

Bachelor Project Computing Science

Adaptive Unsharp Masking for Image Enhancement

Student: Rick de Jonge, s2775832

University: Rijksuniversiteit Groningen

Course code: WBCS13000

Primary supervisor: Dr. MHF Wilkinson

Secondary supervisor: Prof. Dr. M. Biehl

July 19, 2018

Contents

1	Introduction	4
2	Unsharp Masking	4
3	Related concepts	5
3.1	Weber's law	5
3.2	The Machband effect	5
3.3	Histogram Stretching	6
3.4	Sobel and Laplacian Edge Detectors	7
4	Existing Unsharp Masking methods	7
4.1	Traditional Unsharp Masking	7
4.2	Cubic Unsharp Masking	8
4.3	Adaptive Unsharp Masking	8
4.4	Non-linear Unsharp Masking	9
4.5	A Rational Unsharp Masking Technique	9
4.6	Unsharp Masking using segmentation	10
5	Results of the Existing Methods	10
5.1	Traditional Unsharp Masking	11
5.1.1	Conclusions	11
5.2	Cubic Unsharp Masking	13
5.2.1	Conclusions	13
5.3	Rational Unsharp Masking	13
5.3.1	Conclusions	14
6	Adapting Existing Methods	14
6.1	Larger Implementations of the Basic Method	14
6.1.1	Comparisons	16
6.1.2	Conclusions	16
6.2	Larger Implementations of CUM	16
6.2.1	Comparisons	19
6.2.2	Conclusions	19
6.3	Adapting Rational Unsharp Masking	19
6.3.1	Comparisons	20
6.3.2	Conclusions	21
7	Sharpening Astronomy Images	21
7.1	2D Cubic Unsharp Masking	26
7.2	Rational Unsharp Masking, using improved second term Cubic Unsharp Masking	29
8	Comparing previous results to results using ImPPG	31

9 Thesis Conclusions	33
10 Future work	34
10.1 RL Deconvolution	34
10.2 Connected operators	35
10.2.1 Connected flat zones	35
10.2.2 Tree representations	35
11 Code attachments	37

Abstract

This thesis will start with exploring the traditional method of Unsharp Masking. Next, it will discuss a few improvements of the traditional method that can be found in the literature. After these discussions, comparisons will be made between the results of those algorithms to the results of my own adaptations of these methods using a basic image. The algorithms with the best results will then be tested on a few more relevant images. Finally, a small comparison will be made between these results and the results from a program called *ImPPG*, that has a build-in version of Unsharp Masking. Concluding, there will be a section that explains which results failed, were successful and which particular method looked most promising.

The main question this thesis will be answering is: "In what ways can the Unsharp Masking image sharpening method be improved upon?"

1 Introduction

In this document I will present the findings I did while working on my 2018 bachelor's thesis, in order to answer the following question: "In what ways can the Unsharp Masking image sharpening method be improved upon?".

The images needed in certain fields, like astronomy, need to have clear details. Some images however can be blurred, by for example the diffusion of light. Unsharp Masking is a technique that can be used to make such images sharper. Unsharp Masking has various different working implementations already, but this field is far from optimized. In order to expand the knowledge of this particular image sharpening algorithm, I will test and try to improve the existing techniques. This technique is already being used in image editing programs such as *Photoshop* and camera applications on phones.

Since the research question tries to explore Unsharp Masking with better results, the research I did focused on optimizing performance rather than optimizing computation time. This means that I did not alter the program to reduce the time it takes to process an image, but thought of different ways to make the existing algorithms more effective at what they already do. I implemented my solutions in *Python*, mainly because it has easy ways of loading and changing images and I am already very familiar with the language. In the last section I will attach my implementation of the methods described in this paper so that this research can be extended and verified if desired.

2 Unsharp Masking

The Unsharp Masking technique can increase the quality of the details of an image by using a lower detailed version of that image. It improves the visual quality of the image using high-pass filtering and dynamic range compression. In more detail, the lower quality image is obtained by averaging each pixel with its neighbours, using a multiplier that is based on a Gaussian curve. The pixels of the original image are then added to a small portion of the pixels in the created image, which results in a clearer image. The expected result on an image of an eye is shown in the top images of Figure 1.

There are several drawbacks of Unsharp Masking. It will amplify all details, whether it is noise or not. It also will leave ringing artifacts around objects in the image. There are several techniques that modify Unsharp Masking in a way that benefits certain images, for example making the algorithm more noise resistant. An extreme example of the effect of Unsharp Masking on a blurred noisy image can be seen in the bottom pictures of Figure 1.

There are many specific algorithms using Unsharp Masking for specific fields, such as medicine [1]. In this document I will primarily look at Unsharp Masking techniques that would benefit astronomy. These images have in common that there are a few bright areas and a big area of black with low-intensity lights.

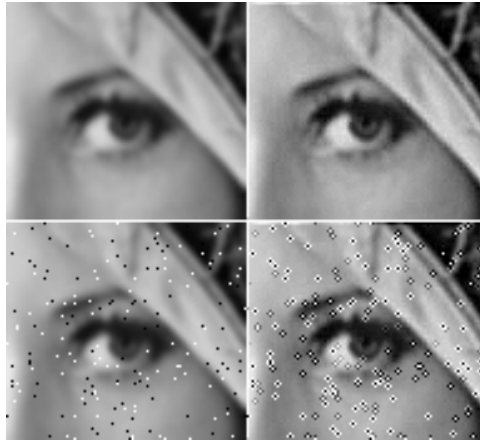


Figure 1: The effect of Unsharp Masking on a blurred image (original top-left, result top-right), and the effect it has on noise (original bottom-left, result bottom-right)

3 Related concepts

First, I will explain a couple concepts related to Unsharp Masking and image enhancement in general.

3.1 Weber's law

Weber's law states that the perception of increase is relative. For image enhancement this means that halving the amount of noise for example will seem the same whether there is a lot or a little bit of noise. For the Unsharp Masking algorithms this means that algorithms that just focus on removing the majority of noise are preferable to algorithms that remove all the noise at the great cost of performance. To illustrate this, while the difference from the two left images in Figure 2 to the two right pictures are both 10 pixels, the difference between the top images is much more noticable.

3.2 The Machband effect

When two areas with different luminance meet, our eyes will increase the contrast at the border, as can be seen in Figure 3. This is called the Machband effect. This has some implications for image enhancement. The Machband effect will increase our perception of noise around borders in images, which is one of the big downsides of the Unsharp Masking technique.

The Machband effect is very similar to Unsharp Masking itself, visualizing points while taking information from surroundings. This may be why Unsharp Masking is visually pleasing, it may change images in the same way we do.

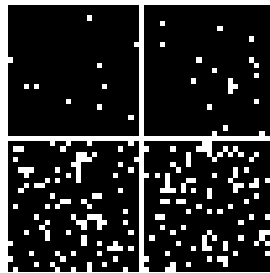


Figure 2: Illustration of Weber's law:
top-left: 10 white pixels, top-right: 20 white pixels
bottom-left: 100 white pixels, bottom-right: 110 white pixels



Figure 3: Illustration of the Machband effect

3.3 Histogram Stretching

Histogram stretching as described in [3] is a technique to redistribute an image based on the contrast. In this technique, a histogram is made of the images grey-levels. The histogram of images this technique is useful on will have a lower and upper are with almost no values. A lower and upper bound are then decided upon and the histogram is redistributed where the lower bound will become the new least value and the upper bound the new highest value. This is a most basic technique on which some of the later described Unsharp Masking algorithms are based. Figure 4 is an example of histogram stretching where the original (left) has a lower upper bound than the result (right), resulting in every pixel becoming lighter.

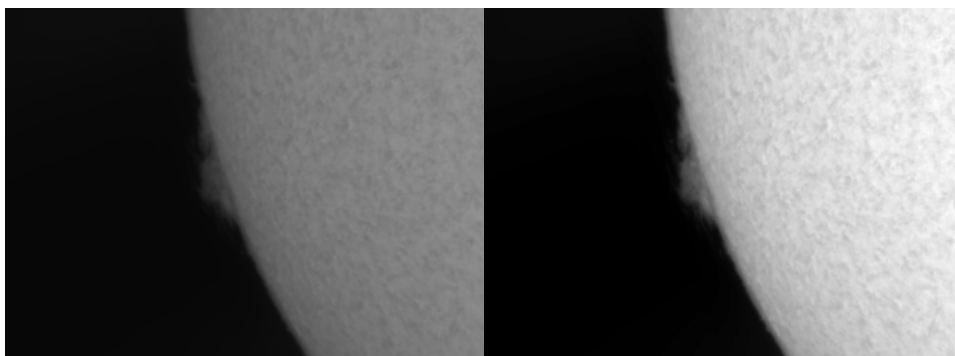


Figure 4: Illustration of histogram stretching



Figure 5: The original image (left) processed with the Sobel edge detector (middle) and the Laplacian edge detector (right)

3.4 Sobel and Laplacian Edge Detectors

Since an aim of our new approach is to decrease ringing artifacts around edges in the image, it is good to look at a pair of well known edge detectors. Both the Sobel and Laplacian edge detectors [3] scans each pixel and looks at the surrounding values to find high differences between neighbouring pixels. The Laplacian edge detector compares the current pixel with the pixels horizontally and vertically next to it. The Sobel detector is different in that it only looks for either differences horizontally or vertically, it looks at 2 opposite neighbouring pixels and the difference between their neighbours. This marks the biggest difference between the two approaches. The Sobel edge detector is a second derivative, while the Laplacian edge detector is a first derivative. This can be noticed in Figure 5, where the edges detected by the Sobel edge detector are thicker. Furthermore, because the Laplacian method only compares with horizontal and vertical neighbours, diagonal edges are harder to detect and are less clear. The edges in the Sobel edge detector are clear in all directions.

4 Existing Unsharp Masking methods

There are plenty of improvements that build upon Unsharp Masking. Certain algorithms have a special way of calculating the mask that will be added. Others introduce new terms into the standard equation, and there are methods that take information from the picture as a whole to improve the results. In this section I will show examples of a few of those improvements upon Unsharp Masking, going into detail what approach they take.

4.1 Traditional Unsharp Masking

The basic formula for creating the image is

$$y_{n,m} = x_{n,m} + \lambda z_{n,m} \quad (1)$$

Here, y is the output image, x is the original image and z is the blurred image constructed using

$$z_n = 2x_n - x_{n-1} - x_{n+1} \quad (2)$$

for 1D approaches and

$$z_{n,m} = 4x_{n,m} - x_{n-1,m} - x_{n+1,m} - x_{n,m-1} - x_{n,m+1} \quad (3)$$

for the 2D approach. This most basic form of Unsharp Masking creates the mask by taking information from the surrounding pixels to create a blurred image.

4.2 Cubic Unsharp Masking

The Cubic Unsharp Masking operator [9], abbreviated *CUM*, is an Unsharp Masking technique that aims to reduce noise sensitivity severely. In addition to the standard filter that subtracts a portion of the surrounding pixels to gain the blur effect, the CUM filter multiplies the resulting value with the square of the difference between the opposite surrounding pixels. This formula looks like

$$z_n = (x_{n-1} - x_{n+1})^2 (2x_n - x_{n-1} - x_{n+1}) \quad (4)$$

where the image z_n makes up the image based on the coordinate x . This makes the filter perform better in high gradient parts of the image and less susceptible to noise. The downside of this filter will then be in the lower gradient parts of the image, which may be regarded on the same level as noise by the filter. This method can be applied in both horizontal, vertical, both or diagonally for different results. The 2-D formula would look like

$$z(n) = (x_{m-1,n} - x_{m+1,n})^2 (2x_{m,n} - x_{m-1,n} - x_{m+1,n}) + (x_{m,n-1} - x_{m,n+1})^2 (2x_{m,n} - x_{m,n-1} - x_{m,n+1}) \quad (5)$$

for the horizontal and vertical version, and like

$$z(n) = (x_{m-1,n} + x_{m+1,n} - x_{m,n-1} - x_{m,n+1})^2 (4x_{m,n} - x_{m-1,n} - x_{m+1,n} - x_{m,n-1} - x_{m,n+1}) \quad (6)$$

for the diagonal version, Inseparable Cubic Unsharp Masking.

4.3 Adaptive Unsharp Masking

The Adaptive Unsharp Masking technique [7] differs from linear Unsharp Masking by changing the weight of the high-pass image dynamically. This results in better results in both high and low contrast areas of the original image. The values to be averaged will be taken from horizontal, vertical and diagonal directions for the total of nine pixels. Each pixel will then be classified in one of three levels, signifying how much contrast the area contains and thus the weight value of the

high-pass image. The algorithm then assigns weight values for each pixel, and finally computes the resulting image. This algorithm can give sufficient results on a wider range of images since it has multiple adjustable variables, but also lacks in focus to achieve better results consistently compared to the other algorithms it was compared to in the paper. The tests showed that the images were less noisy overall and worked well in more detailed portions of the image. The worst results could be observed between smooth and detailed parts of the image.

4.4 Non-linear Unsharp Masking

Non-linear Unsharp Masking as proposed in [2] uses quadratic filters [8] to combat noise. This method replaces the standard high-pass image from Formula 1 with a normalized quadratic filter, like in the formula

$$z_n = \sum_{i=-M}^{i=M} h(i)x(n-i)x(n+i) \wedge \sum_{i=-M}^{i=M} h(i) = 0 \quad (7)$$

This approach proved to be useful for a wide range of images, without the need to adjust parameters such as the standard weight factor. The same paper [2] also shows an algorithm to specifically tackle really noisy images. Instead of using the original image, this algorithm uses a low-pass added to an image created by multiplying a pass using a 3×3 Laplacian filter with a Sobel filtered image [3, 4], and the usual weight factor. This approach works because it uses both high- and low-pass images, one horizontally and one vertically, to sharpen the image.

4.5 A Rational Unsharp Masking Technique

Another of Ramponi's papers [10] proposes adding a Laplacian filter multiplied by a polynomial instead of the blurred image, along with the standard λ multiplier, as in

$$y(n) = x(n) + \lambda c(n)L(n) \wedge c(n) = \frac{g(n)}{kg^2(n) + h} \quad (8)$$

The polynomial is based on the local signal activity and has the property of generally being low when noise is present. This function sets itself apart from previously mentioned algorithm because it has two adjustable values, namely k and h , to make the choice between low noise susceptibility and low ringing effects. The algorithm was developed for a single dimension and later extended to two dimensions with good results.

The proposed formula to calculate the local activity, $g(n)$ in the previous formula, is as follows

$$g(n) = (x(n+1) - x(n-1))^2 \quad (9)$$



Figure 6: The original Lenna picture

4.6 Unsharp Masking using segmentation

An approach from the medical field [12] could be build upon using connected operators. The original algorithm separates the image into three parts, based on the amount of detail in the area. The paper then introduces an improved high pass filter, which is obtained by subtracting a low-pass filtered image from the original image. A big difference with previously shown methods is that this filter averages over a 5×5 area instead of the usual 3×3 . This approach was shown to keep clear edges of mammographic masses which are, similarly to astronomy, light areas in a bigger dark area.

5 Results of the Existing Methods

The older papers the methods were presented in had poor quality. In order to see a bigger picture and to better compare my own results with the original image, I will implement older techniques again and describe my own general findings using those methods on the well known Lenna picture, an image of 512 by 512 pixels, see Figure 6. After that I will use the techniques on photos of sun prominence, to see if the methods work as well on astronomy pictures.

The results that will be presented were judged by visual inspection, because a high priority of the results is that they look visually pleasing. The basis for the judgment of the images in a mathematical or other scientific method is hard to define and therefore not included in this thesis.



Figure 7: The Lenna picture, processed with traditional Unsharp Masking, $\lambda = 1$ (left), and traditional Unsharp Masking, horizontally and vertically, $\lambda = 0.5$ (right)

5.1 Traditional Unsharp Masking

The best results using the traditional algorithm, applied only horizontally, were while setting the variable λ around 1. The resulting picture, the left picture in Figure 7, is sharper than the original, Figure 6. This can be seen best around the shoulder, eyes and especially the hair. The most prevalent side-effects are the increase in contrast, most notably the hat, and increased noise over the whole of the picture.

The 2-dimensional version of the traditional method performed best around $\lambda = 0.5$. The right picture in Figure 7 is very comparable to the picture resulting from horizontally applied Unsharp Masking. The most notable differences are in the eyes and the hair. The hair is even sharper, because of the vertical alignment in the picture. The overall picture is also a little bit sharper at the exact value of λ I used here, but that may differ when adjusted.

5.1.1 Conclusions

The traditional method makes the image clearer, as is its intention, but has major problems with noise in the image and increases contrast a lot at sharp boundaries.

When setting λ at an unusually high number, see Figure 9, the Lenna picture seems to obtain a pattern of cubes, while the sun prominence picture only gains an unusual pattern in the darkest area of the picture. This is most probably the result of the noise of the way the prominence picture was taken or, in the case of Lenna, the result of the original image being in the JPEG format, which holds less information than the resulting PNG image.



Figure 8: Difference between the pictures in Figure 7 and the original Lenna picture

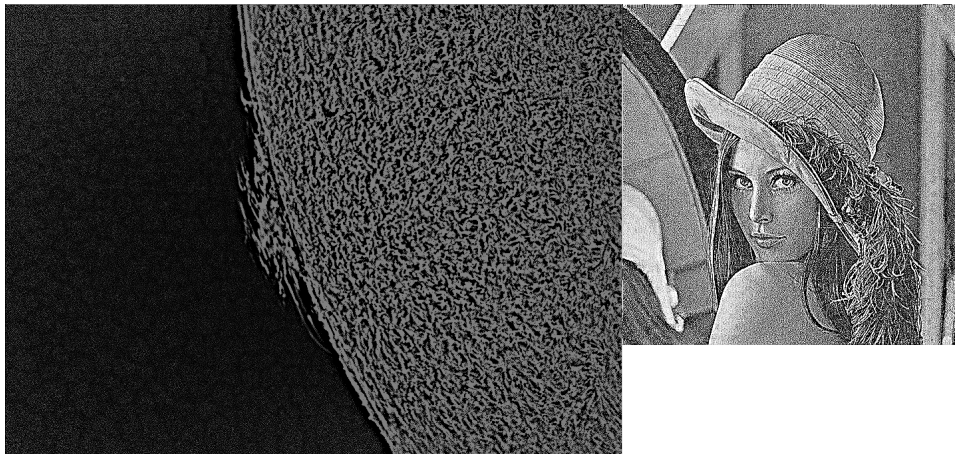


Figure 9: An extreme of processing with traditional Unsharp Masking in 2 dimensions, at $\lambda = 1000$ (left) and $\lambda = 100$ (right)



Figure 10: The Lenna picture, processed with Cubic Unsharp Masking, horizontally, $\lambda = 0.005$, (left), and Cubic Unsharp Masking, horizontally and vertically, $\lambda = 0.002$, (right)

5.2 Cubic Unsharp Masking

Using the horizontally applied Cubic Unsharp Masking algorithm, the left picture in Figure 10, we can obtain a similar sharpness using $\lambda = 0.01$. In this figure, you can see that the noise in the background has definitely decreased, while the foreground keeps the sharpness. I do notice a decrease in quality when an area has a lot of contrast changes, for example in the hat.

When comparing 1D to 2D traditional Unsharp Masking, Figure 7, we can see the same improvements as when comparing the pictures resulting from Cubic Unsharp Masking and 2-dimensional Cubic Unsharp Masking, Figure 10. The background noise has increased, while Lenna herself has increased in sharpness. When comparing horizontal CUM with horizontal and vertical CUM, we can see that, like with the traditional approach, the hair has severely increased in contrast, but with that also a few more white spots became more notable in the background.

5.2.1 Conclusions

Cubic Unsharp Masking is a much more effective method than traditional Unsharp Masking. It is a lot less sensitive to noise overall, although the noise it did react to is now amplified, resulting in better results in flat areas in the image.

5.3 Rational Unsharp Masking

When comparing 1 dimensional Rational Unsharp Masking to the original Lenna picture as in Figure 8, we can see that the effect of the chosen variables give an effect comparable to using CUM, as in Figure 10. The 1-dimensional version of



Figure 11: Difference between the pictures in Figure 10 and the original Lenna picture

Rational Unsharp Masking has more differences with the original picture compared to the 1-dimensional version of CUM. The 2-dimensional result however has a similar effect in high-detail areas, such as Lenna's hair, but shows better results in flat areas. Also notable is the much clearer definition of the two white beams in the right and middle of the image.

5.3.1 Conclusions

Using the Rational Unsharp Masking method results in better looking flat-areas compared to CUM, but at the cost of having to adjust more variables to achieve that result.

6 Adapting Existing Methods

In this section, I will show you adaptations of the methods described above. I will explain what changes I made, why I made them and compare the results to the original methods.

6.1 Larger Implementations of the Basic Method

For the first adaptation, I decided to extend the basic 2-dimensional formula to also take into account the diagonally adjacent pixels, inspired by the approach given in the paper on Unsharp Masking for Mammographic Masses [12]. I hope to improve the stability of the algorithms by doing this. I will also take various weights into account.



Figure 12: The Lenna picture, processed with Rational Unsharp Masking horizontally, with $\lambda = 2$ (left), compared to horizontally and vertically, with $\lambda = 1$ (right). In both cases holds: $k = 0.001$ and $h = 250$



Figure 13: The difference between the pictures of Figure 12 and the original Lenna picture

Table 1: The kernels of the first (left) and second (right) adaptation of 2-dimensional Unsharp Masking

-1	-1	-1	-1	-2	-1
-1	8	-1	-2	12	-2
-1	-1	-1	-1	-2	-1

Table 2: The kernels of the third (left) and fourth (right) adaptation of 2-dimensional Unsharp Masking

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-2	-2	-2	-1
-1	-1	24	-1	-1	-1	-2	32	-2	-1
-1	-1	-1	-1	-1	-1	-2	-2	-2	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

6.1.1 Comparisons

I used the weight formulas described in Table 1 and Table 1 to create the pictures shown in Figure 14. As can be seen in Figure 14, adding diagonal pixels to the calculation added next to nothing to the result except computing time. The lines on the hat seem to be a bit more black in the left picture, with even weight spreading, than in the right picture, where diagonal pixels had less weight. This is because the lines on the hat are diagonally placed. From this I could conclude that small adjustments can be made easily to the original method for pictures with a high ratio of a similar pattern.

6.1.2 Conclusions

Extending the traditional method by increasing the area of the edge detector has some small improvements to the result of the algorithm. The main conclusion however is that this filter is highly adaptable. If the image to be sharpened has some a certain pattern (in certain areas), this table can be easily adapted to optimize results for that pattern.

6.2 Larger Implementations of CUM

With this adaptation, I changed variables in the CUM formulas, namely the power and the pixels that are in the power section. By changing the importance of the added function I wanted to amplify the effects this method has, and by changing the pixels it looks at I wanted to decrease the local nature, and with that the susceptibility to noise, of the formula.



Figure 14: The Lenna picture, processed with Traditional Unsharp Masking using equal weights, as in Table 1, $\lambda = 0.2$ (top-left), and processed with less weight diagonally, as in Table 1, $\lambda = 0.2$ (top-right), with a 5×5 system as in Table 2, $\lambda = 0.05$ (bottom-left), and with a weighted 5×5 system as in Table 2, $\lambda = 0.03$ (bottom-right)



Figure 15: Difference between the pictures in Figure 14 and the original Lenna picture



Figure 16: The increased significance of the power term of the CUM method, with $\lambda = 0.005$ (left) compared to the increased range from one to two pixels, with $\lambda = 0.001$ (right)

6.2.1 Comparisons

Contrary to what I suspected, increasing the significance of the power term, as can be seen in Formula 10, makes the image a bit sharper, but also increases the susceptibility to noise by a fair amount.

$$z_n = (x_{n-1} - x_{n+1})^4(2x_n - x_{n-1} - x_{n+1}) \quad (10)$$

Extending Cubic Unsharp Masking to look at the two neighbouring pixels using

$$z_n = (x_{n-1} - x_{n+1})^2(4x_n - x_{n-1} - x_{n+1} - x_{n-2} - x_{n+2}) \quad (11)$$

however, resulted in a picture that remained as sharp as the original version of CUM. This picture also had less unusual high-contrast points on the hairs and around the eyes. The pictures of both adaptations can be found in Figure 16.

6.2.2 Conclusions

While increasing the wight of the first term seemed to only decrease performance, increasing the pixels of the second term had the expected effect of increased detail in high areas and did not create as much extremes as the original method.

6.3 Adapting Rational Unsharp Masking

Since Rational Unsharp Masking makes use of the standard Laplacian term, it can easily be extended with a few of the methods I used to adapt the previous methods.



Figure 17: Difference between the pictures in Figure 16 and the original Lenna picture

It also can replace the Laplacian term entirely with one of the previous methods. Next to that, the function $g(n)$ of the standard Rational Unsharp Masking formula, Formula 8, can be adapted as well. The original paper [10] only shows two versions of $g(n)$ of the most basic variant. Finally, the original paper suggest a 2-dimensional variant in which the algorithm regards the horizontal and vertical lines separately. I will also approach this differently by computing $g(n)$ only once for both directions.

6.3.1 Comparisons

Two possible extensions could be extending $g(n)$ in the horizontal direction using the formula

$$g(n) = (x(n+1) - x(n-1) + x(n+2) - x(n-2))^2 \quad (12)$$

In the top-left of Figure 18 you can see that this effect is very minimal. The most significant change can be observed in the left of the picture, where there is a bigger unaffected area compared to the traditional method. The rest of the picture has a better effect than the traditional method, but a worse effect compared to the presented version of Rational Unsharp Masking. This is probably because the algorithm is still only applied horizontally and because the picture is relatively small, and therefore has more details close together.

The second version shown in Figure 18 is while using the formula

$$g(n) = (x(n+1, m) - x(n-1, m) + x(n, m+1) - x(n, m-1))^2 \quad (13)$$

I implemented this version, because extending the rational control term could increase the performance in the more detailed areas of the image. This was indeed the case when we compare the result in her hair. This improvement already comes very close to 2-dimensional Rational Unsharp Masking. This could mean a slight decrease in execution time if needed.

The bottom-left image in Figure 18 was created by replacing the standard Laplacian operator by the 1-dimensional Cubic Unsharp Masking operator. This result is very comparable to traditional CUM, but performed a little better in the most detailed areas of the image, like the eyes and in the hair.

The result was even better when using the 5×5 Laplacian operator instead of CUM. This is probably because the rational control term and the cubic term have similar functions, while the 5×5 Laplacian term does the basic function of determining whether there is noise in the current area better than the original Laplacian operator. In the resulting picture, the details on the hat and the distinction between strands of hair were more clear than the previously shown adaptation with CUM.

Since the paper [10] calculated the 2D version of Rational Unsharp Masking for horizontal and vertical directions separately, I wanted to compare this with a version that would calculate this in a single formula. This calculation speed optimization has too much loss in performance, however, see the top pictures in Figure 20. Calculating separate rational control terms significantly decreased the sensitivity to noise in areas without much contrast.

After that, I used the CUM method, both the original 2-dimensional version and my second term adaptation to create the bottom images in Figure 18. This resulted in the best pictures so far, with the algorithm having great performance in both detailed and flat areas of the image and having less extreme white highlights.

6.3.2 Conclusions

Improving the 1-dimensional version of Rational Unsharp Masking had only minor benefits. Using a 2-dimensional version of the Rational Unsharp Masking method, and therefore using this formula as a basis to insert previously shown techniques into, proved more successful. Determining the local variance and noise separately for horizontal and vertical directions worked much better than trying to generalize these directions into a single rational control term and Laplacian operator.

7 Sharpening Astronomy Images

We have seen before that the Unsharp Masking technique and all its adaptations have visually pleasing results on a general image like the Lenna image. Now I would like to inspect if the same good results can be achieved on pictures that are used in current research, like pictures of sun prominence used in astronomy. These images are a bit different. Aside from the subject in the image, they were saved in the TIFF format, a format that can hold more information than the previously



Figure 18: A comparison of four Lenna images, processed with Rational Unsharp Masking, with a wider $g(n)$, $\lambda = 2$ (top-left), where $g(n)$ is 2 dimensional, $\lambda = 1$ (top-right), using the wider adaption of the CUM method, $\lambda = 0.005$ (bottom-left), and where the Laplacian operator has been switched with the 5×5 version, $\lambda = 0.1$ (bottom-right)



Figure 19: The difference between the original Lenna picture and the images of Figure 18



Figure 20: The 2 dimensional version of the Rational Unsharp Masking technique, as shown in the paper, $\lambda = 1$ (top-left) and calculated in a single calculation, $\lambda = 1$ (top-right), using CUM, $\lambda = 0.01$ (bottom-left) and using the second term adaptation of CUM, $\lambda = 0.003$ (bottom-right)



Figure 21: The difference between the original Lenna picture and the pictures of Figure 20

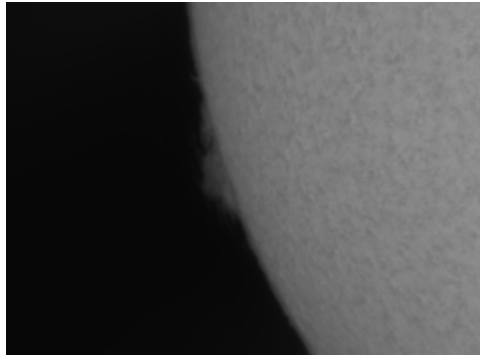


Figure 22: A sun prominence, very visible but blurred

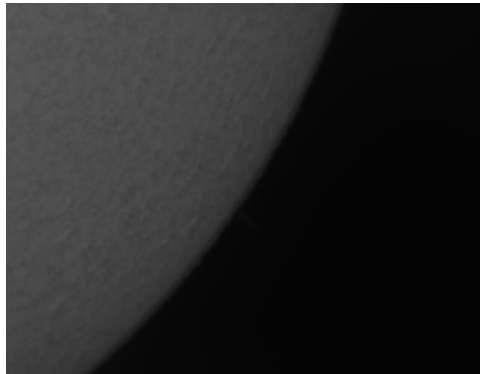


Figure 23: A sun prominence, neither very visible nor sharp

used JPEG and PNG formats. In this section, I will look at three images of sun prominence, see Figures 22, 24 and 23. These images have three different levels of sharpness to begin with and with them we can see whether the algorithm also works for very subtle parts in the image.

Since I already have shown before which algorithms will most likely have the best results, I will only show a few algorithms that have shown the best results with the prominence images.

7.1 2D Cubic Unsharp Masking

Since 2D CUM showed a good base for comparing the best results, I decided to start applying that to the first prominence image. The TIFF format however did not have the same results as the PNG format shown before. The resulting image is definitely sharper than the original when we look at the prominence itself. When we look at the amount of work the algorithm had to do with the right of the image, we can see that the whole of the sun has a lot of differences with the original, where the details of the sun were decreased in intensity by the mask. The dark area on the left was also affected, which was predictable. The original image had

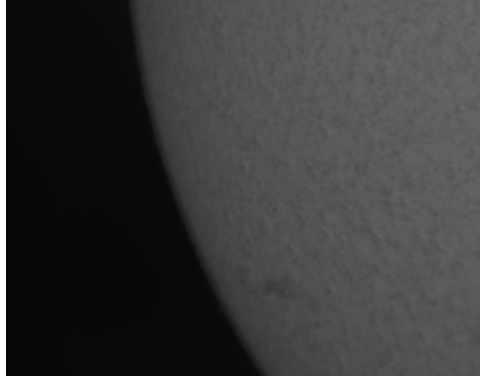


Figure 24: Multiple small prominence at the blurry edge of the sun

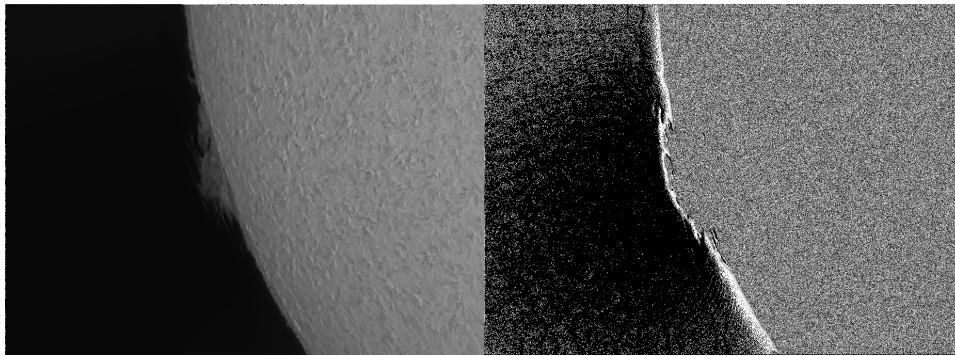


Figure 25: Figure 22, processed with 2D CUM, $\lambda = 0.00015$ (left) and the difference with the original picture (right)

some distinct sections in that area, which should have been a flat area. The result of the most interesting area, the prominence, was much clearer but has a clear high intensity edge. This edge also has clear black spots all over.

When we use Cubic Unsharp Masking on only a part of the first picture that has been converted to the PNG format, it made the edges of the sun a lot more clear. As can be seen in Figure 26, the algorithm mostly just sharpened the edge of the sun, the prominence and some of the irregularities on the surface of the sun. Increasing λ results in more extreme white highlights in these areas, so a λ of 0.03 gave the best results.

When applied to Figure 23, 2D CUM is able to sharpen the prominence around the edge of the sun, but at a cost in overall quality. As seen in the right of Figure 27, the algorithm has to apply a relatively large λ which increases noise in the resulting image.

Since the most interesting parts of Figure 24 are the barely visible prominence, I will not compare the picture as a whole but look at those details. I have processed the whole picture, then cropped a part of that image to show results for. Unfortu-

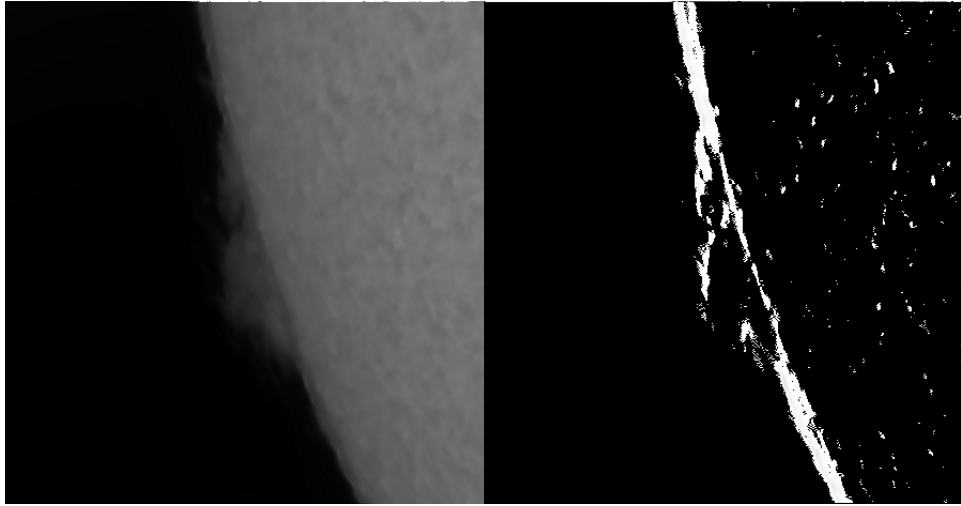


Figure 26: A smaller piece of Figure 22 converted to PNG and then processed with 2D CUM, $\lambda = 0.03$ (left) and the difference with the original picture (right)

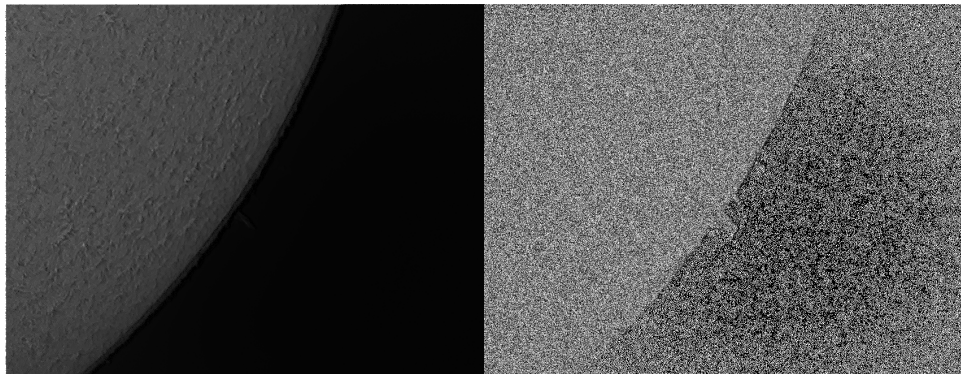


Figure 27: A smaller piece of Figure 23 converted to PNG and then processed with 2D CUM, $\lambda = 0.0002$ (left) and the difference with the original picture (right)

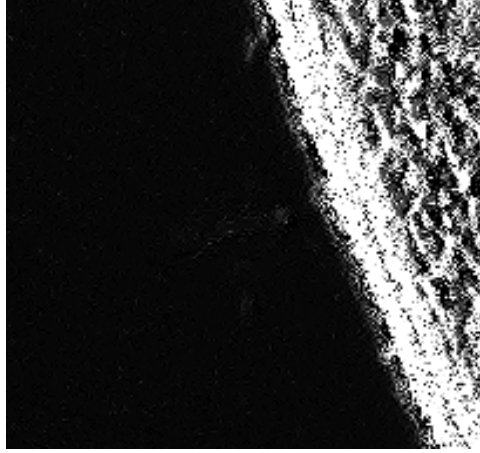


Figure 28: A smaller piece of Figure 24 processed with 2D CUM, $\lambda = 0.001$

nately, even with an extreme λ , namely 0.001, 2D Cubic Unsharp Masking could not make the prominence sharp. The prominence is barely visible compared to the noise in Figure 28.

7.2 Rational Unsharp Masking, using improved second term Cubic Unsharp Masking

Looking at Figure 29, we can see that, again, the surface of the sun is very detailed and therefore affected a lot by the algorithm. The dark area on the left half however has mostly been ignored because of the way the λ value changes based on the local activity.

When comparing this image to the previous, Figure 25, we can also see a difference between pictures in the area of the prominence. Where 2D CUM made the prominence into a more solid grey area, this version of Rational Unsharp Masking has more clearly defined black gaps and smaller prominence. The extreme white highlights are also not as bright as in the previous method.

Comparing Figure 30 to Figure 27, we can immediately see that the algorithm did not enhance noise as much in the large dark area. Besides that, the detail on the surface of the sun remained more detailed using Rational Unsharp Masking. Just by looking at the difference between the result from the adapted Rational Unsharp Masking and the original picture, we can spot a few prominence quite easily.

When we take a close-up of the biggest prominence in Figure 23, see Figure 31, we can see that the 2D CUM method has left a white overshoot around the biggest edge, while close to the edge there still is a more black area. In addition, the adapted Rational Unsharp Masking method has a more clearly defined edge and prominence.

Like in the previous section, using the adapted 2D Rational Unsharp Masking method, the prominence did not become visible at earlier used λ values. Using the

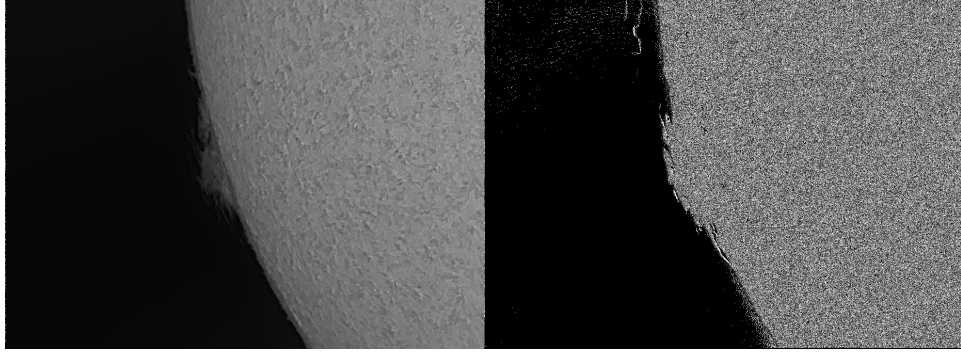


Figure 29: Figure 22, processed with my adaptation of Rational Unsharp Masking, using a modified second term CUM filter, $\lambda = 0.0005$, $k = 0.0000008$, $h = 150000$ (left) and the difference with the original picture (right)

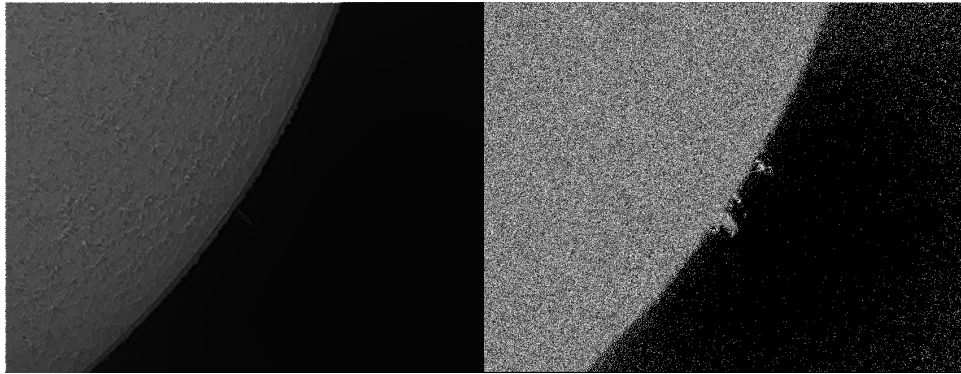


Figure 30: Figure 23, processed with my adaptation of Rational Unsharp Masking, using a modified second term CUM filter, $\lambda = 0.0005$, $k = 0.0000008$, $h = 150000$ (left) and the difference with the original picture (right)

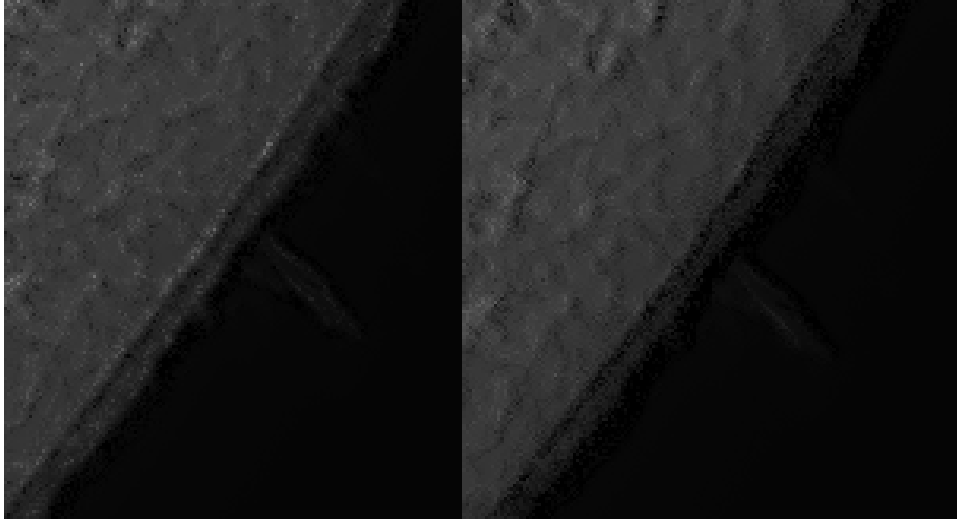


Figure 31: Comparing the area around the brightest prominence from Figures 30 (left) and 27 (right)

extreme $\lambda = 0.01$ resulted in a better quality image than using 2D CUM. As can be seen in Figure 32, processing with these parameters did not make the sun as bright and did not result in an extreme amount of noise. The prominence is still barely visible, but the result is better because of the lack of noise surrounding the prominence.

Since on the weakest prominence picture, Figure 24, the prominence is almost as weak as noise, using a variation of Rational Unsharp Masking that looks at a bigger area seemed appropriate. This will make sure that actual noise stays suppressed, while the prominence will remain visible.

8 Comparing previous results to results using ImPPG

Finally, I will share some results from a program that implements the Unsharp Masking method. This program allows the user to change two variables, namely sigma and amount. The amount variable seems the same, or at least similar, to the previously used λ . With this program, I tried to get the same results as in the previous section, using Figures 22, 24 and 23.

When we compare the result from *ImPPG*, Figure 33 to either Figure 25 or 29, we can see that the image processed by *ImPPG* is much smoother overall. The clear differences between the flat areas in the left of the images has completely smoothed out, and the pixels showing in either of the earlier pictures are not as visible. The results on the borders are comparable to those in Figure 29, but a little less defined.

In addition to being more smooth compared to the images in Figure 31, Figure

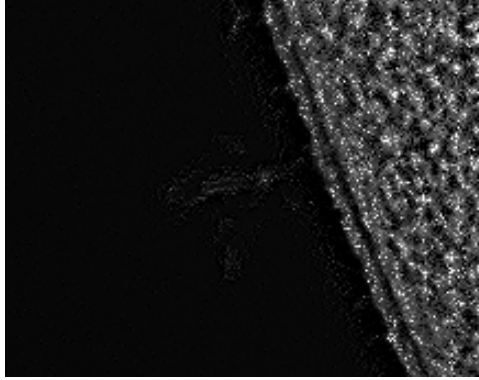


Figure 32: A smaller piece of Figure 24 processed with adapted 2D Rational Unsharp Masking, $\lambda = 0.01$, $k = 0.0008$, $h = 3000$

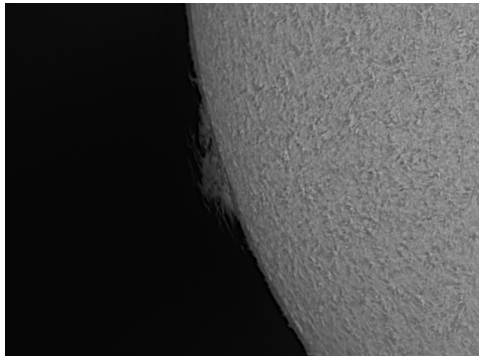


Figure 33: ImPPG's Unsharp Masking on Figure 22, $\text{Sigma} = 1.5$, $\text{Amount} = 20$

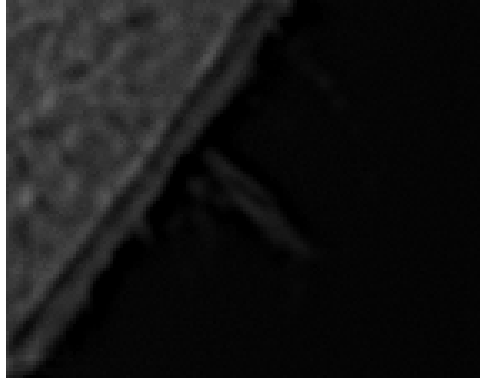


Figure 34: ImPPG's Unsharp Masking on Figure 23, $\text{Sigma} = 2$, $\text{Amount} = 20$

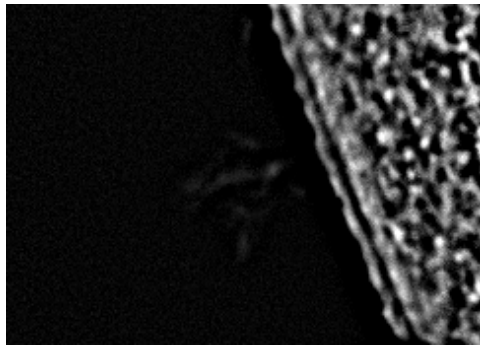


Figure 35: ImPPG's Unsharp Masking on Figure 24, $\text{Sigma} = 2$, $\text{Amount} = 20$

34 shows more of the white highlights that form the center of the prominence. This results in better visible smaller prominence surrounding the middle and most visible prominence.

Comparing Figures 32 and 35, we can see that, despite the extreme adaptation, the prominence in the Rational Unsharp Masking picture is better distinguishable than in the picture produced by *ImPPG*. The background in this last picture is also more distorted and noisy than the result of the Rational Unsharp Mask.

The program *ImPPG* uses a more demanding version of the Unsharp Masking technique, namely Gaussian Linear Unsharp Masking, to great effect. The resulting images are more smooth than the images resulting from my own implementations.

9 Thesis Conclusions

While traditional Unsharp masking is a fast and effective solution to blurry images, the algorithm can be improved in many ways. The algorithms can be adapted to be more effective against certain patterns in a picture or to make better distinction between what is noise and what are details.

Currently, there already exist multiple great improvements that will enhance the algorithm. I have found that these improvements are good stand-alone algorithms, but can be even more effective by combining them together, most notably the adaptation where Rational Unsharp Masking and Cubic Unsharp Masking were combined into a single algorithm. These algorithms can be significantly improved using various methods. Increasing the area around the current position to more accurately determine the local variance and having smarter ways of determining that variance in the first place are two concepts that can be used here.

We can conclude from the multitude of examples in this paper that the algorithm is highly adaptable and can therefore work for almost any picture it is used for. While this adaptability can show great results, it still has to be researched further for optimal results and to filter out more parameters that are not as beneficial to those results.

Existing programs that implement different adaptations of Unsharp Masking can probably also benefit from being combined with the adaptations presented here that have shown good results.

The answer to the question I posed at the start of the paper, "In what ways can the Unsharp Masking image sharpening method be improved upon?", can be summarized with the following: The Unsharp Masking technique can be adapted with multiple parameters and various optimizations, and while further research is needed the techniques already present can already combine to give good results.

10 Future work

All the comparisons and implementations I have shown in before all only just the surface of the Unsharp Masking technique. The methods I have implemented can be extended in more ways than I have shown and the methods I have only described can be researched in the same way. This means that I have only began this process.

From using *ImPPG* and looking at how easily you can see the pixels in my implementations, we can conclude that even my current implementations are far from perfect. They will need to be adapted to not show as much contrast between individual pixels.

10.1 RL Deconvolution

The Richardson-Lucy algorithm based on their papers[11, 5] could be used before applying any of the Unsharp Masking techniques. The algorithm iteratively increases the sharpness of an image in a different way than using an unsharp mask. Since this algorithm is used to great effect, combining the two techniques can result in better results. This algorithm is also build into the *ImPPG* program, but was not used for the previous results.

10.2 Connected operators

10.2.1 Connected flat zones

A different but related field I looked into, but did not get to implementing is about connected operators. Connected operators [6] can be useful in the Unsharp Masking algorithms because they can separate the image in multiple sectors based on a property. Since the main drawbacks of Unsharp Masking come from images with high contrast or gradients, isolating parts of the image with roughly the same contrast or gradient will reduce susceptibility to noise or ringing. The simplest form of a connected operator is as follows. The image is split into connected areas with the exact same value, named flat zones. These flat-zones can then be merged to partitions by combining flat zones where the values are within a certain threshold. The first connected operator only worked on binary images. It was used to remove the flat-zone by eroding the same-value pixels around the starting point.

10.2.2 Tree representations

Another way of separating the image into sectors is by using trees to represent them. There are different kinds of trees, namely the min-tree, max-tree and inclusion tree. In these trees, the nodes represent an area of the original image each. The root of the trees consist of the entire image and the leaves represent the maxima, minima or surrounded areas for max-trees, min-trees and inclusion trees respectively.

References

- [1] V. Digalakis et al. “Automatic adaptive contrast enhancement for radiological imaging”. In: *1993 IEEE International Symposium on Circuits and Systems*. 1993, 810–813 vol.1. DOI: 10.1109/ISCAS.1993.393846.
- [2] Sanjit K. Mitra Tian-Hu Yu Giovanni Ramponi Norbert Strobel. “Nonlinear unsharp masking methods for image contrast enhancement”. In: *Journal of Electronic Imaging* 5.3 (July 1996), pp. 353 –366.
- [3] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (2nd Ed)*. Prentice Hall, 2002.
- [4] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall Information and System Sciences Series. Prentice-Hall, 1989.
- [5] Leon B. Lucy. “An iterative technique for the rectification of observed distributions”. In: *The astronomical journal* 79.79 (1974), p. 745.
- [6] Michael H.F. Wilkinson Philippe Salembier. “Connected Operators”. In: *IEEE Signal Processing Magazine* (Nov. 2009), pp. 136 –157.

- [7] A. Polesel, G. Ramponi, and V. J. Mathews. “Image enhancement via adaptive unsharp masking”. In: *IEEE Transactions on Image Processing* 9.3 (2000), pp. 505–510. ISSN: 1057-7149. DOI: 10.1109/83.826787.
- [8] G. Ramponi and G. L. Sicuranza. “Quadratic digital filters for image processing”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 36.6 (1988), pp. 937–939. ISSN: 0096-3518. DOI: 10.1109/29.1611.
- [9] Giovanni Ramponi. “A cubic unsharp masking technique for contrast enhancement”. In: *Signal Processing* 67.2 (June 1998), pp. 211–222.
- [10] Giovanni Ramponi and Andrea Polesel. “A Rational Unsharp Masking Technique”. In: *Journal of Electronic Imaging* 7 (1998), pp. 333–338.
- [11] William Hadley Richardson. “Bayesian-Based Iterative Method of Image Restoration*”. In: *J. Opt. Soc. Am.* 62.1 (1972), pp. 55–59. DOI: 10.1364/JOSA.62.000055. URL: <http://www.osapublishing.org/abstract.cfm?URI=josa-62-1-55>.
- [12] Siddharth, R. Gupta, and V. Bhateja. “An improved Unsharp Masking algorithm for enhancement of mammographic masses”. In: *2012 Students Conference on Engineering and Systems*. 2012, pp. 1–4. DOI: 10.1109/SCES.2012.6199066.

11 Code attachments

Listing 1: Recreation of earlier methods and the adaptations on them

```
1 from PIL import Image
2 import numpy
3 import datetime
4
5
6 # Variable setup including source and resulting files and
  path
7 IMAGE = 'source_image_name'
8 TYPE = 'folder/type_name'
9 FILTER_LAMBDA = 1
10
11 if IMAGE.endswith('tif'):
12     COLOR_MAX = 65535
13     ARRAY_TYPE = "uint16"
14     SAVE_AS = f'images/{TYPE}/sun2_{FILTER_LAMBDA}.tif'
15 else:
16     COLOR_MAX = 255
17     ARRAY_TYPE = "uint8"
18     SAVE_AS = f'images/{TYPE}/sun1a_{FILTER_LAMBDA}.png'
19
20
21 # Accepts filepath to image
22 # Returns image, height, width
23 def load_image(path):
24     im = Image.open(path)
25     h, w = im.size
26     return numpy.asarray(im), h, w
27
28
29 # Unsharp masking techniques
30 def use_mask(original_image, height, width, blurfunc):
31     result = numpy.zeros((height, width), dtype=ARRAY_TYPE)
32     for x in range(0, height):
33         for y in range(0, width):
34             try:
35                 f = max(0, min(COLOR_MAX, original_image[x
36                     ][y] + FILTER_LAMBDA * blurfunc(
37                         original_image, x, y)))
38                 result[x][y] = f
39             except IndexError:
40                 result[x][y] = original_image[x][y]
41     return result
42
43 # Traditional Unsharp Masking
```

```

43 def basic(ori, x, y):
44     blur = (2 * ori[x][y] - ori[x - 1][y] - ori[x + 1][y])
45     return blur
46
47
48 def basic_vertical(ori, x, y):
49     blur = (2 * ori[x][y] - ori[x][y - 1] - ori[x][y + 1])
50     return blur
51
52
53 # 2D Traditional Unsharp Masking (plus pattern)
54 def basic2d(ori, x, y):
55     blur = (4 * ori[x][y] - ori[x - 1][y] - ori[x + 1][y] -
56             ori[x][y - 1] - ori[x][y + 1])
57     return blur
58
59 # Unsharp Masking 3x3, equal weights
60 def basic_plus(ori, x, y):
61     blur = 8 * ori[x][y] - ori[x - 1][y] - ori[x + 1][y] -
62           ori[x][y - 1] - ori[x][y + 1] - ori[x - 1][y - 1] -
63           ori[x + 1][y - 1] - ori[x - 1][y + 1] - ori[x + 1][y
64           + 1]
65     return blur
66
67
68 # Unsharp Masking 3x3, more weights on the horizontal and
69     vertical pixels
70
71 def basic_plus2(ori, x, y):
72     blur = 12 * ori[x][y] - 2 * ori[x - 1][y] - 2 * ori[x +
73           1][y] - 2 * ori[x][y - 1] - 2 * ori[x][y + 1] - ori
74           [x - 1][y - 1] - ori[x + 1][y - 1] - ori[x - 1][y +
75           1] - ori[x + 1][y + 1]
76     return blur
77
78
79 # Unsharp Masking 5x5, equal weights
80 def basic_plus3(ori, x, y):
81     blur = -1*ori[x - 2][y - 2] - ori[x - 2][y - 1] - ori[x
82           - 2][y] - ori[x - 2][y + 1] - ori[x - 2][y + 2] \
83           - ori[x - 1][y - 2] - ori[x - 1][y - 1] - ori[x
84           - 1][y] - ori[x - 1][y + 1] - ori[x - 1][y +
85           2]\
86           - ori[x][y - 2] - ori[x][y - 1] + 24 * ori[x][y]
87           - ori[x][y + 1] - ori[x][y + 2]\
88           - ori[x + 1][y - 2] - ori[x + 1][y - 1] - ori[x
89           + 1][y] - ori[x + 1][y + 1] - ori[x + 1][y +
90           2]\

```

```

77         - ori[x + 2][y - 2] - ori[x + 2][y - 1] - ori[x
          + 2][y] - ori[x + 2][y + 1] - ori[x + 2][y +
          2]
78     return blur
79
80
81     # Unsharp Masking 5x5, more weight closer to the center
82     def basic_plus4(ori, x, y):
83         blur = -1*ori[x - 2][y - 2] - ori[x - 2][y - 1] - ori[x
          - 2][y] - ori[x - 2][y + 1] - ori[x - 2][y + 2] \
84             - ori[x - 1][y - 2] - 2 * ori[x - 1][y - 1] - 2
          * ori[x - 1][y] - 2 * ori[x - 1][y + 1] - ori
          [x - 1][y + 2] \
85             - ori[x][y - 2] - 2 * ori[x][y - 1] + 32 * ori[x
          ][y] - 2 * ori[x][y + 1] - ori[x][y + 2] \
86             - ori[x + 1][y - 2] - 2 * ori[x + 1][y - 1] - 2
          * ori[x + 1][y] - 2 * ori[x + 1][y + 1] - ori
          [x + 1][y + 2] \
87             - ori[x + 2][y - 2] - ori[x + 2][y - 1] - ori[x
          + 2][y] - ori[x + 2][y + 1] - ori[x + 2][y +
          2]
88     return blur
89
90
91     # Cubic Unsharp Masking
92     def cubic(ori, x, y):
93         cub = ori[x - 1][y] - ori[x + 1][y]
94         blur = cub*cub*(2 * ori[x][y] - ori[x - 1][y] - ori[x +
          1][y])
95     return blur
96
97
98     def cubic_vert(ori, x, y):
99         cub = ori[x][y - 1] - ori[x][y + 1]
100        blur = cub*cub*(2 * ori[x][y] - ori[x][y - 1] - ori[x][
          y + 1])
101    return blur
102
103
104    # Cubic Unsharp Masking, first term power=4
105    def cubic_plus(ori, x, y):
106        cub = ori[x - 1][y] - ori[x + 1][y]
107        blur = cub*cub*cub*cub*(2 * ori[x][y] - ori[x - 1][y] -
          ori[x + 1][y])
108    return blur
109
110
111    # Cubic Unsharp Masking, second term uses 4 pixels in a
        single direction

```

```

112 def cubic_plus2(ori, x, y):
113     cub = ori[x - 1][y] - ori[x + 1][y] + ori[x - 2][y] -
        ori[x + 2][y]
114     blur = cub*cub*(4 * ori[x][y] - ori[x - 1][y] - ori[x +
        1][y] - ori[x - 2][y] - ori[x + 2][y])
115     return blur
116
117
118 def cubic_plus2_vert(ori, x, y):
119     cub = ori[x][y - 1] - ori[x][y + 1] + ori[x][y - 2] -
        ori[x][y + 2]
120     blur = cub*cub*(4 * ori[x][y] - ori[x][y - 1] - ori[x][
        y + 1] - ori[x][y - 2] - ori[x][y + 2])
121     return blur
122
123
124 # 2D Cubic Unsharp Masking
125 def cubic2d(ori, x, y):
126     cub1 = ori[x - 1][y] - ori[x + 1][y]
127     cub2 = ori[x][y - 1] - ori[x][y + 1]
128     cub3 = ori[x - 1][y] + ori[x + 1][y] - ori[x][y - 1] -
        ori[x][y + 1]
129     blur = cub1 * cub1 * (2 * ori[x][y] - ori[x - 1][y] -
        ori[x + 1][y]) + \
130         cub2 * cub2 * (2 * ori[x][y] - ori[x][y - 2] -
        ori[x][y + 1]) + \
131         cub3 * cub3 * (4 * ori[x][y] - ori[x - 1][y] -
        ori[x + 1][y] - ori[x][y - 1] - ori[x][y + 1])
132     return blur
133
134
135 # Rational Unsharp Masking
136 # Traditional
137 def rational_mask(ori, x, y, um_func):
138     k = 0.001
139     h = 250
140     global SAVE_AS
141     SAVE_AS = SAVE_AS.replace('[ ]', f'k={k}_h={h}_')
142
143     # g_n = (ori[x][y]*ori[x][y] - ori[x+1][y] * ori[x-1][y]
        ])
144     g_n = (ori[x + 1][y] - ori[x - 1][y])
145     g_n = g_n*g_n
146     rational_control_term = g_n / ((k * g_n*g_n) + h)
147     return rational_control_term * um_func(ori, x, y)
148
149
150 # g(n) with 2 pixels wide
151 def rational_mask_plus(ori, x, y, um_func):

```

```

152     k = 0.001
153     h = 250
154     global SAVE_AS
155     SAVE_AS = SAVE_AS.replace('[ ]', f'k={k}_h={h}_')
156
157     #  $g_n = (ori[x][y]*ori[x][y] - ori[x+1][y] * ori[x-1][y]$ 
158      $g_n = (ori[x + 1][y] - ori[x - 1][y] + ori[x + 2][y] -$ 
159      $ori[x - 2][y])$ 
160      $g_n = g_n * g_n$ 
161     rational_control_term =  $g_n / ((k * g_n * g_n) + h)$ 
162     return rational_control_term * um_func(ori, x, y)
163
164 # 2D  $g(n)$ 
165 def rational_mask_plus2(ori, x, y, um_func):
166     k = 0.001
167     h = 250
168     global SAVE_AS
169     SAVE_AS = SAVE_AS.replace('[ ]', f'k={k}_h={h}_')
170
171     #  $g_n = (ori[x][y]*ori[x][y] - ori[x+1][y] * ori[x-1][y]$ 
172      $g_n = (ori[x + 1][y] - ori[x - 1][y] + ori[x][y + 1] -$ 
173      $ori[x][y - 1])$ 
174      $g_n = g_n * g_n$ 
175     rational_control_term =  $g_n / ((k * g_n * g_n) + h)$ 
176     return rational_control_term * um_func(ori, x, y)
177
178 def rational(ori, x, y):
179     return rational_mask(ori, x, y, basic)
180
181
182 def rational_plus(ori, x, y):
183     return rational_mask_plus(ori, x, y, basic)
184
185
186 def rational_plus2(ori, x, y):
187     return rational_mask_plus2(ori, x, y, basic)
188
189
190 def rational_cubic(ori, x, y):
191     return rational_mask(ori, x, y, cubic)
192
193
194 def rational_cubic_plus2(ori, x, y):
195     return rational_mask(ori, x, y, cubic_plus2)
196

```

```

197
198 # 2D Rational Unsharp Masking
199 # Original: using 2 rational control terms
200 def rational_mask2d(ori, x, y, um_func, um_func_vert):
201     k = 0.0008
202     h = 3000
203     global SAVE_AS
204     SAVE_AS = SAVE_AS.replace('[]', f'k={k}_h={h}_')
205
206     g_nx = (ori[x + 1][y] - ori[x - 1][y])
207     g_nx = g_nx * g_nx
208     g_ny = (ori[x][y + 1] - ori[x][y - 1])
209     g_ny = g_ny * g_ny
210
211     def rational_ct(g_n):
212         return g_n / ((k * g_n * g_n) + h)
213
214     return um_func(ori, x, y) * rational_ct(g_nx) +
           um_func_vert(ori, x, y) * rational_ct(g_ny)
215
216
217 # Using a single rational control term
218 def rational_mask2d_alt(ori, x, y, um_func):
219     k = 0.001
220     h = 250
221     global SAVE_AS
222     SAVE_AS = SAVE_AS.replace('[]', f'k={k}_h={h}_')
223
224     # g_n = (ori[x][y]*ori[x][y] - ori[x+1][y] * ori[x-1][y]
225     #        )
226     g_n = (ori[x + 1][y] - ori[x - 1][y] + ori[x][y + 1] -
           ori[x][y - 1])
227     g_n = g_n*g_n
228     rational_control_term = g_n / ((k * g_n*g_n) + h)
229     return rational_control_term * um_func(ori, x, y)
230
231 def rational2d(ori, x, y):
232     return rational_mask2d(ori, x, y, basic, basic_vertical
           )
233
234
235 def rational2d_cubic(ori, x, y):
236     return rational_mask2d(ori, x, y, cubic, cubic_vert)
237
238
239 def rational2d_cubic_plus2(ori, x, y):
240     return rational_mask2d(ori, x, y, cubic_plus2,
           cubic_plus2_vert)

```

```

241
242
243 def rational2d_plus4(ori, x, y):
244     return rational_mask2d(ori, x, y, basic_plus4,
245                             basic_plus4)
246
247 def rational2d_alt(ori, x, y):
248     return rational_mask2d_alt(ori, x, y, basic2d)
249
250
251 def rational_plus4(ori, x, y):
252     return rational_mask2d(ori, x, y, basic_plus4)
253
254
255 # Main project type functions
256 def unsharp_masking_proj(type, image, height, width):
257     new_image_array = None
258     if type == 'basic':
259         new_image_array = use_mask(image, height, width,
260                                     basic)
261     elif type == 'basic2d':
262         new_image_array = use_mask(image, height, width,
263                                     basic2d)
264     elif type == 'cubic':
265         new_image_array = use_mask(image, height, width,
266                                     cubic)
267     elif type == 'cubic_plus':
268         new_image_array = use_mask(image, height, width,
269                                     cubic_plus)
270     elif type == 'cubic_plus2':
271         new_image_array = use_mask(image, height, width,
272                                     cubic_plus2)
273     elif type == 'cubic2d':
274         new_image_array = use_mask(image, height, width,
275                                     cubic2d)
276     elif type == 'basic_plus':
277         new_image_array = use_mask(image, height, width,
278                                     basic_plus)
279     elif type == 'basic_plus2':
280         new_image_array = use_mask(image, height, width,
281                                     basic_plus2)
282     elif type == 'basic_plus3':
283         new_image_array = use_mask(image, height, width,
284                                     basic_plus3)
285     elif type == 'basic_plus4':
286         new_image_array = use_mask(image, height, width,
287                                     basic_plus4)
288     elif type == 'rational':

```

```

279         new_image_array = use_mask(image, height, width,
                                     rational)
280     elif type == 'rational_plus':
281         new_image_array = use_mask(image, height, width,
                                     rational_plus)
282     elif type == 'rational_plus2':
283         new_image_array = use_mask(image, height, width,
                                     rational_plus2)
284     elif type == 'rational_plus4':
285         new_image_array = use_mask(image, height, width,
                                     rational_plus4)
286     elif type == 'rational_cubic':
287         new_image_array = use_mask(image, height, width,
                                     rational_cubic)
288     elif type == 'rational_cubic_plus2':
289         new_image_array = use_mask(image, height, width,
                                     rational_cubic_plus2)
290     elif type == 'rational2d':
291         new_image_array = use_mask(image, height, width,
                                     rational2d)
292     elif type == 'rational2d_cubic':
293         new_image_array = use_mask(image, height, width,
                                     rational2d_cubic)
294     elif type == 'rational2d_cubic_plus2':
295         new_image_array = use_mask(image, height, width,
                                     rational2d_cubic_plus2)
296     elif type == 'rational2d_plus4':
297         new_image_array = use_mask(image, height, width,
                                     rational2d_plus4)
298     elif type == 'rational2d_alt':
299         new_image_array = use_mask(image, height, width,
                                     rational2d_alt)
300     if new_image_array is not None:
301         return new_image_array
302     raise ValueError
303
304
305 if __name__ == '__main__':
306     start = datetime.datetime.now()
307
308     # Retrieve image and image information from file
309     image, width, height = load_image(f'images/{IMAGE}.png')
310
311     # Apply mask
312     result = unsharp_masking_proj(TYPE, image, height,
                                   width)
313
314     # Convert array back to image and save to file
315     result = Image.fromarray(result)

```



```
316     try:
317         result.save(SAVE_AS.replace('[', ''))
318     except FileNotFoundError:
319         print('ERROR: Directory does not exist')
320         result.show()
321     print('Execution took', (datetime.datetime.now() -
322                             start).seconds, 'seconds')
322     print(f'Saved as {SAVE_AS.replace("[", "_")}')

```