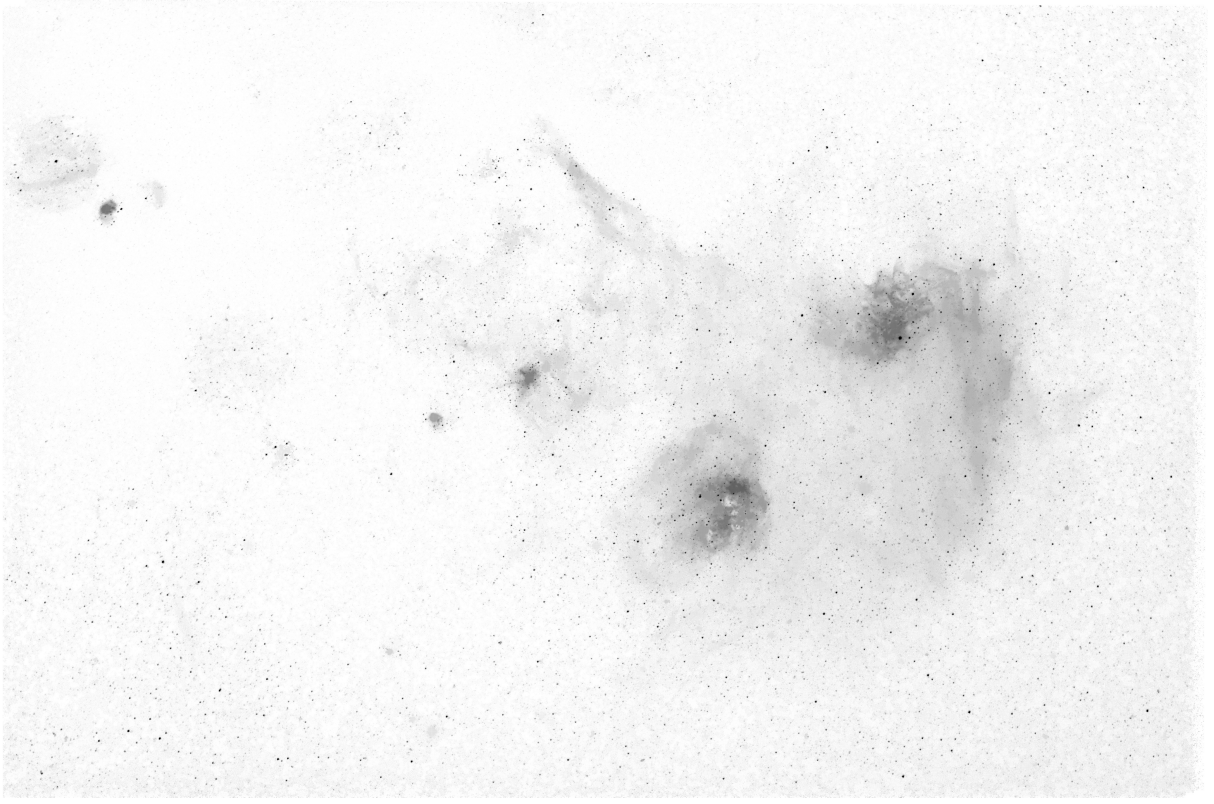

Figure A.4: The image with background noise removed and stars filled in through linear interpolation.

B

Full sized noisy images

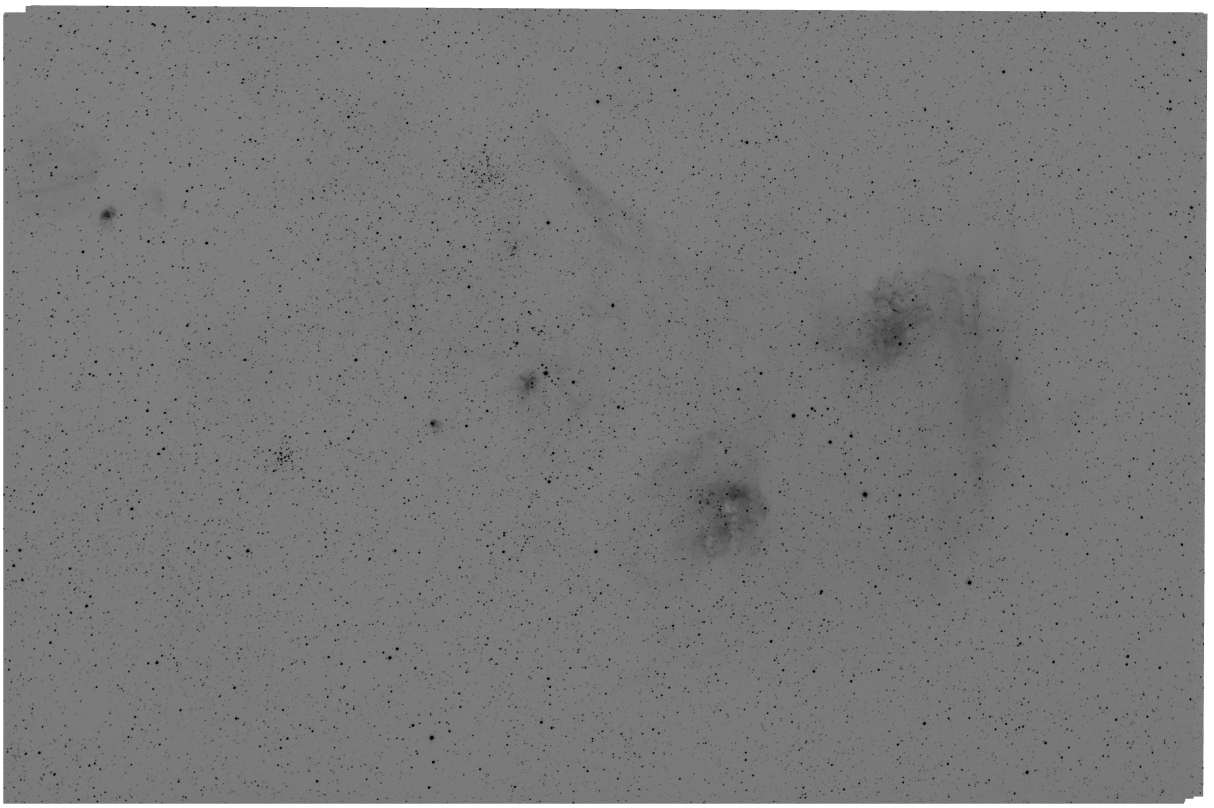

Figure B.1: The inverted original image.

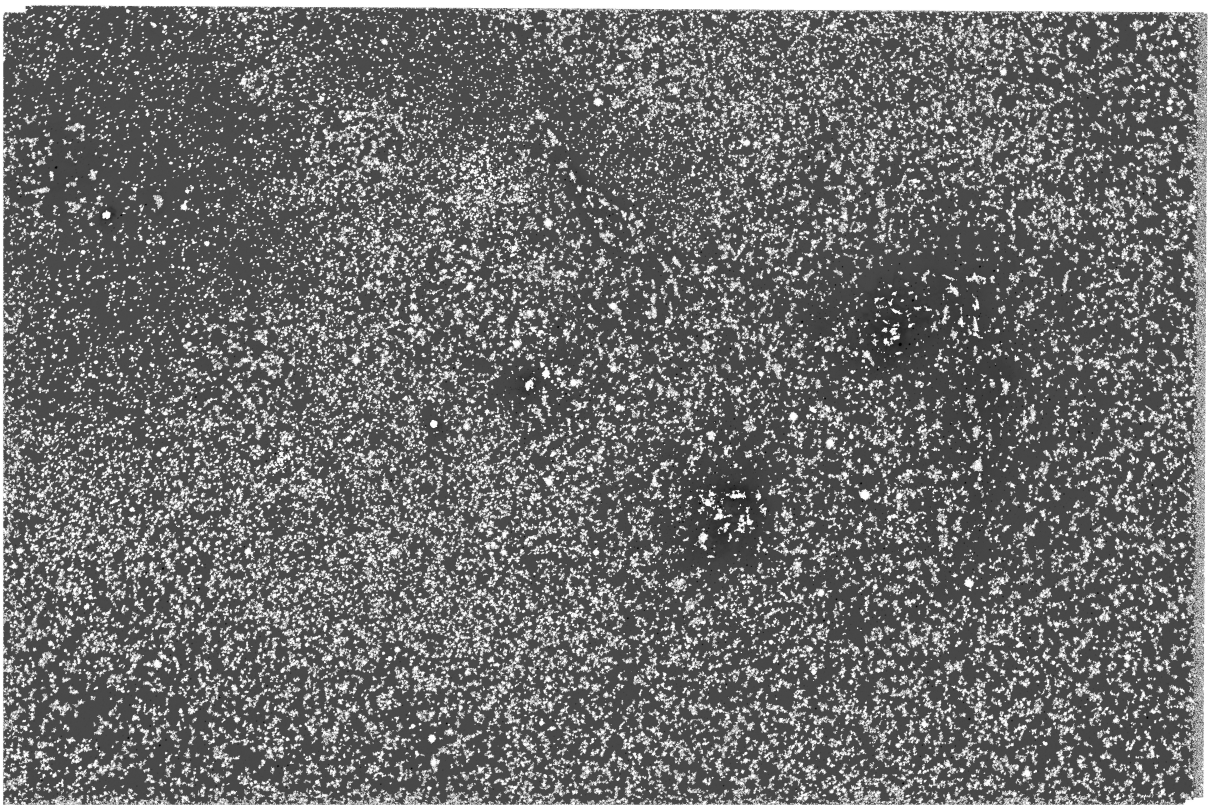

Figure B.2: The image with its stars removed.

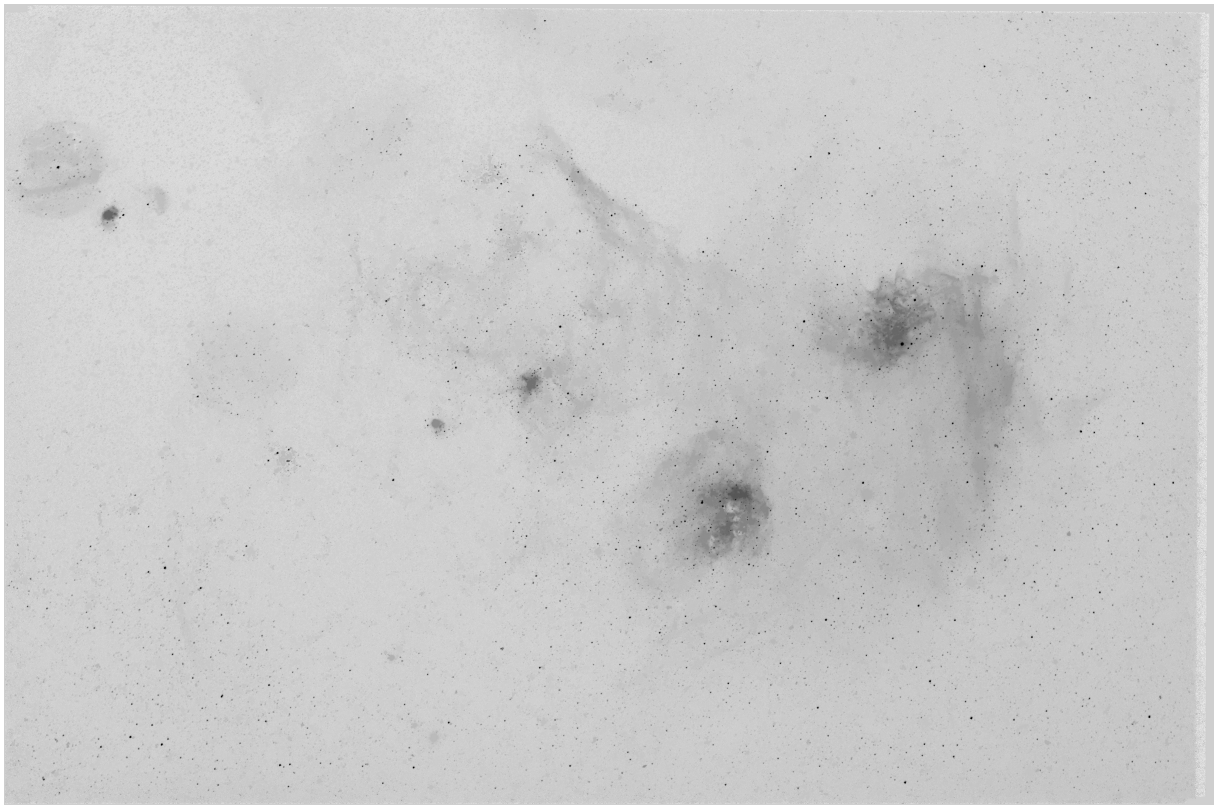

Figure B.3: The image with its stars filled in through the MaxTree structure.

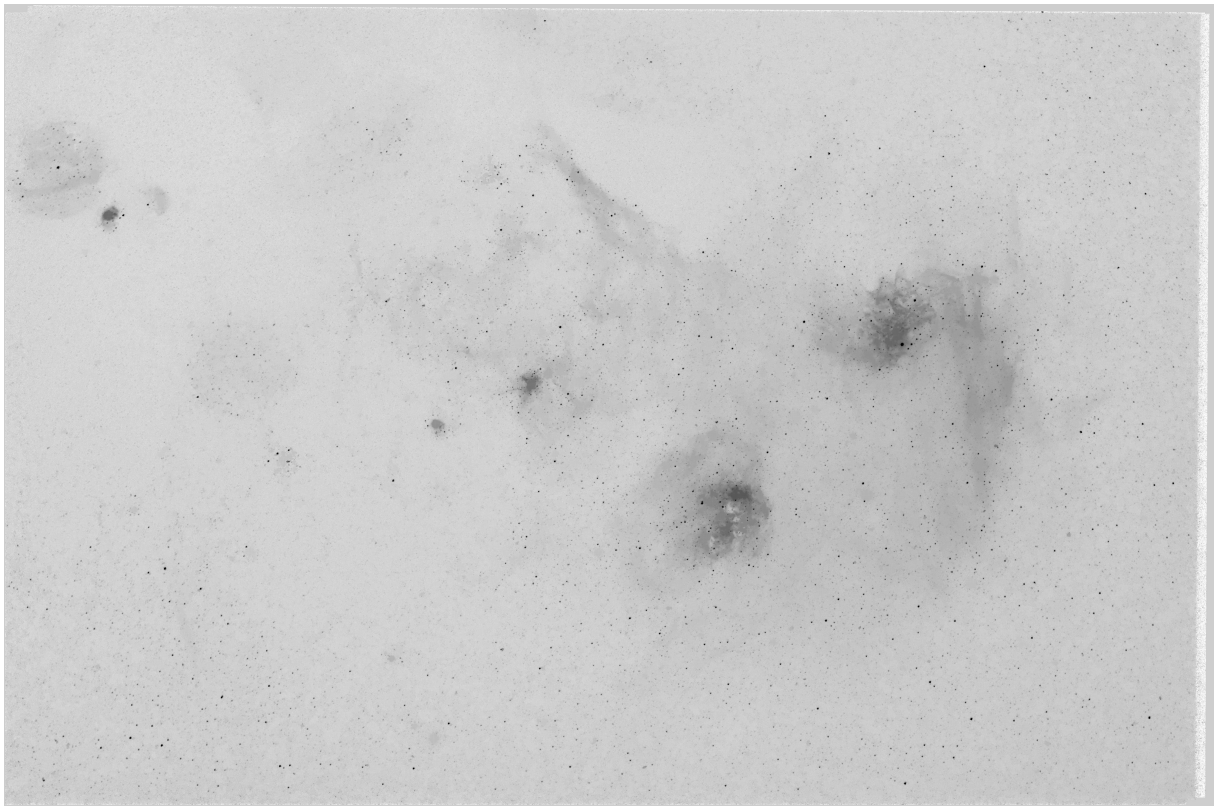

Figure B.4: The image with its stars filled in through linear interpolation.