Improved Segmentation in MTObjects Using Adaptive Filtering With Fixed Target Signal-to-Noise Ratio

Sjoerd Bruin (s2563304), Michael H. F. Wilkinson, and Caroline Haigh

Abstract— Software-driven image segmentation is an important task in astronomical measurements, since it is intractable for a human being to perform the required segmentation manually. MTObjects is a successful tool for detecting objects in astronomical observations, but experiences some problems in dealing with noise present in the measurements. These problems are likely caused by the uniform Gaussian smoothing operation not being discerning enough for different levels of noise within the same image.
This research develops an adaptive filtering technique based on the local signal-to-noise ratio in order to alleviate this issue and produce better segmentation results. The experiments performed in this research show that the signal-to-noise ratio-based adaptive filtering technique produces a better segmentation result, both in terms of segmentation quality metrics and in terms of similarity to noiseless equivalents of the test images used. The research also compares against another signal-to-noise ratio-based adaptive filtering technique called Adaptsmooth. The comparison shows that the adaptive filtering technique developed in this research performs on par with or better than Adaptsmooth, while the runtime is vastly superior due to its compatibility with GPU execution. The runtime is also competitive with uniform Gaussian smoothing if a GPU can be used, meaning that it does not significantly deteriorate the overall runtime of the MTObjects framework. Future research could improve the dependence of the filter kernel on the signal-to-noise ratio, for example by introducing an anisotropic filter kernel that changes the shape of the kernel in more sophisticated ways.

Index Terms— Adaptive Filtering, Signal-to-Noise Ratio, Segmentation, MTObjects, Object Detection, Attribute Filtering, Max Tree, Adaptsmooth

1 Introduction

Segmentation is a much-researched area of computer vision. The goal is to recognise objects in images by determining properties that might point to a portion of an image corresponding to an object. A number of prominent segmentation frameworks for astronomical images have been developed. It has become clear from the comparison done by Haigh et al. (2020) that the MTObjects framework produces superior results compared to other popular frameworks: it has the best quality metric results - which are also the most consistent across different test images - and has a competitive runtime. MTObjects is therefore a sensible choice for segmentation tasks in astronomical applications, and it makes sense to perform further research to improve this effective segmentation framework further.

MTObjects has some qualitative problems with its segmentation. The framework easily picks up noisy structures, and can erroneously identify them as actual objects. This results in crinkly extensions to recognised regions (caused by noisy pixels), and also results in gaps in regions where noise caused the framework to find holes in the detected objects. A possible reason for this is that the framework applies a uniform Gaussian smoothing operation on the image, which might simultaneously oversmooth certain parts of the image and undersmooth other parts of the image.

In this research, I will attempt to improve the abovementioned issues with the MTObjects framework by replacing the uniform Gaussian smoothing operation with an adaptive smoothing operation that aims to drive each pixel in the image towards a minimum local signal-to-noise ratio (SNR). By using such a smoothing operation, I believe that I will not oversmooth regions that already are clear enough and do not need extra smoothing, and at the same time not undersmooth regions where a lot of smoothing is appropriate. Consequently, this smoothing behaviour would likely mean that the image is more suitable for segmentation, and because of that an improvement in quality metrics could result. This research will implement this smoothing operations and compare the results with the results for the uniform Gaussian smoothing operation. In addition, I will compare the abovementioned adaptive filtering operation with another adaptive smoothing technique called Adaptsmooth developed by Zibetti (2011), so that I can judge the performance of the SNR-based adaptive filtering technique with the simple uniform Gaussian smoothing operation as well as another adaptive filtering approach that aims to do the same thing as SNR-based adaptive filtering.

In section 2, I will discuss the theoretical underpinnings of MTObjects, of Adaptsmooth, and of the SNR-based adaptive filtering algorithm. In section 3, I will discuss the images used in this research, explain the quality metrics that I will use to judge the performance of the smoothing techniques, and describe the optimisation framework for finding the best parameters for the MTObjects framework when using the new adaptive smoothing operation. In section 4, I will discuss the results of performing the optimisation and evaluating the MTObjects framework on the images used for this research. This section will also discuss the runtime of the different smoothing techniques, and discuss another quality comparison based on the sum squared difference. Section 5 summarises the findings and conclusions in this research, and suggests further research that could improve the SNR-based adaptive filtering technique developed in this research.

The code for this repository can be found in a Github repository for testing purposes. Please use the following link to access the repository: github.com/Bruin96/MTObjects_SNR_based_adaptive_filtering.git

2 Theory

In this section, I will first briefly discuss the MTObjects framework for segmentation that was first introduced by Teeninga et al. (2015a), and discuss the previously established behaviour of this framework. Then, I will discuss signal-to-noise ratio (SNR) based adaptive filtering. After that, I will discuss a multigrid implementation of the adaptive filtering that produces similar results with a signficantly reduced computational cost. Lastly, I will discuss how the multigrid version of the adaptive filter can be accelerated on the GPU, leading to vastly improved performance.

2.1 MTObjects

The MTObjects framework is an attribute-based segmentation framework that uses a maxtree. The maxtree, as used in MTObjects, is described by Teeninga et al. (2016). A maxtree is a tree structure that starts with a lowest image intensity level that forms the base of the maxtree. Then, the maxtree goes through all intensity levels. We add a node to the tree where the intensity is equal to or higher than the current intensity level under consideration. A node contains a connected region, which consists of multiple neighbouring pixels that have an intensity equal to or higher than the current intensity level. We do this for all intensity levels. In order to model nested connected regions at different intensity levels, we use a parent pointer at each node, such that the region associated with the current node is a subset of the parent node and the minimum intensity of the child node is strictly larger

than the minimum intensity of the parent node. At each node, we can store or compute a number of attributes which we can then use for filtering the tree based on one or more attributes. The filtering process is straightforward: at each node, we evaluate a criterion Λ on an attribute associated with that node, and remove the node from the maxtree if it fails to meet the criterion.

Before MTObjects constructs a maxtree, it first computes the background of the image, which denotes the intensity level in regions where no astronomical objects are present. The background estimation is described by Teeninga et al. (2015b). The framework estimates the background by finding flat tiles in the image. In order to find out whether a region is flat, it uses two statistical tests. The first test determines the closeness of the intensity distribution to a normal distribution using the Pearson-d'Agostino K^2-statistic. The second test uses a t-test that determines whether the four quadrants of the tile have the same mean value. If the tile passes both tests, we call it flat. In order to estimate the background, we determine the largest flat tile in the image, and compute its mean and variance, which together represent the background estimate. We then subtract the background mean from the image. In astronomical images, the noise is determined by the measurement of photons, which has the inherent characteristics of Poissonian noise. In the resulting image, the variance of the noise is a function of the original, noiseless image O (typically with a gain factor g applied to it) and the estimated background variance σ_{bg}^2, as given in:

$$\sigma_{image}^2 \propto g^{-1}(O) + \sigma_{bg}^2 \tag{1}$$

After subtracting the background, MTObjects constructs a maxtree and applies attribute filtering as described above. MTObjects defines four different significance tests on a node in the maxtree for determining whether it should be kept as a significant node or not. The application of such a significance test to each node results in a filtered maxtree. This filtered maxtree represents the regions in the image that represent actual objects. The result of the MTObjects framework, evaluated on a simulated Fornax-like galaxy cluster, is shown in figure 1.

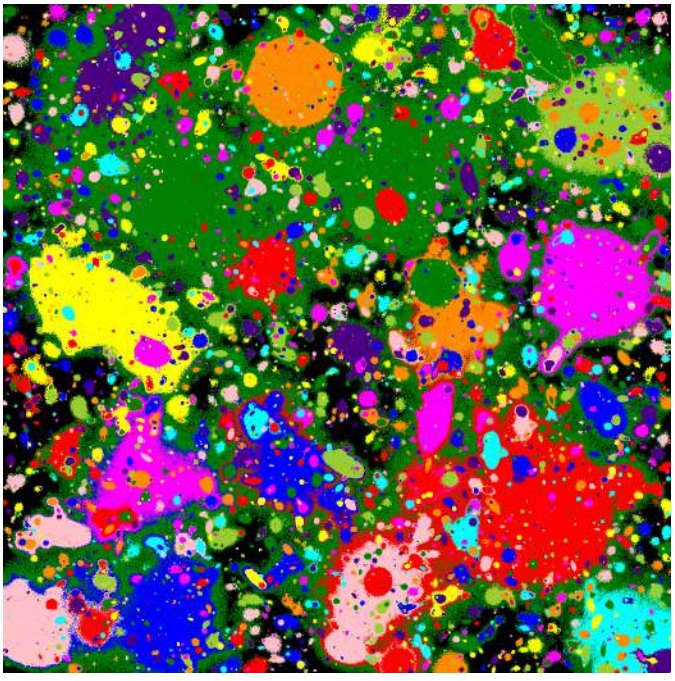


Fig. 1: The result of applying the MTObjects framework with a uniform Gaussian blur filter to a simulated Fornax Deep Survey-like image.

The latest version of MTObjects uses two versions of the input data: the first one is the original, background-subtracted input image, while the second one is a Gaussian-smoothed image. The original input image is used for computing statistics, while the smoothed image is used for finding regions. The use of the original image for statistics means that there are no extra correlations introduced by the smoothing operation, resulting in more informative statistical information. The second image is used for detecting areas of interest in the image, which leads to smoother regions that might be detected as nodes, creating segmentations that are as robust to noise as possible. Specifically, the smoothing operation tries to manipulate the image such that the MTObjects framework can more easily detect regions of approximately constant intensity called flat zones. Due to noise in the image, it is hard to detect flat zones in the unsmoothed image. Smoothing helps to produce these flat zones by smoothing out the variability caused by the noise.

2.2 SNR-Based Adaptive Filtering

For this research, I will consider the latest version of the MTObjects framework that Haigh et al. (2020) uses. This version is similar to the MTObjects framework described in section 2.1, but first applies a Gaussian blur filter to the image. This filter is applied after subtracting the background but before constructing the maxtree. The uniformity of the Gaussian blur filter has a significant potential drawback. This drawback has to do with the fact that in astronomical observations, the measured image will contain inherent Poissonian noise due to the measurement of photons. As a result of this, the measured intensity level has a noise variance that depends on the intensity of the signal. The uniform Gaussian blur filter does not take this into account. This means that the blur operation may be too strong in certain regions of the image, such that it could blur out faint structures, while in other regions, the blur operation may be too weak, such that the noise is not smoothed out. The latter case has as a consequence that a connected region might contain gaps due to noise values that cause the significance tests to incorrectly detect a small, noisy region as a separate object. This same phenomenon can also lead to the detection of tortuous, curve-like extensions to regions due to the detection of noisy values as part of a given region. The result of this effect is illustrated in figure 2.

Adaptive filtering could potentially improve on this situation. The idea behind adaptive filtering is to apply a different degree of smoothing at each point in the image depending on the properties of the image in that location. The formula for computing the Gaussian blur filter is given in:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

We need to apply padding to the borders of the image in order to allow the algorithm to be evaluated on all of the pixels in the image. For this, we use reflection padding, where the padded region mirrors the values of the original image.

The degree of smoothing depends on the Gaussian parameter σ. In adaptive filtering, we compute the value of σ for each position in the image. This allows us to vary the amount of smoothing depending on how far the image is from the target condition in that location in the image. For this research, I am interested in improving the signal-to-noise (SNR) ratio. We compute the SNR ratio as given in:

$$SNR = \frac{\mu_{win}}{\sigma_{poisson}} \tag{3}$$

In order to safeguard against instabilities that might arise in the measurements, we use a window around the image location to determine the mean value μ_{win} at that location. The standard deviation $\sigma_{poisson}$ is subsequently computed as the square root of the mean value μ_{win}: $\sigma_{poisson} = \sqrt{\mu_{win}}$.

In order to improve the SNR ratio, I first choose a target SNR value SNR_{target}. I define two situations: firstly, the SNR value SNR_{curr} at some point in the image might be larger than SNR_{target}. In this case, I do not apply any smoothing. Secondly, SNR_{curr} might be smaller than SNR_{target}. In this case, $\sigma = f(SNR_{target} - SNR_{curr})$, where f is a polynomial function.

Fig. 2: A zoomed-in section of the MTObjects framework output image shown in figure 1. Notice the black pixels inside of the large green areas of the image, indicating pixels that are - likely erroneously - excluded from a region. We can also see tortuous lines moving outwards from the green region in the top left of the image, which likely coincide with trajectories of noisy pixel values that have been erroneously attributed to the region adjacent to them.

The properties of the images in astronomical applications need to be considered when designing the adaptive filter. In particular, the difference in magnitude is quite large, with typical photon counts ranging from on the order of 10^0 to 10^{13}. These large differences can lead to an overly strong blurring effect if I simply use the linear relation $\sigma \propto SNR_{target} - SNR_{curr}$. In particular, very large differences in intensity lead to a very large value for $SNR_{target} - SNR_{curr}$, which in turn leads to oversmoothing in the sense that the resulting SNR value ends up well above SNR_{target}. Instead of using the linear equation, I use the formula in equation 4 in order to compute the smoothing parameter σ:

$$\sigma = 0.1 \cdot \left(SNR_{target} - SNR_{curr}\right)^{0.5} + 0.01 \tag{4}$$

Raising $SNR_{target} - SNR_{curr}$ to the power of 0.5 somewhat attenuates the large differences, in order to make the formula more robust for large differences in SNR values. The strength of the smoothing is further softened by the 0.1 term, in order to make the smoothing approach the target SNR value in a more controlled manner. The additive 0.01 term is there for ensuring numerical stability when implementing the adaptive smoothing operation. Applying the adaptive filter for a single iteration does not guarantee that the image has converged on a final SNR value in that position. Convergence is typically rather slow, so multiple iterations are needed. As a consequence, I need to apply the adaptive filter to the image multiple times in order to achieve a result that is close to a convergent state. In general, the problem of convergence is ill-posed: I do not know beforehand which SNR value the image will converge towards. The reason for this is that the noise variance of the image depends on the mean value of the window, and as such I cannot accurately predict how the SNR value evolves. The solution, therefore, is to apply the adaptive filter for a large number of iterations in order to come as close as possible to convergence.

2.3 Multigrid Adaptive Filter

The number of iterations needed to ensure a result that is close to convergence might lead to a large runtime. In order to reduce this runtime, I use a multigrid approach that is based on the multigrid approach for adaptive smoothing that Saint-Marc et al. (1991) present. The algorithm starts by smoothing the original image for n_1 iterations. Then, the image is decimated to $\frac{1}{4}$ of the size of the original image. At this level, we apply n_2 iterations of smoothing. This process can continue for m levels. Once we hit the lowest level, we traverse back up the recursion in the following way: first, we subtract the decimated image from the smoothed, decimated image at the current level. The result is then interpolated to the size of the level above the current level. Then, we subtract the original, smoothed image from the interpolated image. Lastly, we smooth the result for n_i iterations again. This procedure continues all the way up the multigrid recursion until we return to the original image size. The procedure is shown for three levels in figure 3.

The multigrid algorithm consists of four operations:

- **Adaptive Smooth:** The adaptive filtering operation described in section 2.2.
- **Decimate:** This operation reduces the size of the image by a factor 2 on both axes. The operation is equivalent to simple downsampling where we pick the top-left pixel to represent the four pixels.
- **Interpolate:** This operation increases the size of the image by a factor 2 on both axes. The operation is equivalent to simple upsampling where we duplicate the value in the original position to all of the four pixels in the resulting image.
- **Subtract:** This operation subtracts elements of two images element-wise, and returns the difference image.

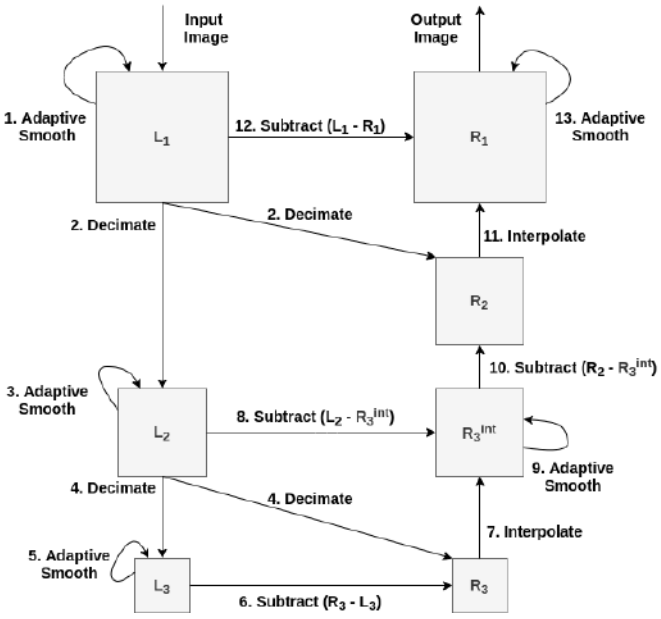


Fig. 3: The multigrid adaptive smoothing algorithm for a 3-level grid. I indicate the order of the operations by the numbers prepended to them. Note that I introduce an extra image (R_3^{int}) so that I can hold the intermediate result as well as the decimated result from the previous level. In an implementation, I can combine these steps into a single operation in order to reduce the memory footprint.

According to Saint-Marc et al. (1991), the multigrid approach to adaptive filtering reaches convergence in on the order of 22 times

fewer iterations at the original image size than the plain adaptive filtering algorithm. Because the size of the grids decreases by a factor 4 for each lower level, the total runtime mostly depends on the number of iterations at the highest level. Indeed, if I assume that I perform n iterations of adaptive filtering at each step with a computational cost of c per iteration, then the computational cost is asymptotically bounded by the identity given in:

$$\lim_{N\to\infty} c\cdot n\cdot \sum_{n=0}^{N} 2\cdot 4^n = 2\frac{2}{3}\cdot c\cdot n \tag{5}$$

Equation 5 tells us that the total computational cost depends mostly on the $2\cdot n$ iterations of adaptive filtering performed at the original image size: adding levels in the grid does not have a large impact on the computational cost. A consequence of this result is that the computational cost needed to reach a similar level of convergence as with the plain adaptive filtering algorithm is approximately a factor $\frac{22}{2\frac{2}{3}} = 8.25$ times smaller. However, this does not take the overhead from the subtract, decimate, and interpolate operations into account. The result that Saint-Marc et al. (1991) found is that they could reduce the execution time from 750 seconds using the plain adaptive filtering approach to 104 seconds using the multigrid adaptive filtering approach.

2.4 GPU Implementation of Multigrid Adaptive Filter

Adaptive filtering changes only one element of the input image at a time, and performs this operation for each element before moving on to the next iteration. Therefore, the adaptive filtering operation is suitable for GPU acceleration. Because the kernel depends on local image properties, the kernel value needs to be recomputed for each image location, which limits the possible acceleration somewhat. I will discuss the speedup in more detail in section 4.3.

In this research, I have accelerated the multigrid implementation using the GPU and the OpenCL GPU programming language. In order to save on GPU global memory, I combine several steps shown in figure 3 into a single operation. For example, the subtraction and interpolation at steps 6 and 7 can be combined into a single operation, and furthermore can be stored in L_2, so that R_3^{int} is no longer necessary. A similar consideration allows for the removal of buffer R_1. The complete GPU implementation workflow for three levels is given in figure 4.

In a GPU implementation, it is important to limit the transfer of data in and out of the GPU memory. For the adaptive filter steps, the buffers remain on the GPU for each iteration: I only need to launch a new kernel with the same buffer arguments. Because I do not want to have the random order of operation influencing the result of the adaptive filtering step, I introduce a second buffer into which the GPU writes its result. At the end of each iteration, I swap the buffers so that the result of the current iteration is available in the loading operation. This operation slightly decreases the speed of the algorithm, but it is necessary in order to maintain the deterministic nature of the algorithm. In the experiments that I have performed for this research, this extra step did not result in a noticeable increase in runtime.

The original image data only needs to be uploaded to the GPU once: memory for the other buffers in the algorithm is allocated directly on the GPU. This is possible because these intermediate results do not need to be returned from the GPU, and therefore they can be allocated and freed completely on the GPU. This is an important observation, because there should be as little memory movement between the GPU and the CPU as possible. The only data that needs to be retrieved from the GPU is the final result at the very end of the algorithm.

2.5 Adaptsmooth

Rather than only comparing with the primitive uniform Gaussian smoothing operation, it makes sense to compare the SNR-based adaptive filtering approach to a comparable adaptive filtering method. I will consider Adaptsmooth as a comparison for SNR-based adaptive filtering. Adaptsmooth was first introduced by Zibetti (2011). It is an adaptive filtering technique that considers windows of increasing radius. At each radius, Adaptsmooth computes the mean filter result

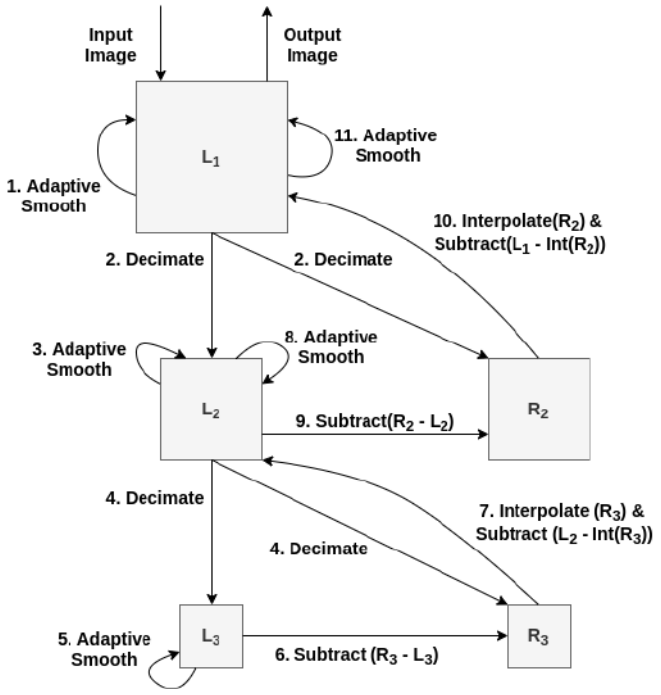


Fig. 4: The multigrid adaptive smoothing algorithm on the GPU for a 3-level grid. Note that I combine one interpolation and subtraction operation without using an intermediate buffer. This saves space on the GPU memory at the cost of a slightly more complicated OpenCL kernel.

of the current window and subsequently computes the SNR value of this window. If the SNR value is above the target SNR value, then the mean filtered result will be assigned to the pixel. Otherwise, the radius is increased, and the same operation is carried out once again. This repeats until the SNR value surpasses the target SNR value, or until the algorithm reaches the (preset) maximum radius. This operation is performed for all pixels in the image.

The main difference between SNR-based adaptive filtering and Adaptsmooth is that SNR-based adaptive filtering tries (and applies) only a single filter which is computed based on the current SNR value, whereas Adaptsmooth tries to fit multiple filters until the current SNR value matches the target SNR value. SNR-based adaptive filtering tries to reach convergence towards the target SNR value by repeated application of the same kernel, while Adaptsmooth tries to do this by a single application of multiple different kernels until one matches the target goal. When taking this difference into account, it would stand to reason that Adaptsmooth has more control over whether the final result actually reaches the target SNR value. However, the repeated verification (and associated branching operations) needed in Adaptsmooth means that it is rather difficult to accelerate the algorithm using the GPU. This means that Adaptsmooth is very unlikely to perform competitively in terms of runtime with SNR-based adaptive filtering when the latter can run on a GPU.

3 Methodology

The aim of this research is to determine whether the MTObjects framework obtains a better segmentation result when it applies an SNR-based adaptive filter instead of a uniform Gaussian blur filter. In order to test this, I replicate the testing framework that Haigh et al. (2020) uses, and I use simulated Fornax Galaxy images as test data. In this section, I will first discuss these images in more detail. After that, I will discuss the quality metrics that Haigh et al. (2020) introduces and that I will use in order to assess the accuracy of the results for the MTObjects framework with SNR-based adaptive filtering. Lastly, I will discuss the Bayesian optimisation framework that Haigh et al.

(2020) uses and that I will use as well in order to find the target SNR value that gives the best results.

3.1 Simulated Fornax Galaxy Images

The images that I use in this research have been provided by Caroline Haigh. The images emulate images of the galaxies present in the Fornax Deep Survey, which is a survey of the Fornax cluster at a distance of 20 Mpc. Each image consists of approximately 1500 simulated stars, 4000 simulated cluster galaxies, and 50 simulated background galaxies. I have been provided with a total of ten such simulated images for this research. The benefit of using a simulated version of the Fornax Deep Survey is that the number of galaxies and other objects is known in advance. This way, I can perform accurate analysis of the results produced by the MTObjects framework.

Each image is accompanied by metadata about the location and size of the objects that are present in that image. However, because of the nature of astronomical observation, there is a certain degree of Poissonian noise present in each image (the noise has been simulated as well). In order to account for this, a ground truth image should take this noise into account. This is done by taking the background level bg plus a fraction of the standard deviation of the noise σ, and computing a threshold t as given in:

$$t = bg + \eta \cdot \sigma \tag{6}$$

Any part of an object that exceeds this threshold intensity will become part of the ground truth image. I will use different values of η in order to test different levels of resolution for the MTObjects framework. A smaller value of η requires that MTObjects detect fainter objects in the image, whereas a larger value of η means that it should detect brighter objects and leave fainter objects undetected.

3.2 Quality Metrics

For measuring the quality of the segmentation produced by a segmentation tool, Haigh et al. (2020) uses four different quality metrics. The first quality metric is the F-score, which is a harmonic mean of the precision and recall of the segmentation. The precision of the segmentation denotes the proportion of objects detected by the segmentation tool that can be matched with an actual object in the ground truth. The recall of the segmentation denotes the proportion of objects in the ground truth that can be matched to objects in the segmentation. For both measures, we find the brightest point in a region as the point that we try to match for the precision and recall calculations. The F-score combines these two measures into a single quality measure as given in:

$$\text{F-score} = 2 \times \frac{\text{precision } \times \text{ recall}}{\text{precision } + \text{ recall}} \tag{7}$$

The second quality measure is the area score. This quality measure consists of two components: firstly, the overmerging (OM) error is a measure of how much the segmentation merges regions that should be separate. Secondly, the undermerging (UM) error is a measure of how much the segmentation denotes regions as separate that are indicated as single, merged regions in the ground truth. The undermerging error is computed using:

$$\text{UM} = \sum_{j=1}^{M} \frac{(A_k - (T_j \cap R_k))(T_j \cap R_k)}{A_k} \tag{8}$$

In equation 8, the ground truth segmentation contains N segments $R_1, ..., R_N$ with areas $A_1, ..., A_N$, and the segmentation tool output contains M segments $T_1, ..., T_M$ with areas $a_1, ..., a_M$. The undermerging error computes the segment R_k in the ground truth segmentation such that for each segment T_j, the intersection $T_j \cap R_k$ is maximised. By doing this for all segments T_j, the undermerging error can be computed. Maximising the intersection $T_j \cap R_k$ corresponds to finding the region in the ground truth segmentation that corresponds closest to the segment T_j currently under consideration. The division by A_k results in a normalisation, such that the sum of the terms $\frac{(A_k - (T_j \cap R_k))(T_j \cap R_k)}{A_k}$ adds up to a number between 0 and 1.

The overmerging error is defined similarly to the undermerging error, as given in:

$$OM = \sum_{k=1}^{N} \frac{(a_j - (T_j \cap R_k))(T_j \cap R_k)}{a_j} \tag{9}$$

For the overmerging error, I find the segmentation tool segment T_j for each R_k such that the intersection $T_j \cap R_k$ is maximised. The overmerging and undermerging error are used to compute the area score, which is given in:

$$\text{Area score } = 1 - \sqrt{\text{OM}^2 + \text{UM}^2} \tag{10}$$

In addition to the F-score and area score, Haigh et al. (2020) defines two combined scores A and B, which are given in:

$$A = \sqrt{\text{Area score}^2 + \text{F-score}^2} \tag{11}$$

$$B = \sqrt[3]{(1 - \text{OM}) \times (1 - \text{UM}) \times \text{F-score}} \tag{12}$$

Both combined scores require the segmentation tool to find a balance between a good area score and a good detection score (F-score). In this research, I will use combined score B as the quality measure with which to judge the performance of the MTObjects framework with SNR-based adaptive filtering compared with the MTObjects framework with a uniform Gaussian blur filter.

3.3 Bayesian Optimisation Framework

In order to find the best combination of parameters for the set of 10 images, I will use a Bayesian Optimisation framework. This framework has been provided by Caroline Haigh, and uses the GPyOpt framework as the backbone of the optimisation process.authors (2016) The framework has two primary modes of operation: parameter optimisation on a single image, or cross-validation parameter optimisation on an entire set of images. The former mode finds the best parameter combination for an individual image, but that combination of parameters might not be a good choice for the other images in the set. The latter mode tries to find the best set of parameter for all images, but might not have the best result for each individual image.

I have performed the optimisation both on a per-image basis and cross-validated on the set of images, in order to compare the performance and the stability of the choice of parameters. An important question to answer in this regard is whether the performance depends significantly on the precise choice of the SNR value, or whether we can accept some margin around the optimal value without significantly impacting the quality measure outcomes. This can be assessed by looking at the performance of the cross-validated results and the per-image results: if both results are comparable, then this indicates a certain level of stability with regards to the choice of parameters. This would be an encouraging result, since it means that setting parameters in future applications will have fewer tuning issues associated with it.

The optimisation framework optimises for the combined B score discussed in section 3.2 for both modes of parameter optimisation. This means that the optimisation will search for parameters that maximise the combined B score either for an individual image or over a set of images. I believe the combined B score represents a well-weighted compromise between good area recognition performance (due to the inclusion of the over- and undermerging errors) and the object detection performance (due to the inclusion of the F-score), and is therefore a good metric for optimisation.

4 Results & Discussion

In this section, I set out a number of points of comparison between the different smoothing methods. I look at uniform Gaussian smoothing, SNR-based adaptive filtering, and Adaptsmooth. In section 4.1, I look at the segmentation produced by MTObjects for each of the smoothing approaches, and visually analyse the properties of the different

smoothing approaches. In section 4.2, I use the summed squared difference measure in order to find out how faithful the smoothed results are to the original image. In section 4.3, I look at the execution time for the SNR-based adaptive filtering algorithm, and contextualise it by comparing it to uniform Gaussian smoothing and Adaptsmooth. Finally, in section 4.4, I look at the results of the parameter optimisation with respect to the quality metrics laid out in section 3.2.

4.1 Segmentation Area Results

In order to understand the visual output of the different segmentations, I first discuss the segmentation map of image 2 in the simulated image set that MTObjects produces for different smoothing approaches. For SNR-based adaptive filtering, I will use the best parameter settings for image 2, while for Adaptsmooth, I will use $SNR = 30.0$, which has turned out to be a strong result for Adaptsmooth. I will do this for different ground truths, and compare with those in order to judge the correspondence between the ground truths and the smoothed segmentation results. The results are shown in figure 5.

Firstly, I look at the results for $\eta = 1.0$. For this ground truth, all methods manage to reproduce the major structures of the ground truths. However, the SNR-based adaptive filtering approach is very clearly the best result: it interprets much less empty space as part of one structure or another, and the shapes closely resemble the shapes in the ground truths than for the other methods. It will become clear in section 4.4 that the abovementioned conclusions about the quality of SNR-based adaptive filtering translates over into better results for the quality metrics defined in section 3.2.

For the ground truth with $\eta = 0.5$, the results for SNR-based filtering and Adaptsmooth are quite close. Adaptsmooth appears to incorporate more of the empty space into some structure, but it also has fewer single-pixel gaps compared to SNR-based adaptive filtering. Both methods mostly manage to reconstruct the structures of the ground truth, but we see some clear deviation even for the larger structures. The Gaussian smoothing approach assigns the most empty space to some structure, has the largest single-pixel gaps in regions, and has the longest erroneous noise structures at the edges of the recognised segments. The results in section 4.4 will show that SNR-based adaptive filtering and Adaptsmooth are very close for this ground truth image.

The ground truth with $\eta = 0.1$ is the most difficult one to approximate for a segmentation framework, because structures need to be recognised deep into the noise present in the image. As a result, all images deviate significantly from the ground truth image. However, I can recognise that several structures from the ground truth are present in the segmentations. For this ground truth, the Gaussian smoothing approach appears to be more competitive with the other approaches than for the other ground truths discussed previously. This time, it appears that it is Adaptsmooth that has a hard time approximating the ground truth. Its biggest issue seems to be that it marks too many pixels as empty space whereas the ground truth shows them as being part of an actual structure. The result of SNR-based adaptive filtering visually appears very close to the result from Gaussian smooth. It has small differences in that it appears to manage to detect faint stars a little bit better. As a result, I would expect that the SNR-based adaptive filtering result is capable of detection more slightly more objects, which in turn would also possibly decrease the overmerging error since fewer objects are erroneously assigned to the more expansive region lying underneath the object. We will see in section 4.4 that something like this is indeed reflected in the quality metrics.

4.2 Adaptive Filtering Sum Squared Difference

An important goal for filtering is to be as similar to the original image as possible. This property is not in general preserved by a blur filter: the uniform Gaussian filter, for example, has the property of smoothing away noise, but it also impacts the non-noisy parts of the image, and might decrease the accuracy of those parts. This might be a situation where an adaptive filter could perform well: it blurs more in noisy regions, and blurs less in noiseless regions. This should result in a blurred image that is closer to the original image than I would get if I applied a uniform Gaussian filter. However, this does not necessarily mean that the blurred image is closer to the original image than the noisy image, either: the blurring operation could still distort the image and produce a less accurate result.

I still need to define accuracy. Because I want to have the image to be as close to the original, I use the sum squared difference over all pixels, which I define in:

$$SSD(I_1, I_2) = \sum_{x,y \in I_1} \big(I_1(x,y) - I_2(x,y)\big)^2 \tag{13}$$

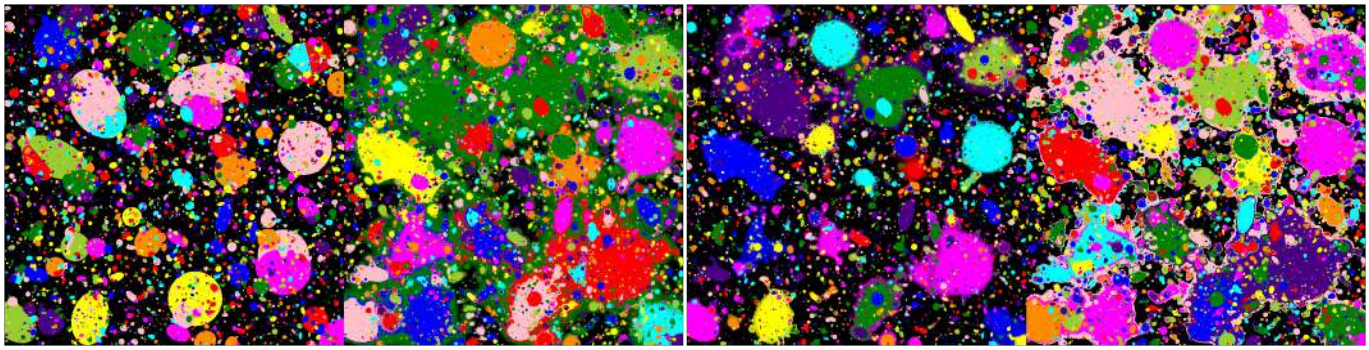
Using equation 13, I define the closest match between two images as the image I_1 for which $SSD(I_1, I_2)$ is the smallest for some reference image I_2. In this case, I use the noiseless image as I_2, and the different blurred images are represented by I_1.

I apply the sum squared difference to several Gaussian blurred images and adaptive filter blurred images for different values of SNR_{target} to the simulated images described in section 3.1. I have two input images: one image is the noiseless version of the image, the other is the same image with Poissonian noise added. The results of computing the sum squared difference for the different resulting images are given in table 1, and the same information is shown graphically in figure 6.

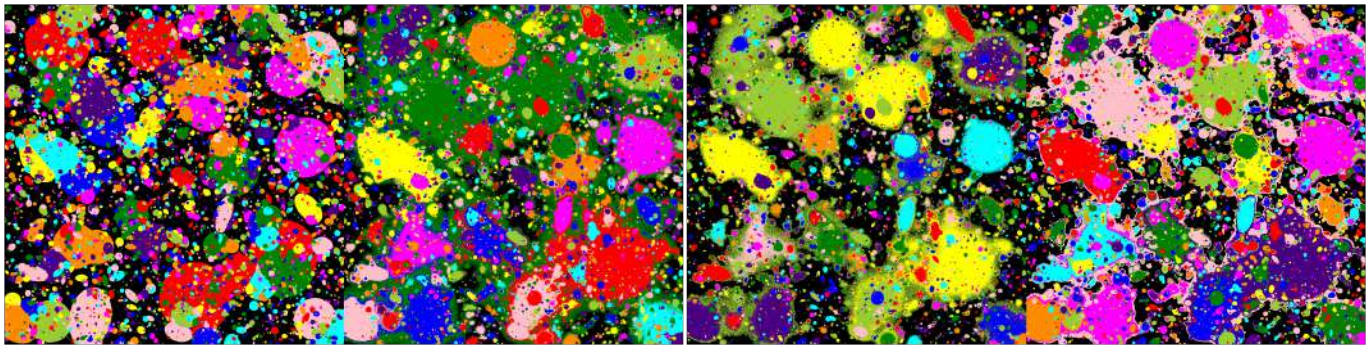
Method	Image 1	Image 2	Image 3
No smoothing	$2.588 \cdot 10^{-16}$	$2.220 \cdot 10^{-16}$	$2.214 \cdot 10^{-16}$
GF	$1.272 \cdot 10^{-12}$	$1.461 \cdot 10^{-12}$	$7.556 \cdot 10^{-13}$
AF $SNR = 2.0$	$2.494 \cdot 10^{-16}$	$2.023 \cdot 10^{-16}$	$1.956 \cdot 10^{-16}$
AF $SNR = 3.0$	$2.494 \cdot 10^{-16}$	$2.023 \cdot 10^{-16}$	$1.956 \cdot 10^{-16}$
AF $SNR = 4.0$	$2.494 \cdot 10^{-16}$	$2.022 \cdot 10^{-16}$	$1.955 \cdot 10^{-16}$
AF $SNR = 5.0$	$2.492 \cdot 10^{-16}$	$2.020 \cdot 10^{-16}$	$1.952 \cdot 10^{-16}$
AF $SNR = 8.0$	$2.458 \cdot 10^{-16}$	$1.978 \cdot 10^{-16}$	$1.901 \cdot 10^{-16}$
AF $SNR = 10.0$	$2.353 \cdot 10^{-16}$	$1.868 \cdot 10^{-16}$	$1.775 \cdot 10^{-16}$
AF $SNR = 15.0$	$1.891 \cdot 10^{-16}$	$1.414 \cdot 10^{-16}$	$1.332 \cdot 10^{-16}$
AF $SNR = 17.0$	$1.780 \cdot 10^{-16}$	$1.313 \cdot 10^{-16}$	$1.248 \cdot 10^{-16}$
AF $SNR = 18.0$	$1.751 \cdot 10^{-16}$	$1.290 \cdot 10^{-16}$	$\mathbf{1.234 \cdot 10^{-16}}$
AF $SNR = 19.0$	$\mathbf{1.742 \cdot 10^{-16}}$	$\mathbf{1.286 \cdot 10^{-16}}$	$1.241 \cdot 10^{-16}$
AF $SNR = 20.0$	$1.757 \cdot 10^{-16}$	$1.305 \cdot 10^{-16}$	$1.273 \cdot 10^{-16}$
AF $SNR = 30.0$	$5.745 \cdot 10^{-16}$	$4.850 \cdot 10^{-16}$	$5.255 \cdot 10^{-16}$
AS $SNR = 5.0$	$1.886 \cdot 10^{-16}$	$1.439 \cdot 10^{-16}$	$1.359 \cdot 10^{-16}$
AS $SNR = 10.0$	$1.800 \cdot 10^{-16}$	$1.319 \cdot 10^{-16}$	$1.251 \cdot 10^{-16}$
AS $SNR = 15.0$	$1.742 \cdot 10^{-16}$	$1.271 \cdot 10^{-16}$	$1.207 \cdot 10^{-16}$
AS $SNR = 20.0$	$1.709 \cdot 10^{-16}$	$1.247 \cdot 10^{-16}$	$1.187 \cdot 10^{-16}$
AS $SNR = 25.0$	$1.694 \cdot 10^{-16}$	$1.239 \cdot 10^{-16}$	$\mathbf{1.180 \cdot 10^{-16}}$
AS $SNR = 26.0$	$1.693 \cdot 10^{-16}$	$\mathbf{1.239 \cdot 10^{-16}}$	$1.181 \cdot 10^{-16}$
AS $SNR = 27.0$	$1.693 \cdot 10^{-16}$	$1.239 \cdot 10^{-16}$	$1.181 \cdot 10^{-16}$
AS $SNR = 28.0$	$\mathbf{1.692 \cdot 10^{-16}}$	$1.240 \cdot 10^{-16}$	$1.183 \cdot 10^{-16}$
AS $SNR = 29.0$	$1.693 \cdot 10^{-16}$	$1.241 \cdot 10^{-16}$	$1.184 \cdot 10^{-16}$
AS $SNR = 30.0$	$1.693 \cdot 10^{-16}$	$1.243 \cdot 10^{-16}$	$1.186 \cdot 10^{-16}$
AS $SNR = 35.0$	$1.703 \cdot 10^{-16}$	$1.256 \cdot 10^{-16}$	$1.202 \cdot 10^{-16}$
AS $SNR = 40.0$	$1.722 \cdot 10^{-16}$	$1.277 \cdot 10^{-16}$	$1.226 \cdot 10^{-16}$
AS $SNR = 45.0$	$1.753 \cdot 10^{-16}$	$1.307 \cdot 10^{-16}$	$1.259 \cdot 10^{-16}$
AS $SNR = 50.0$	$1.794 \cdot 10^{-16}$	$1.344 \cdot 10^{-16}$	$1.299 \cdot 10^{-16}$

Table 1: The sum squared difference (SSD) of different smoothing methods. GF means Gaussian Filtering, AF means Adaptive Filtering, and AS means Adaptsmooth. Remarkably, the adaptive filtering and Adaptsmooth approaches produce an image that is closer to the noiseless image than the noisy image is. The Gaussian filter does not produce a similar result, since it has a significantly larger SSD. For adaptive filtering, the minimum happens for $SNR \in [18.0, 19.0]$, while for Adaptsmooth, the minimum happens for $SNR \in [25.0, 28.0]$.

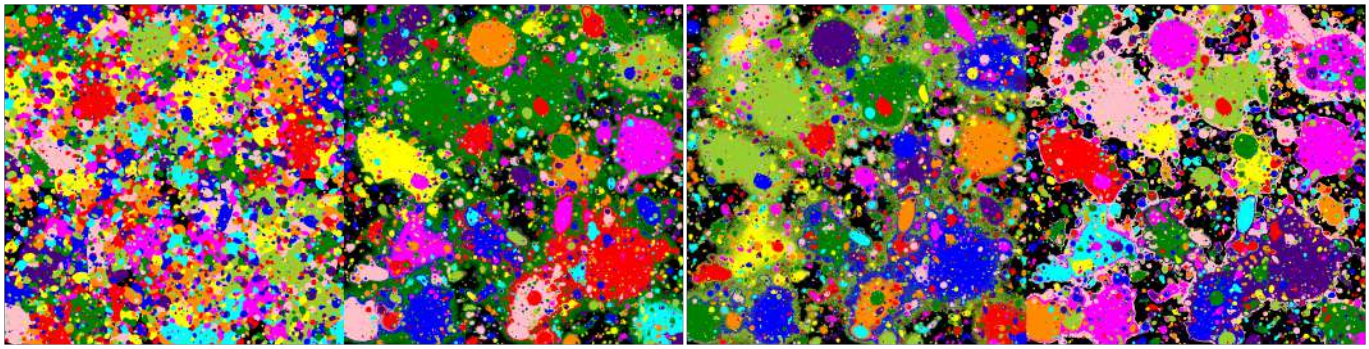
Table 1 and figure 6 show several very interesting results. It appears that the noisy image has a larger SSD value than the adaptively filtered image for an SNR value between 2.0 and 20.0. This result is consistent for all three images that I have tested. This indicates



(a) Ground truth comparison for $n = 1.0$



(b) Ground truth comparison for $n = 0.5$



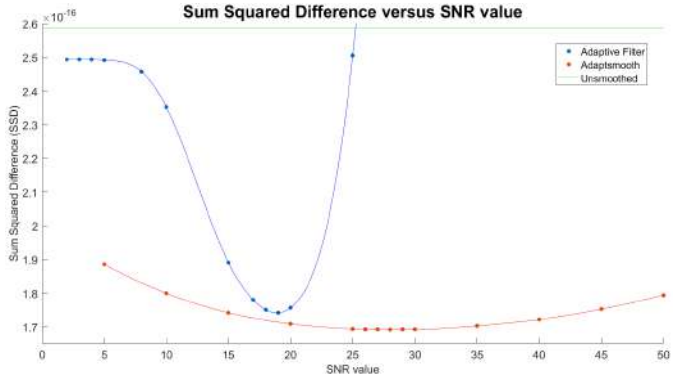
(c) Ground truth comparison for $n = 0.1$

Fig. 5: Segmentation maps for the ground truths and three smoothing approaches. Each row represents a comparison with a ground truth with a value of $\eta \in \{0.1, 0.5, 1.0\}$. Each row contains four segmentation maps, which are, in order: ground truth, Gaussian smoothing, SNR-based adaptive filtering (using optimised parameters for this specific image), Adaptsmooth with $SNR = 30.0$.

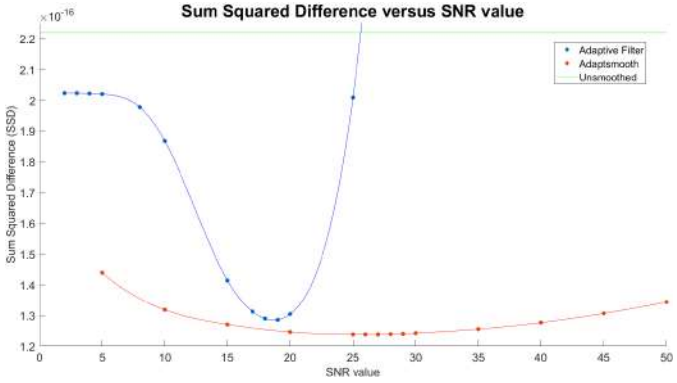
that the adaptive filter approach can remove noise while preserving the original image, in such a way that the correspondence between the adaptively filtered image and the noiseless image is closer than the correspondence between the noisy image and the noiseless image. The best result is achieved for $SNR \in [18.0, 19.0]$, and after that the SNR value increases rapidly. If the target SNR value becomes too large, the SSD value increases, suggesting that we are smoothing too strongly, such that we lose too much of the original image's information. The Gaussian filter has a clearly lower SSD value, which confirms with our intuition that the Gaussian filter smoothes a lot of accurate information away because it is applied uniformly. Adaptsmooth performs slightly better than SNR-based adaptive filtering: it reaches a better SSD value for lower and higher SNR values, and its minimum SSD value is lower than the minimum SSD value for SNR-based adaptive filtering. One possible reason for this could be that Adaptsmooth performs more checks to see if the final result actually achieved the target SNR value, and tries a different filter if it has not. SNR-based adaptive filter, in contrast, only applies a single kernel, the parameters of which are computed based on the statistics of the area around the pixel in question. It is possible that this difference in approach leads to Adaptsmooth producing a lower SSD value.

Another interesting observation is that the quality of the SNR-based adaptive filtering result quickly degrades for SNR values above 20. This implies that it would make sense to build in a limitation on the SNR value in order to avoid erroneous choices that lead to poor results, for example because of a choice of too high an SNR value. However, it is not obvious how this choice of a limit would vary for different images. Further research could look into implementing the limitation on the SNR value I laid out above in such a way that it produces strong results for different images.

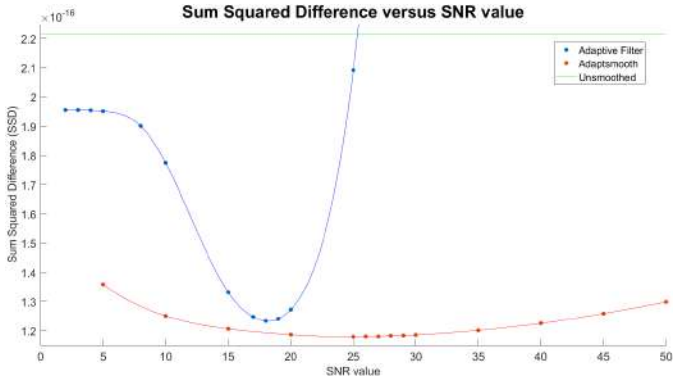
The above results have implications for SNR-based adaptive filtering as a general smoothing tool. Its ability to decrease the loss of information from the noise makes it a very useful smoothing approach, even outside the scope of the MTObjects framework. Its drawback is an increased computation time, but as discussed in section 4.3, the overhead from adaptive filtering might be mitigated by the use of the GPU so that the resulting runtime is tractable for the problem at hand. This depends on the problem for which it will be used, of course. For



(a) Image 1



(b) Image 2



(c) Image 3

Fig. 6: The sum squared difference values plotted against the SNR value for three images in the simulated image set. Around $SNR = 18.0$, SNR-based adaptive filtering performs close to Adaptsmooth, but Adaptsmooth is mostly better in terms of SSD value. Both methods improve the image quality compared to the unsmoothed image.

Adaptsmooth, the same considerations apply, but the impact on the runtime remains large, unfortunately. As such, Adaptsmooth is more useful when the runtime is not at all important and the faithfulness of the output to the noiseless equivalent of the input image is the only thing that matters. Otherwise, SNR-based adaptive filtering likely provides a better compromise between runtime and quality.

4.3 Execution Time

Because of the uncertain convergence properties of adaptive filtering that have been set out in section 2.2, we need to execute the adaptive filtering algorithm for multiple iterations. A consequence of this is that applying the adaptive filtering algorithm to an image takes more time than applying a uniform Gaussian blur filter. Therefore, it is important to determine the execution time of the adaptive filtering algorithm for a number of different settings. For this research, I use the Zeus compute server available at the Rijksuniversiteit Groningen. The Zeus compute server has 64 CPU cores, which allows me to analyse the parallelisability of the adaptive filtering algorithm. Both the plain and multigrid adaptive filtering algorithm have been accelerated with multiple CPU cores using the OpenMP library. The GPU implementation of the multigrid algorithm uses the OpenCL library.

In this section, I first look at the plain adaptive filtering algorithm, and discuss how the algorithm scales on multiple CPU cores. After that, I look at the multigrid adaptive filtering algorithm, compare its runtime with the plain algorithm, and discuss how the multigrid algorithm scales on multiple CPU cores. Then, I look at the GPU implementation of the multigrid algorithm. I use an RTX 2070 mobile GPU for the GPU execution. Finally, I look at the runtime for Adaptsmooth. Gaussian smoothing takes on the order of 2 seconds on a $10,000 \times 10,000$ image, and other smoothing approaches must compare with this runtime.

4.3.1 Plain Adaptive Filtering

I run the plain adaptive filtering algorithm for 50 iterations on the Zeus compute server. Table 2 shows the runtime for a number of CPU core counts.

Number of Cores	Runtime (seconds)	Speedup
1	60771.75	1.0
2	30254.95	2.01
4	15129.96	4.02
8	7563.47	8.03
16	3806.59	15.96
24	2532.21	24.00
32	2061.11	29.48
48	1468.12	41.39
64	1244.01	48.85

Table 2: Runtime for the plain adaptive filtering algorithm for a number of different CPU cores. As the number of CPU cores grows, the acceleration deviates noticeably from a linear speedup, which is in line with the behaviour expected by Amdahl's Law.

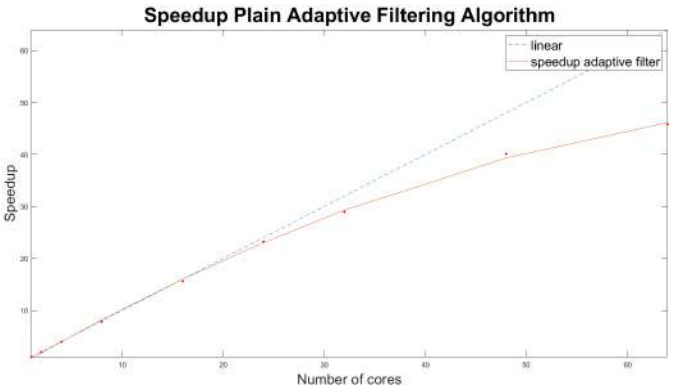


Fig. 7: The speedup for the plain adaptive filtering algorithm for multiple cores plotted against a purely linear speedup. As predicted by Amdahl's Law, the speedup deviates from a linear speedup for a larger number of cores.

From table 2, it becomes clear that the algorithm scales well with an increasing number of cores, with a speedup of 48.85 for 64 cores. In general, Amdahl's Law predicts that the speedup will deviate from a linear speedup for large numbers of cores, and the results in table 2 align with this law. I can use Amdahl's Law to compute the fraction of code that I have to compute sequentially, as given in:

$$f = \frac{p - s}{s \cdot (p - 1)} \tag{14}$$

Here, p is the number of cores, s is the speedup, and f is the fraction of sequential code. Filling in for $p = 64$, $f = \frac{64-48.85}{48.85 \cdot 63} = 4.92 \cdot 10^{-3}$. This value of f has some error, and more detailed statistical experimentation is needed to establish a more confident range of values for f. For the current research, I will use this value, but I maintain the caveat that it does not represent a definitive statement for the speedup.

With knowledge of the fraction of sequential code, I can compute the theoretical limit for the speedup of the adaptive filtering algorithm using:

$$\lim_{p\to\infty} s(p) = \lim_{p\to\infty} \frac{1}{f + \frac{1-f}{p}} \to \frac{1}{f} \tag{15}$$

Equation 15 is completely intuitive: if I have an infinite number of cores, then the part $1-f$ that can be perfectly parallelised will be executed in negligible time, meaning that only the sequential part determines the total time, and the speedup is therefore inversely proportional to the size of the sequential part of the code. Using this result, the theoretical speedup limit for the adaptive filtering algorithm becomes $s(p) \to \frac{1}{4.92 \cdot 10^{-3}} \approx 203$. I will see in section 4.3.3 that the speedup on the GPU is actually greater than what I have found here. There are three explanations for this: firstly, the OpenMP library might influence the theoretical maximum speedup because it produces overhead on the construction of each parallel region, as well as in managing the shared and private memory. Secondly, the Zeus compute server has 64 cores, and its use was not limited to just this experiment. Therefore, other (background) tasks might have interfered with the experiment, leading to suboptimal results for this experiment. Thirdly, the CPU cores in the Zeus compute server are older hardware than the GPU, so it is possible that the per-unit performance of the GPU CUDA cores is actually larger than the performance of the CPU cores in the Zeus compute server. On the GPU, those factors do not occur to the same extent, so the fraction f of sequentially executed code is likely a lot smaller. In future research, it will be interesting to look at the speedup in more detail, using a more isolated environment for the CPU, possibly with more than 64 cores to decrease system overhead, and using a lower-level library like POSIX threads to have more direct control over the parallelisation. These factors might allow us to establish a more accurate value for the fraction of sequential code.

4.3.2 Multigrid Adaptive Filtering

Because the multigrid adaptive filtering algorithm produces comparable results in a significantly smaller number of iterations, I use 5 iterations per adaptive filtering step in the multigrid algorithm. Table 3 shows the execution times and speedups for the CPU implementation of the multigrid adaptive filtering algorithm.

Number of Cores	Runtime (seconds)	Speedup
1	14836.19	1.0
2	7597.49	1.98
4	3767.70	3.94
8	1891.40	7.84
16	952.30	15.58
24	639.03	23.22
32	512.33	28.96
48	369.95	40.10
64	323.99	45.79

Table 3: Runtime for the multigrid adaptive filtering algorithm for a number of different CPU cores. Similar to the plain algorithm, the speedup deviates from linearity as the number of cores grows, as expected by Amdahl's Law.

The number of image pixels in the multigrid algorithm compared to the plain algorithm is a factor $\frac{50}{5 \cdot (2+2 \cdot 0.25+0.0625} = 3.90$ smaller. We see this mirrored in the results when I compare tables 2 and 3: the execution time for 64 cores, for example, is $\frac{1244.01}{323.99} = 3.84$ faster when using the multgrid algorithm. This represents some substantial savings in terms of time, which is important because the overall CPU execution time is quite large in absolute terms.

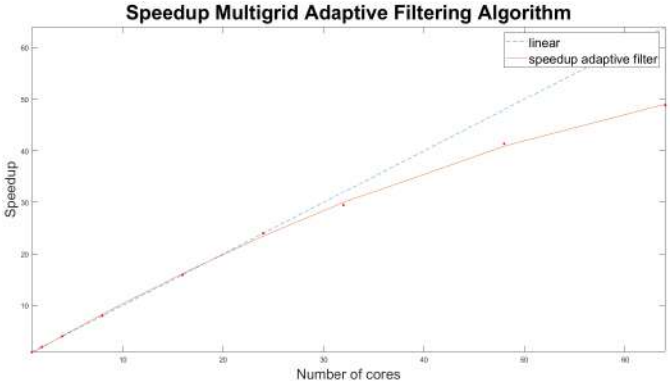


Fig. 8: The speedup of the multigrid adaptive filtering algorithm plotted against a purely linear speedup. The multigrid adaptive filtering algorithm has a similar speedup compared with the plain adaptive filtering algorithm.

4.3.3 Multigrid GPU Implementation

The GPU implementation of the multigrid algorithm has been evaluated on the NVIDIA RTX 2070 mobile GPU. The evaluation of the multigrid algorithm with 5 iterations per adaptive filtering step took 3.89 seconds. This represents a speedup of a factor 3814 compared to the single-core CPU implementation of the multigrid algorithm when run on the Zeus compute server. As mentioned in section 4.3.1, there are a number of reasons why this speedup is larger than the computed maximal speedup, including potentially more powerful per-core performance and less sequential code overhead in the GPU implementation.

The short execution time on the GPU is important in keeping the computation time of the MTObjects framework low. The execution time of the rest of the MTObjects framework is larger than the adaptive filtering execution time on the GPU, meaning that the adaptive filtering procedure does not represent a significant computational bottleneck at the moment. There is currently no GPU implementation of MTObjects available, so the speed advantage that the GPU brings to the adaptive filtering algorithm compared to the rest of the algorithm can be expected to endure for the foreseeable future.

4.3.4 Adaptsmooth

The Adaptsmooth library does not support multiple CPU cores, or GPU execution. As a result, I can only make use of a single core implementation, and must judge its runtime based on that. The runtime of Adaptsmooth varies a lot based on the specific image, as mentioned by Zibetti (2011). For the set of images used in this research, the runtime has been found to be between approximately 6,400 and 8,000 seconds for the mean filter and $SNR = 30.0$ on the Zeus compute server's CPU cores. This runtime is faster than the single CPU core implementation of SNR-based adaptive filtering, but a lot slower than the GPU implementation. Because of the lack of GPU support, SNR-based adaptive filtering can be significantly faster if the relevant hardware is present. This means that the use of SNR-based adaptive filtering is more feasible in a suitable setup with a GPU, especially considering that the runtime of the MTObjects framework is on the order of a few minutes. Adaptsmooth, on the other hand, would dominate the runtime of the MTObjects framework, leading to significantly degraded runtime performance for the entire framework.

It must be noted that the algorithm underlying the Adaptsmooth framework does not appear to throw up significant hurdles that would make a multi-core implementation impossible. As such, it stands to reason that the Adaptsmooth framework could be accelerated by using multiple CPU cores. This could reduce the execution time significantly, depending on how well the algorithm can be parallelised.

Because each CPU core could work on a distinct set of pixels of the image, the parallelisation could potentially be strong, with comparable speed-ups as SNR-based adaptive filtering achieved. That said, GPU implementation would likely suffer from the many conditional branching operations implicit in the definition of the Adaptsmooth algorithm. This means that SNR-based adaptive filtering will likely maintain a significant advantage when GPU acceleration is available.

4.4 Quality Metric Results

The quality metrics laid out in section 3.2 provide a quantitative approach to comparing the different methods. In order to perform this optimisation, I have optimised the parameters on the combined B score quality metric, both on a per-image basis and using cross-validation on the entire set. Before I present those results, I give the quality metric results for Gaussian smoothing and Adaptsmooth.

4.4.1 Gaussian smoothing results

The Gaussian smoothing results form a baseline against which I can compare the performance of Adaptsmooth and SNR-based adaptive filtering. The results are presented in tables 4-6.

Image	UM	OM	Area	F	A	B
1	0.307	0.339	0.542	0.953	0.460	0.758
2	0.369	0.259	0.549	0.959	0.453	0.765
3	0.378	0.263	0.539	0.962	0.463	0.761
4	0.443	0.220	0.506	0.958	0.496	0.747
5	0.351	0.251	0.568	0.960	0.434	0.775
6	0.318	0.301	0.563	0.956	0.440	0.770
7	0.420	0.311	0.478	0.956	0.524	0.726
8	0.347	0.249	0.573	0.957	0.429	0.777
9	0.348	0.236	0.580	0.956	0.423	0.781
10	0.379	0.246	0.548	0.957	0.454	0.765

Table 4: Gaussian smoothing results for $\eta = 1.0$

Image	UM	OM	Area	F	A	B
1	0.236	0.438	0.503	0.953	0.500	0.742
2	0.259	0.363	0.554	0.959	0.448	0.768
3	0.282	0.364	0.540	0.962	0.462	0.760
4	0.322	0.270	0.580	0.958	0.422	0.780
5	0.276	0.372	0.535	0.959	0.465	0.759
6	0.248	0.407	0.524	0.955	0.478	0.753
7	0.310	0.401	0.494	0.956	0.508	0.734
8	0.271	0.363	0.547	0.956	0.455	0.763
9	0.262	0.357	0.557	0.955	0.445	0.768
10	0.274	0.292	0.600	0.957	0.403	0.789

Table 5: Gaussian smoothing results for $\eta = 0.5$

Image	UM	OM	Area	F	A	B
1	0.152	0.490	0.487	0.952	0.515	0.744
2	0.149	0.452	0.524	0.958	0.478	0.764
3	0.154	0.462	0.513	0.961	0.488	0.759
4	0.145	0.445	0.532	0.957	0.470	0.769
5	0.158	0.462	0.511	0.959	0.490	0.757
6	0.151	0.486	0.491	0.955	0.511	0.747
7	0.153	0.468	0.508	0.955	0.494	0.755
8	0.159	0.457	0.516	0.956	0.486	0.759
9	0.160	0.465	0.508	0.955	0.494	0.754
10	0.159	0.458	0.515	0.957	0.487	0.758

Table 6: Gaussian smoothing results for $\eta = 0.1$

The results for Gaussian smoothing appear to be rather consistent across the three ground truth levels. This suggests that the results for Gaussian smoothing can maintain a consistent performance regardless of the depth to which we have to dig into the noisy regions to extract segments - although this statement needs an added qualification that we cannot simply assume this will continue to hold for ground truth levels below $\eta = 0.1$.

While the combine B score stays relatively stable across the different ground truth levels, it becomes clear when studying the over- and undermerging error that the details of the segmentation performance have changed significantly. For $\eta = 1.0$, the undermerging error is rather high, while the overmerging error is rather low. This suggests that on the one hand, the framework has a tendency to identify small regions at this ground truth level which the ground truth marks as larger regions, while on the other hand, it has relatively fewer problems with creating too large regions that should have been broken up into smaller regions. As the value of n goes down, we see the undermerging error decrease, while the overmerging error increases. This tells us that the framework is having more issues with erroneously combining multiple smaller regions in the ground truth into large regions, while it has fewer issues with indicating multiple small regions where a single large region should have been found. This result highlights a property of uniform smoothing: it has a tendency to overmerge small structures, whereas it might undermerge larger structures. This is of course a consequence of applying the same filter everywhere, regardless of how noisy a part of an image may be. I would expect that this type of behaviour does not occur for the adaptive filters, since those do not have to trade off between oversmoothing sharp parts of the image and undersmoothing noisy parts, at least not to the same degree.

4.4.2 Adaptsmooth results

For Adaptsmooth, I use the mean filter with $SNR = 30.0$. The results are given in tables 7-9.

Image	UM	OM	Area	F	A	B
1	0.308	0.253	0.601	0.948	0.402	0.788
2	0.333	0.225	0.599	0.955	0.404	0.790
3	0.336	0.207	0.606	0.956	0.397	0.795
4	0.325	0.198	0.619	0.951	0.384	0.802
5	0.321	0.209	0.617	0.955	0.386	0.800
6	0.313	0.247	0.601	0.951	0.402	0.789
7	0.361	0.233	0.570	0.951	0.432	0.775
8	0.355	0.218	0.583	0.951	0.420	0.783
9	0.366	0.215	0.576	0.952	0.427	0.780
10	0.285	0.211	0.645	0.952	0.358	0.813

Table 7: Adaptsmooth results for $\eta = 1.0$

Image	UM	OM	Area	F	A	B
1	0.157	0.365	0.603	0.948	0.401	0.798
2	0.178	0.319	0.635	0.954	0.368	0.811
3	0.192	0.290	0.652	0.955	0.351	0.818
4	0.163	0.276	0.679	0.951	0.324	0.832
5	0.199	0.295	0.644	0.955	0.358	0.814
6	0.192	0.343	0.607	0.951	0.396	0.796
7	0.210	0.331	0.608	0.951	0.395	0.795
8	0.205	0.302	0.634	0.951	0.369	0.808
9	0.225	0.296	0.628	0.951	0.375	0.804
10	0.167	0.316	0.643	0.952	0.360	0.816

Table 8: Adaptsmooth results for $\eta = 0.5$

Adaptsmooth performs noticeably better than Gaussian smoothing for $\eta = 1.0$ and $\eta = 0.5$. Surprisingly, it seems to perform worse than Gaussian smoothing for $\eta = 0.1$. One possible reason for this is that the choice of $SNR = 30.0$ was made based on studying the results for Adaptsmooth using a manual search on the $\eta = 0.5$ ground truth level comparison. It is conceivable that there is a different set of parameters for Adaptsmooth that might give the best results for different ground truth levels. It would be useful to perform a parameter optimisation

Image	UM	OM	Area	F	A	B
1	0.151	0.484	0.493	0.948	0.510	0.746
2	0.151	0.487	0.490	0.954	0.512	0.746
3	0.153	0.470	0.506	0.961	0.496	0.756
4	0.145	0.522	0.458	0.950	0.544	0.730
5	0.157	0.470	0.505	0.959	0.497	0.754
6	0.151	0.503	0.475	0.950	0.527	0.738
7	0.152	0.510	0.468	0.950	0.535	0.733
8	0.158	0.472	0.502	0.951	0.501	0.750
9	0.159	0.463	0.510	0.951	0.492	0.754
10	0.158	0.473	0.501	0.957	0.501	0.752

Table 9: Adaptsmooth results for $\eta = 0.1$

on the Adaptsmooth framework for this set of images, which could include the SNR value as well as the level cut parameter. Such an optimisation process will likely take weeks of optimisation, however, and is outside the scope of the current research.

Adaptsmooth somewhat shows the result of moving from uniform to adaptive smoothing that I identified in section 4.4.1. While the undermerging error does decrease from $\eta = 1.0$ to $\eta = 0.1$, the drop is less pronounced - mostly because the $\eta = 1.0$ ground truths results in a lower undermerging error than Gaussian smoothing to begin with - and the change mostly happens between $\eta = 1.0$ and $\eta = 0.5$. The overmerging error increases significantly from $\eta = 1.0$ to $\eta = 0.1$, which runs counter to the prediction I made in section 4.4.1. This could be because the choice of parameters happens to not be conducive to good results for $\eta = 0.1$, which has a higher overmerging error than Gaussian smoothing. We will see in section 4.4.3 that the best choice of parameters for $\eta = 0.1$ is quite different from $\eta = 0.5$ and $\eta = 1.0$ for SNR-based adaptive filtering as well.

The F-score results appear to decrease somewhat for Adaptsmooth compared with Gaussian smoothing. The change is not very large, however, and the overall result is still very high. The F-score results are consistent across the different ground truth levels, but it seems to perform slightly better at $\eta = 0.1$. This happens because the recall is slightly better for lower ground truths, indicating that the adaptive smoothing has some benefits for low-level ground truths. Note that we did not see this increase for Gaussian smoothing.

4.4.3 Per-image optimisation results

The parameters that I have found for each individual image are shown in tables 10-12, for each of the three ground truth levels.

Image	move_factor	min_distance	snr
1	0.92568767	0.28261579	1.38643
2	0.65950762	0.0	3.65660402
3	0.81889062	0.38562925	4.98629275
4	0.96431417	1.0	6.13671007
5	1.0	0.57567605	5.23107303
6	0.95714614	0.4504884	3.54010363
7	1.0	1.0	5.32495799
8	0.8650103	0.89752605	3.52065815
9	0.99761604	0.89295837	3.01267856
10	0.74205335	0.30191291	1.37794358

Table 10: Per-image optimised parameters for $\eta = 1.0$

From the results in tables 10-12, I get the following parameter results:

- For $\eta = 1.0$, mean SNR value $\mu_{1.0} = 3.817$ and standard deviation $\sigma_{1.0} = 0.487$
- For $\eta = 0.5$, mean SNR value $\mu_{0.5} = 3.277$ and standard deviation $\sigma_{0.5} = 0.464$
- For $\eta = 0.1$, mean SNR value $\mu_{0.1} = 9.392$ and standard deviation $\sigma_{0.1} = 0.416$

Image	move_factor	min_distance	snr
1	0.34692768	0.82828137	3.97802471
2	0.41209269	0.59014492	4.61129784
3	0.52351734	0.26098801	4.96949735
4	0.39360359	0.20038126	2.92393168
5	0.39433985	0.50332742	1.1097218
6	0.39762053	0.19246389	4.53909965
7	0.52429865	1.0	2.40003547
8	0.5	0.0	5.0
9	0.5763393	0.57847771	2.23622315
10	0.29215489	0.0	1.0

Table 11: Per-image optimised parameters for $\eta = 0.5$

Image	move_factor	min_distance	snr
1	0.0	1.0	6.6528306
2	0.0	1.0	10.283411
3	0.0	0.72322604	8.47069779
4	0.0	0.70734705	11.03602553
5	0.0	0.50344385	9.04118522
6	0.0	0.63323575	9.60683135
7	0.0	0.46569087	10.86473639
8	0.0	1.0	8.94459809
9	0.0	0.0	8.3024839
10	0.0	0.57474463	10.72188578

Table 12: Per-image optimised parameters for $\eta = 0.1$

The difference between the mean values for $\eta = 1.0$ and $\eta = 0.5$ falls within $2 \cdot \sigma$ for both results, and therefore the statement that the two mean values are different is not statistically significant. Performing a pairwise t-test corroborates this: I get $p = 0.4617$ for the pairwise difference between elements, which is not statistically significant. This means that based on the current image samples, SNR-based adaptive filtering does not appear to take on significantly different values for the optimised parameters between $\eta = 1.0$ and $\eta = 0.5$. The mean SNR value for $\eta = 0.1$ is significantly different from the other two other optimised parameter sets, and clearly requires a higher SNR value for a good result. This makes sense, since our aim is to produce the best segmentation for a very low ground truth level for $\eta = 0.1$, and therefore we need to clean up a lot of the noise in order to recognise structures within that noise. Therefore, a higher SNR value makes intuitive sense, because more smoothing takes place in low-SNR parts of the image.

The quality metric results for the per-image optimised parameters are presented in tables 13-15. Figure 9 shows a direct comparison between Gaussian smoothing, Adaptsmooth, and SNR-based adaptive filtering for the combined B scores.

Image	UM	OM	Area	F	A	B
1	0.136	0.293	0.677	0.957	0.326	0.836
2	0.157	0.242	0.711	0.960	0.291	0.850
3	0.113	0.229	0.745	0.965	0.258	0.871
4	0.119	0.208	0.761	0.960	0.243	0.875
5	0.166	0.240	0.708	0.962	0.294	0.848
6	0.147	0.296	0.670	0.958	0.333	0.832
7	0.122	0.268	0.705	0.956	0.298	0.850
8	0.137	0.245	0.719	0.960	0.284	0.855
9	0.129	0.265	0.705	0.959	0.298	0.850
10	0.138	0.235	0.728	0.961	0.275	0.859

Table 13: Per-image optimisation SNR-based adaptive filtering results for $\eta = 1.0$

The results of SNR-based adaptive filtering for $\eta = 1.0$ are strong: the combined B score is roughly 0.05 higher than for Adaptsmooth, and 0.09 higher than for Gaussian smoothing. This represents a signif-

Image	UM	OM	Area	F	A	B
1	0.154	0.385	0.415	0.956	0.417	0.793
2	0.179	0.327	0.373	0.960	0.375	0.809
3	0.138	0.329	0.357	0.964	0.358	0.823
4	0.147	0.305	0.339	0.960	0.341	0.829
5	0.179	0.336	0.380	0.962	0.382	0.807
6	0.181	0.361	0.404	0.957	0.406	0.794
7	0.139	0.365	0.391	0.955	0.393	0.805
8	0.183	0.333	0.380	0.959	0.382	0.805
9	0.177	0.335	0.379	0.959	0.381	0.807
10	0.172	0.326	0.369	0.960	0.371	0.812

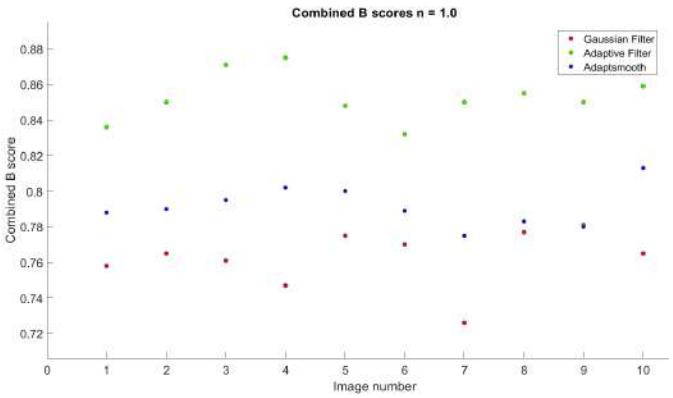
Table 14: Per-image optimisation SNR-based adaptive filtering results for $\eta = 0.5$

Image	UM	OM	Area	F	A	B
1	0.152	0.459	0.484	0.956	0.486	0.760
2	0.150	0.432	0.457	0.960	0.459	0.774
3	0.153	0.437	0.463	0.963	0.465	0.771
4	0.145	0.438	0.462	0.959	0.463	0.772
5	0.158	0.438	0.466	0.961	0.468	0.769
6	0.150	0.458	0.482	0.957	0.484	0.761
7	0.153	0.459	0.484	0.956	0.486	0.760
8	0.159	0.420	0.449	0.959	0.451	0.776
9	0.159	0.426	0.454	0.959	0.456	0.774
10	0.158	0.454	0.480	0.959	0.482	0.761

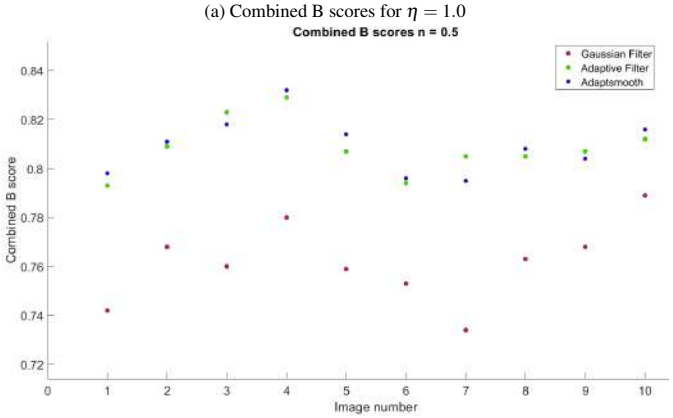
Table 15: Per-image optimisation SNR-based adaptive filtering results for $\eta = 0.1$

icant improvement, and shows that SNR-based adaptive filtering produces large improvements for situations in which we are interested in finding brighter structures and do not care as much about recognising faint structures. The F-score is higher for SNR-based adaptive filtering compared to both Adaptsmooth and Gaussian smoothing. This suggests that SNR-based adaptive filtering is slightly more capable of detecting objects in the image that are present in the ground truth segmentation. The difference in F-score has a rather small absolute change, however, and the most significant change comes from the area measures. In particular, we see that the overmerging error is quite close to the results produced by Gaussian smoothing and Adaptsmooth, but the undermerging error is vastly lower. This makes sense: I mentioned that at high level ground truths, Gaussian smoothing is likely to not blur together the individual signals in the noisy parts of the image, recognising objects that may be due to noise rather than corresponding to the objects in the ground truths. SNR-based adaptive smoothing alleviates this problem by smoothing more strongly in areas of large noise (and therefore low SNR value). It appears that this approach significantly helps in reducing the undermerging of regions (likely due to undersmoothing of the noise in those regions), which leads to areas that correspond more closely to the shapes in the ground truth.

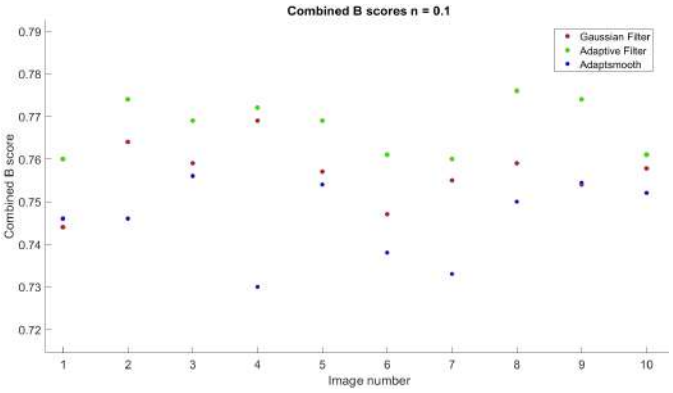
The results at $\eta = 0.5$ are quite interesting. SNR-based produces clearly better results than Gaussian smoothing, which again is mostly due to a better undermerging error. But the results between SNR-based adaptive filtering and Adaptsmooth are quite close. The mean combined B score for SNR-based adaptive filtering is $\mu_{AF} = 0.8084$ and the standard deviation is $\sigma_{AF} = 0.0106$ and for Adaptsmooth, we have $\mu_{AS} = 0.8092$ and $\sigma_{AS} = 0.0110$. These results do not suggest that the two are significantly different, since the two combined B score results are within two standard deviations of one another. A pairwise t-test gives $p = 0.6384$, meaning that the pairwise differences are not statistically significant. This is an interesting result because it suggests that Adaptsmooth can perform comparably to SNR-based adaptive filtering in terms of quality, and it suggests that optimising parameters for Adaptsmooth might be a worthwhile undertaking to see if Adaptsmooth can outperform SNR-based adaptive filtering for an optimised parameter set. This does not solve the performance problem, however, so the practical applicability of Adaptsmooth remains a challenge.



(a) Combined B scores for $\eta = 1.0$



(b) Combined B scores for $\eta = 0.5$



(c) Combined B scores for $\eta = 0.1$

Fig. 9: Combined B score results for $\eta \in \{0.1, 0.5, 1.0\}$

For the low ground truth level of $\eta = 0.1$, the lead over Gaussian smoothing has decreased significantly, but the results for SNR-based adaptive filtering are still better than those for Gaussian smoothing and Adaptsmooth. The decreasing lead can mostly be ascribed to the increase in overmerging error: the undermerging error remains low even at this lower. However, Gaussian smoothing saw the undermerging error drop to a similar level for this ground truth level. A possible explanation for this is that at this low ground truth level, the effect of undersmoothing from uniform Gaussian smoothing have become negligible. The larger amount of oversmoothing could explain why Gaussian smoothing has a larger overmerging error for this ground truth level than SNR-based adaptive filtering.

The results for $\eta = 0.1$ raise some interesting questions about the performance towards the limits of visibility in the simulated images. The current research is limited to looking at ground truths with

$\eta \in \{0.1, 0.5, 1.0\}$, but it is presently unclear how the performance differential between SNR-based adaptive filtering and Gaussian smoothing will develop for $\eta < 0.1$. From the observations that I made about the $\eta = 0.1$ result, I would expect that the overmerging error remains worse for Gaussian smoothing compared with SNR-based adaptive filtering, while the undermerging error probably remains comparable, since undersmoothing does not occur anymore for either smoothing method.

In summary, it appears that SNR-based adaptive filtering can produce very strong results compared with Gaussian smoothing at all tested ground truth levels, and also seems to outperform or match Adaptsmooth. This is a great result, considering it can be competitive in terms of execution time with Gaussian smoothing when using the GPU, while performing significantly better in terms of quality metrics. Also, it is significantly more useful in terms of execution time than Adaptsmooth while producing equal or better results - although parameter optimisation needs to be done in order to determine whether this holds for systematically optimised parameters for Adaptsmooth.

4.4.4 Cross-validated optimisation results

In order to check the stability of the parameters found in section 4.4, I now look at the cross-validation parameters and compare the quality metrics for each ground truth level and each image with the per-image parameter optimisation results. The per-image optimisation should produce results that are better for each image (or at least equally good), but those values are specific to one image, and are not in general applicable to all the other images. Good generalisability of the parameters requires that a single choice of parameters can produce good results for all image in the set (for a specific ground truth level). The cross-validated optimisation for each ground truth level has produced the results presented in table 16.

Parameter	$\eta = 1.0$	$\eta = 0.5$	$\eta = 0.1$
move_factor	0.90962876	0.49350124	0.0
min_distance	0.40943044	0.70785947	0.27691628
snr	2.96249242	4.80072810	9.49636014

Table 16: Cross-validation optimised parameters per ground truth level

The optimal parameters found in cross-validation are not too surprising. Specifically the SNR value is in line with the kinds of values that the per-image optimisation found. The true test, however, is whether the combined B scores are close. The quality metric results are given in tables 17-19.

Image	UM	OM	Area	F	A	B
1	0.137	0.292	0.678	0.957	0.325	0.836
2	0.136	0.264	0.703	0.960	0.300	0.848
3	0.107	0.236	0.741	0.964	0.261	0.870
4	0.128	0.202	0.761	0.960	0.242	0.874
5	0.175	0.235	0.706	0.962	0.296	0.847
6	0.152	0.291	0.671	0.958	0.331	0.832
7	0.130	0.262	0.708	0.955	0.296	0.850
8	0.133	0.249	0.717	0.959	0.286	0.855
9	0.139	0.259	0.706	0.959	0.297	0.849
10	0.122	0.250	0.722	0.961	0.280	0.859

Table 17: Cross-validated optimisation SNR-based adaptive filtering results for $\eta = 1.0$

For the most part, the cross-validated results are in line with the idea that the parameters are stable: most of the combined B scores stick very close to their corresponding per-image optimisation result, with difference mostly staying within 0.003 of the per-image result. Two results stand out negatively: for $\eta = 0.5$, image 3 sees a 0.008 deterioration in the combined B score, and image 10 sees a 0.011 deterioration. It would appear that the $\eta = 0.5$ ground truth has a few more issues with finding a stable set of parameters for the entire image set than the results for the other ground truths. It is not clear to me why this might be the case. It is possible that this is simply a result of the optimisation process, which might need some hyperparameter tuning in order to prevent accidental non-optimal results. However, the difference is not very large, and the results are still adhere to the qualitative description given in section 4.4. Therefore, I will not pursue this line of inquiry further, instead leaving it as a point of attention that might be of interest for further research.

Image	UM	OM	Area	F	A	B
1	0.144	0.397	0.577	0.956	0.425	0.790
2	0.164	0.337	0.625	0.960	0.377	0.810
3	0.167	0.326	0.634	0.964	0.368	0.815
4	0.134	0.321	0.652	0.960	0.350	0.826
5	0.166	0.347	0.615	0.962	0.386	0.806
6	0.173	0.370	0.591	0.957	0.411	0.793
7	0.143	0.363	0.610	0.955	0.392	0.805
8	0.182	0.335	0.619	0.959	0.383	0.805
9	0.177	0.325	0.620	0.959	0.382	0.804
10	0.146	0.374	0.598	0.960	0.404	0.801

Table 18: Cross-validated optimisation SNR-based adaptive filtering results for $\eta = 0.5$

Image	UM	OM	Area	F	A	B
1	0.152	0.464	0.512	0.956	0.490	0.757
2	0.150	0.432	0.543	0.960	0.459	0.774
3	0.153	0.439	0.535	0.963	0.467	0.770
4	0.146	0.447	0.530	0.960	0.472	0.768
5	0.158	0.441	0.532	0.961	0.470	0.768
6	0.150	0.458	0.518	0.957	0.484	0.761
7	0.153	0.461	0.515	0.957	0.487	0.759
8	0.157	0.423	0.548	0.959	0.454	0.775
9	0.159	0.431	0.541	0.958	0.461	0.771
10	0.158	0.457	0.517	0.959	0.485	0.760

Table 19: Cross-validated optimisation SNR-based adaptive filtering results for $\eta = 0.1$

5 Conclusion

In this research I have looked at a form of adaptive filtering that uses the local window signal-to-noise ratio to determine the degree of smoothing applied to a specific position. This approach to smoothing contrasts with uniform Gaussian smoothing in that it takes the local properties of a point in an image into account when smoothing, rather than applying the same filter strength everywhere. By using this SNR-based adaptive filtering approach, I try to prevent both undersmoothing in places where strong smoothing is needed, and oversmoothing in places where little to no smoothing should be applied. I have compared SNR-based adaptive filtering with uniform Gaussian smoothing and with another adaptive filtering approach called Adaptsmooth, which grows the radius of a mean filter kernel until it reaches the target SNR value. I have applied each of these smoothing techniques in the context of the MTObjects segmentation framework, and have compared the results. I have also compared the different smoothing techniques in terms of their ability to decrease the sum squared difference between the noiseless image and the smoothed image (from a noisy variant of the image). Lastly, I have studied the execution time for each smoothing approach.

I have compared the results using a number of different quality metrics. The quality metrics have shown that SNR-based adaptive filtering always holds an advantage over uniform Gaussian smoothing, and often has the advantage over Adaptsmooth as well. My analysis also suggests that, while the lead that SNR-based adaptive filtering has against Gaussian smoothing decreases for lower-level ground truths, it seems likely that the advantage of SNR-based adaptive filtering holds for even lower-level ground truths than the ones I have tested in this research. However, experiments need to be done (which includes gen-

erating those ground truths) in order to verify this prediction. These results show that SNR-based adaptive filtering is a useful addition to the MTObjects framework.

The execution time of SNR-based adaptive filtering is very large for a CPU-only device, but with a GPU, the execution time becomes very small, on the order of several seconds for a $10,000 \times 10,000$ pixel image. This is quite close to the 2-second runtime of Gaussian smoothing, and means that SNR-based adaptive filtering does not incur a significant penalty on the total runtime of the MTObjects framework. Adaptsmooth, on the other hand, does not support GPU execution, and as a result requires on the order of 1 hour to evaluate on a $10,000 \times 10,000$ pixel image. This would mean that it increases the total execution time of a framework like MTObjects by a large factor, and is therefore not a good option for the MTObjects framework - at least if we want to preserve the competitive runtime that MTObjects has.

Our experiments show that the sum squared difference of SNR-based adaptive filtering of a noisy image compared with the noiseless image is significantly lower than that of uniform Gaussian smoothing, and furthermore is lower than the sum squared difference of the unsmoothed, noisy image. This has practical implications for SNR-based adaptive filtering as a general-purpose smoothing tool. Since it decreases the sum squared difference, it improves the image quality, and therefore could prove useful for many applications where increasing the image quality is desired. Adaptsmooth can produce even lower sum squared difference values, but for a suitable choice of SNR value, SNR-based adaptive filtering can get quite close to the optimal result that Adaptsmooth achieves. This competitive accuracy combined with the superior execution time (on a GPU) likely puts SNR-based adaptive filtering ahead as the most practical, accurate method in this comparison.

As I have mentioned a number of times in this paper, I have not performed parameter optimisation on Adaptsmooth due to the extremely long execution time required for such a parameter optimisation. Future research could perform this parameter optimisation in order to check the accuracy of the findings in this research for optimal parameter settings for Adaptsmooth.

SNR-based adaptive filtering uses a simple Gaussian filter, and only varies the standard filter parameter as a response to the local SNR value. Future research could look into more suitable kernels, for example by using an anisotropic filter kernel. Such a kernel would make it possible to fit an even more appropriately-shaped kernel for the local noise characteristics, and it would not harm the GPU-compatibility of the algorithm as long as the shape of the kernel can still be determined without iterated evaluations on the image for each smoothing step.

As mentioned in section 4.2, it might be sensible to implement a limitation on the SNR values depending on the specific image, so that a user does not inadvertently choose too high an SNR value that would degrade the performance of SNR-based adaptive filtering. Future research could look into a way to implement this limitation, and study how this limitation can be applied effectively across a range of different images.

REFERENCES

authors, T. G. (2016). GPyOpt: A Bayesian Optimization framework in python. `http://github.com/SheffieldML/GPyOpt`.

Haigh, C. et al. (2020). Optimising and comparing source-extraction tools using objective segmentation quality criteria. In Astronomy & Astrophysics vol. 645, A107, pp. 1–30,.

Saint-Marc, P., Chen, J.-S. and Medioni, G. (1991). Adaptive Smoothing: A General Tool for Early Vision. In Transactions on Pattern Analysis and Machine Intelligence vol. 13:6, pp. 514–529,.

Teeninga, P., Moschini, U., Trager, S. C. and Wilkinson, M. H. (2015a). Improved detection of faint extended astronomical objects through statistical attribute filtering. In International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing pp. 157–168, Springer.

Teeninga, P., Moschini, U., Trager, S. C. and Wilkinson, M. H. (2015b). Improving background estimation for faint astronomical object detection. In Proceedings - International Conference on Image Processing pp. 1046–1050,.

Teeninga, P., Moschini, U., Trager, S. C. and Wilkinson, M. H. (2016). Statistical attribute filtering to detect faint extended astronomical sources. In Mathematical Morphology - Theory and Applications vol. 1, pp. 100–115, De Gruyter Open.

Zibetti, S. (2011). Introducing ADAPTSMOOTH, a new code for the adaptive smoothing of astronomical images.