Department of Commerce | National Oceanic & Atmospheric Administration | National Weather Service

Space Weather Prediction Center Controlled Document

Editors: Jeff Johnson, Tamara Bledsoe, George Millward, Nathan Miles

Approver: Steven Hill

Revision: 0.2.3.000308

11/08/2023 9:38 AM

SWFO-L1 CCOR ATBD

Table of Contents:

1 Introduction

The purpose of this document is to clearly define the image processing steps to be performed at Level 3 on the coronagraph images ingested at SWPC from the CCOR-1 and CCOR-2 coronagraphs onboard the GOES-U and SWFO-L1 spacecraft respectively.

Software developed by the CCOR vendor Naval Research Laboratory (NRL), and implemented for operations by the SWPC SWFO-L1 dev team, will produce images in the uncompressed FITS image format at levels L0a through L2.

The pipeline steps up to Level2 for the CCOR1 instrument are:

1. L0 CCSDS frames
2. L0b - Unpacked CCSDS frames and images rotated so that Solar North is at the top - FITS format
3. L1a - image as Digital Number (DN), bias and defects removed, divide by exposure time - FITS format

4. L1b - stray and scattered light removed - FITS format
5. L2 - fully calibrated - FITS format

The pipeline steps for CCOR2 are slightly different:

1. L0 CCSDS frames
2. L0b - Unpacked CCSDS frames and images rotated so that Solar North is at the top - FITS format
3. L1a - image as Digital Number (DN), bias and defects removed, divide by exposure time - FITS format
4. L2 - fully calibrated, F-corona subtracted, flat-filed vignetting, distortion correction applied - FITS format

CCOR image files at all levels up to Level 2 have an image pixel size of 2048 (width) by 1920 (height). Files at level 0a and 0b are of size 8MB while those of level 1 and 2 are 16MB. New coronagraph files will nominally be produced every 15 minutes (i.e., cadence = 15 minutes).

It is assumed that CCOR FITS files at Level 2, as processed by algorithms provided by the vendor, will be somewhat 'finished' in the sense of having the most important processes applied to produce the best images of structures within the Solar Corona. Chief among these processes will be 'background subtraction' which is specifically a vendor-defined process for images being processed from Level 1 to Level 2.

Processing at Level3 will entail reducing the files to half size and saving them in a .jpeg format.

2 Algorithm Description

2.1 Reduction of images to half-size and saving in .jpeg format

2.1.1 Algorithm Overview

Images at the full pixel size of 2048 x 1920, whilst best for scientific analysis, are too large for more general use. Level 3 will rebin these images to half-size, 1024 by 960, utilizing a median rebin process.

Downsampling images as part of the L3 processing helps improve computational efficiency because image and movie viewers can ingest the lower-resolution images directly. Furthermore, it provides an opportunity to ensure that the downsampling is optimal and consistent across platforms.

2.1.2 Processing Outline

Median downsampling is straightforward to implement in python using the **scikit-image** image processing package[1]:

```
import numpy as np
from skimage.measure import block_reduce

result = block_reduce(image, block_size = 2, func = np.nanmedian)
```

where image is the 2048 x 1920 input image and result is the 1024 x 960 output image. This example includes a further robustness check by using the nanmedian numpy function to exclude corrupted pixels that may be flagged with an IEEE Not a Number (NaN) value.
At this stage, the images will also be saved as jpeg files, with a "high quality" (=90%) setting.

In keeping with previous generally disseminated coronagraph images, most notably SOHO LASCO C3, the general CCOR coronagraph images will be rendered with a blue-white colormap, referred to as 'Blues_r' in MatplotLib-Python, ie:

```
import matplotlib as mpl
```

[1] https://scikit-image.org/

```
cmap = mpl.colormaps['Blues']
```

and colortable 1 in IDL, ie:

```
Loadct, 1
```

Using this blue-white colormap does not represent anything scientific but is utilized so that the images are instantly recognizable as outer-field coronagraph images, carrying on the tradition started with LASCO C3 images.

The Level3 CCOR images will also feature the NOAA logo (top left), and the image time in the format of YYYYMMDD HH:MM (bottom left). An example image, using an image from LASCO C3, is shown in Figure 1.

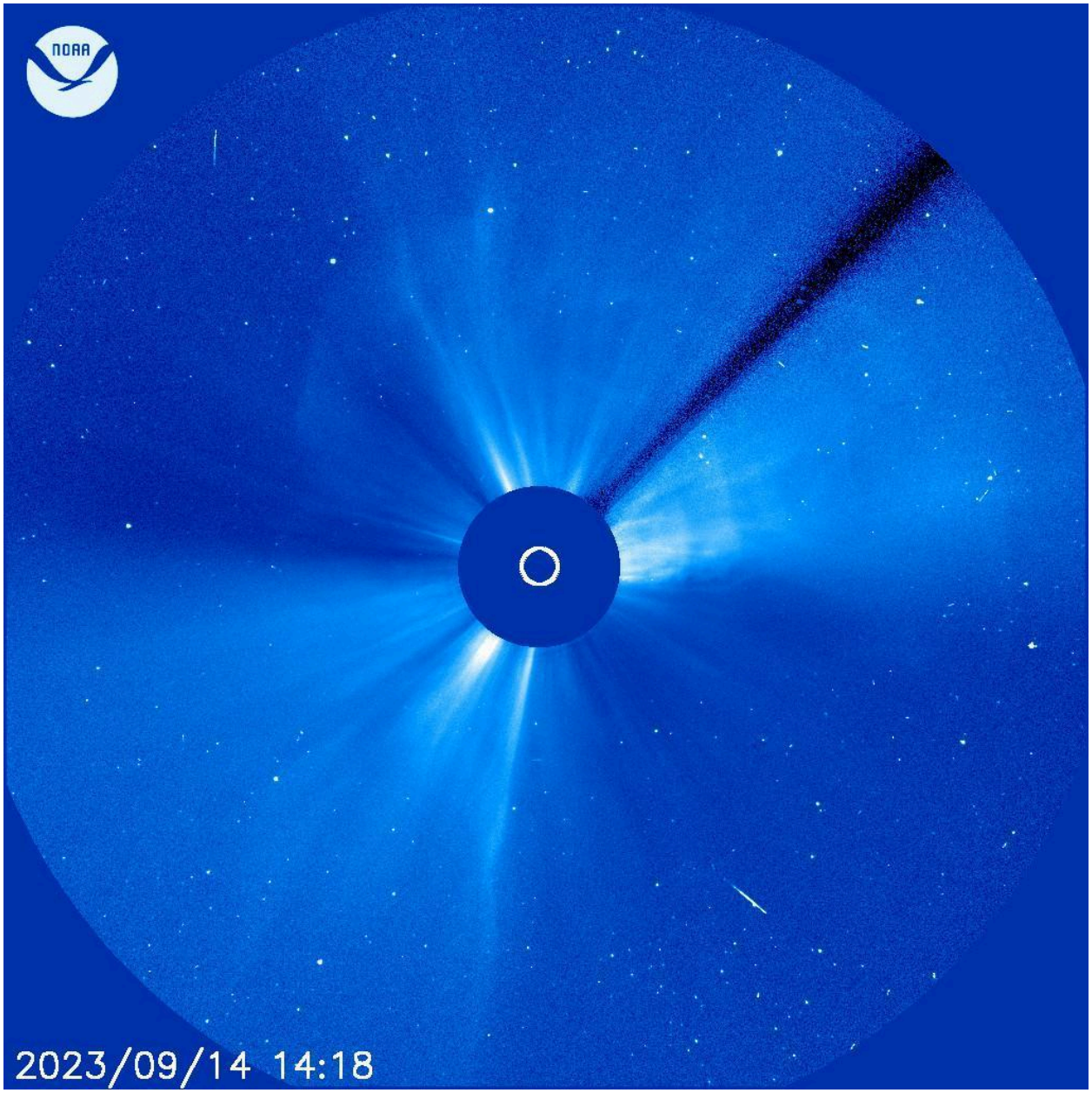

Figure 1: An example of a Level 3 CCOR image, featuring a blue-white colormap, date-time stamp and NOAA logo

In addition to producing Level 3 images as jpeg files, Level 3 will also concatenate still images to produce movies showing evolving coronal structures, most notably CMEs. Movies in .mp4 format and a frame size of 1024 by 960 will be produced using the program ffmpeg:

```
ffmpeg -r $framerate -pattern_type glob -i '*.jpeg' -vcodec libx264 -crf 18 -r
$framerate -y output.mp4
```

Where $framerate is the desired movie frame rate in images per second and the above command will work on a folder of timestamped .jpeg files. The mp4 files to be generated will include:

- last 1 day, 7 days, 27 days from the current time
- last 1 day, 7 days, 27 days on UTC day boundaries

3 Possible Future Work

The CCOR images processed to Level 2 by algorithms provided by the vendor, NRL, are expected to be somewhat 'finished' in terms of best imaging the Solar Corona and representing coronal structures such as CMEs. Thus the only processing envisaged at Level3 will be the image reduction and saving as .jpeg as described above in Section 1.

However, it is anticipated that once live images are being received, some additional image processing may be desirable as a SWPC Level 3 process to further enhance the imagery for general dissemination. Note: these additional processes, documented here in sections 2a, 2b, and 2c, are only ***possible*** additional processes, dependent upon the quality of the images as already produced at Level 2, and may not end up as part of the Level 3 process at all.

Possible anticipated additional processing components are:

Removal of bright points
Feature enhancement
Noise reduction

The L3 processing algorithm used in operations need not include all components. For example, applying a noise filter may not be necessary for operational L3 data, especially if a bright point filter has already been applied. However, noise reduction can optionally be added if it is found that it improves the quality of movie loops produced during post-launch testing for both GOES-U and SWFO-L1. Similarly, feature enhancement and noise reduction may be desirable when producing images and movies for press releases and other special events.

3.1 Removal of bright points with a median bright point filter

A simple, robust, and effective bright point filter can be constructed to remove noise in the form of random bright pixels which are detrimental to the coronal structures in the image. The first step in this algorithm is to define a circular region around each pixel defined by a specified sampling radius, in pixels. For example, if the sampling radius is chosen to be 10 pixels, then the sampling region around pixel (*i, j*) consists of all pixels (*l, m*) for which $\sqrt{(l-i)^2+(m-j)^2} \leq 10$. The next step is to compute the median of all values within the sampling region for each pixel. If the value of pixel (i, j) is larger than the median in the sampling region multiplied by some user-defined threshold, then the pixel value is replaced by the median value. An example processing of a STEREO COR2 image is shown in Fig. 2.

The median filter can be implemented in python by leveraging the scikit-image package for specified values of threshold and radius:

```
import numpy as np
from skimage.filters import median
from skimage.morphology import disk

med = median(image, disk(radius))
mask = image > threshold * med
a = np.where(mask, med, image)
```

This algorithm assumes that the outliers are positive, which is appropriate for energetic particles that typically induce a high charge state in imaging sensors. But, it may not be desirable for difference-images where positive outliers can be transformed into negative outliers through subtraction. If needed, this can be addressed in a straightforward manner by replacing the image in the code segment above by its absolute value.

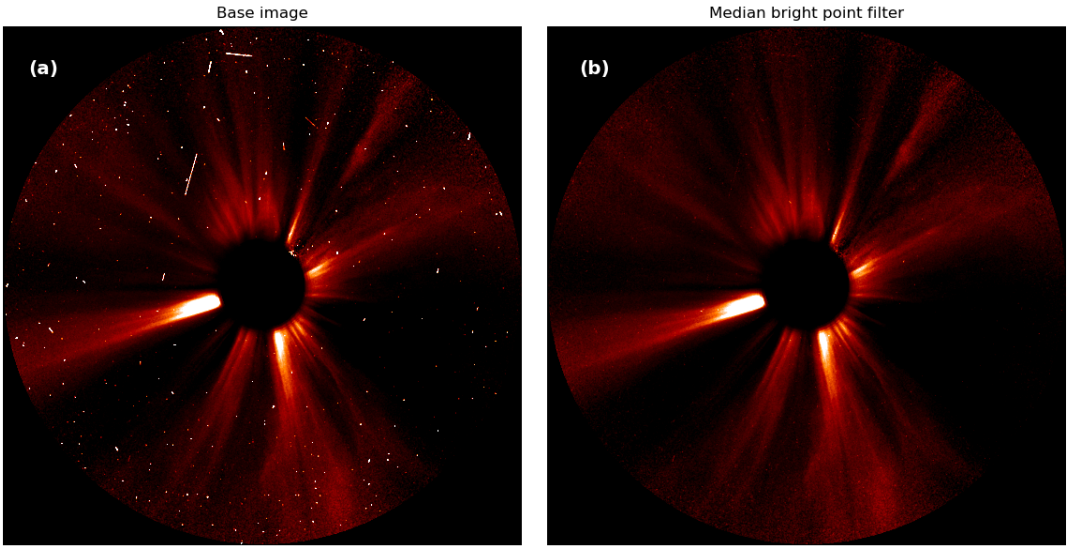

Figure 2. (a) L1 background-subtracted and downsampled image from STEREO-A at 22:54 UTC on Sept 20, 2012. (b) Median bright point filter applied to image from frame (a) with a threshold of 1.1 and a sampling radius of 10 pixels (scale 1 to 1.3).

3.2 Feature Enhancement by multiscale gaussian norm (MGN)

After removing spurious bright points, the next step in L3 image processing is feature enhancement. The simplest type of feature enhancement is what is often referred to as a gamma correction. This uses a nonlinear (power law or logarithmic) color table to equalize brightness between low and high intensity features in the image. In the case of coronagraph images, this generally means making dimmer features more prominent so the image is not dominated by a few small bright features. The SWPC_CAT tool already provides an interactive gamma correction, and the color tables typically used for coronagraph images are already nonlinear.

The Multiscale Gaussian Norm (MGN) technique, developed by Morgan & Druckmüller (2014), can be thought of as an extended version of a gamma correction, with additional enhancement of coherent features on multiple scales. The algorithm begins by defining a symmetric 2D Gaussian kernel G_i with a width w_i (expressed in pixels) in each of the two dimensions. Then features of scale w_i are extracted by convolving the image B with G_i, subtracting this smoothed image from the original image, and applying a local normalization based on the statistics of surrounding pixels:

$$C_i = \frac{B - B \otimes G_i}{\sigma_i} \tag{1}$$

where $\otimes$ denotes a convolution and σ_i^2 is the local variance of the difference image, smoothed over scale w_i by convolving it again with G_i:

$$\sigma_i = \sqrt{[(B - B \otimes G_i)^2] \otimes G_i} \tag{2}$$

The output image is then given by a weighted sum over the different image components:

$$R = h\, C_\gamma + \frac{1-h}{n} \sum_{i=1}^{n} g_i \arctan(k\, C_i) . \tag{3}$$

Here C_γ is just the standard gamma correction:

$$C_\gamma = \left(\frac{B - b_0}{b_1 - b_0}\right)^{1/\gamma} \tag{4}$$

where γ controls the degree of nonlinearity in the color/brightness table and b_0 and b_1 are minimum and maximum clip values that are used to define the range at which the color table saturates.

The MGN image in eq. (5) reduces to a standard gamma correction when h = 1. So, the parameter h is used to control how much emphasis to place on the Gaussian-filtered component relative to the more gamma correction.

The second term in eq. (5) is a sum over Gaussian-filtered images obtained from eq. (3) using Gaussian kernels with n different widths, chosen by the user. Each component is weighed by a user-specified coefficient g_i. The arctan transformation is applied so that the range of pixels is more precisely controlled by the weights g_i, along with the scaling parameter k. Large values of k (> 1) will tend to saturate the color/brightness table for each Gaussian-filtered image, increasing the contrast.

The user-specified parameters for the MGN algorithm are summarized in Table 1. Though optimal results can be obtained by tuning these parameters for particular images, robust default values must be specified for operational use.

Table 1: Parameters for MGN Algorithm

Parameter	Description
w_i	An array of length n specifying the widths of the different Gaussian kernels

g_i	An array of length n specifying the weights for each of the Gaussian-filtered components: eq. (5)
k	Controls the saturation of the color/brightness table of the Gaussian-filtered components through the arctan transformation in eq. (5)
ɣ	The (inverse) exponent to use for the gamma correction: eq. (6)
b_0, b_1	The minimum and maximum clip values that control the scale in the gamma correction: eq. (6)
h	Controls the relative weighing of the gamma correction (dominant for h=1) to the Gaussian-filtered component (dominant for h=0): eq. (5)

A python implementation of the MGN algorithm is freely available through the SunPy package (Barnes et al 2020; https://sunpy.org). Example usage is:

```
from sunkit_image import enhance
result = enhance.mgn(image, h = 0.8, gamma = 1.5, gamma_min = 0.0, gamma_max = 1.0)
```

The sunkit_image package should not be confused with the skimage package we encountered above. The former is a component of SunPy while the latter is the scikit-image python module, a general-purpose image processing package.

The values of h = 0.8 and ɣ = 1.5 in the code segment above were found to give robustly good results for coronagraph images, along with the default SunPy mgn values for the other parameters. This is demonstrated in Fig. 3. The default list of Gaussian widths, w_i (pixels), is set to [1.25, 2.5, 5, 10, 20, 40] with the corresponding weights g_i set to unity. The default value for k is 0.7. The clip values b_0 and b_1 default to the minimum and maximum values of the input image but here they have been set to 0 and 1, appropriate for normalized images. Specifying these parameters through the gamma_min and gamma_max arguments to MGN helps to ensure consistent scaling across movie frames,

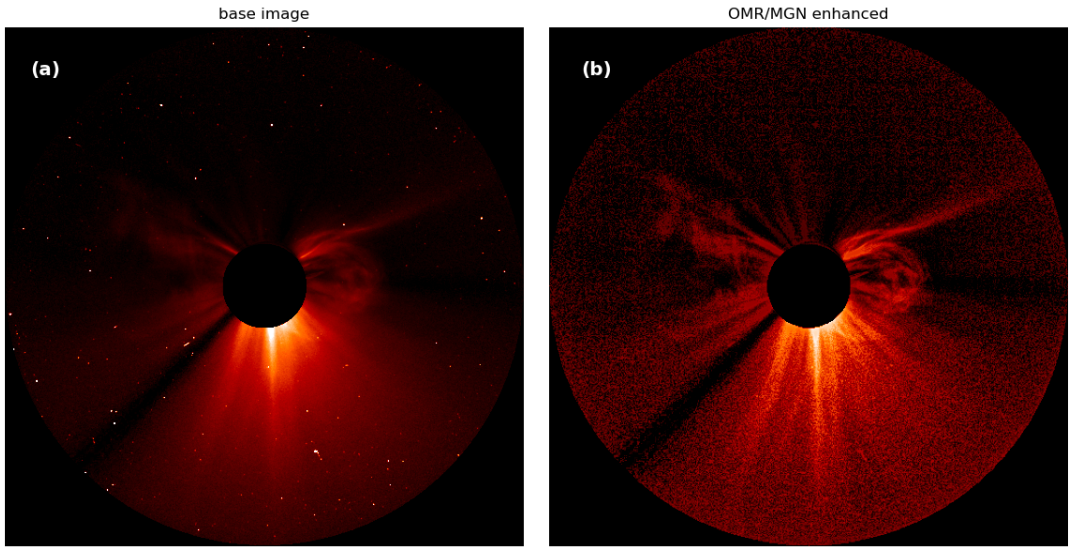

Figure 3: (a) LASCO/C3 L0.5 data (background-subtracted and normalized with exposure time) from 4:06 UTC on April 4, 2012. (b) Image (a) after processing with median downsampling, a median bright-point filter, and MGN feature enhancement.

3.3. Noise Removal

Feature enhancement, as discussed above, brings out detail in the image, but can also amplify noise. This is particularly the case near the outer edge of the field of view, where the signal to noise ratio is relatively low. An example is shown in Fig. 12 using L0.5 data from the LASCO C3 coronagraph. Detector noise is most apparent in the lower portion of the processed image (12b) as a mottling of the brightness pattern.

The noise in LASCO/C3 is more prominent than in STEREO/COR2 in part because of the wider field of view, extending out to $30R_\odot$ compared with 15 $R_\odot$ for the latter. As mentioned in Section 3.1.1, the brightness of coronal features decreases rapidly with distance from the Sun, making them more challenging to detect at larger radii. When algorithms such as MGN or FNRGF enhance the signal in these dim regions, they also enhance the noise. The FOV for CCOR-1 (17 $R_\odot$) and CCOR-2 (22 $R_\odot$) lie in between those for STEREO/COR2 and LASCO/C3 so it is expected that feature enhancement for these instruments will have a similar effect.Noise removal is a broad area of image processing, encompassing a wide range of challenges and approaches and producing a rich volume of literature. In section 2 we suggested a technique to remove spurious bright points which can be regarded as a form of noise. However, astronomical images are generally contaminated with other sources of noise, including Poisson components associated with photon counts and Gaussian noise associated with the electronics of the detector (such as readout noise in CCDs).

Naive smoothing or spectral filters have a tendency to blur the image, which can obscure much of the detail we sought to highlight with the feature enhancement algorithm discussed in Section 3. More sophisticated noise removal algorithms seek to isolate the noise component from the legitimate signal, attenuating the former while retaining even the high-frequency components of the latter. In short, we wish to remove the noise while retaining the sharpness of the image.

Noise in astronomical images contains both Poisson and Gaussian components, producing a broad power spectrum that is far from flat (white noise). Sophisticated noise removal algorithms model this noise spectrum based either on known performance characteristics of the detector or on the statistical properties of one or more images. If one can reliably model the noise spectrum, one can selectively remove it while retaining as much of the signal as possible, including the high-frequency components that give rise to fine structure.

DeForest (2017) describes one such model, known as noise-gating. As with other image and signal processing algorithms, noise-gating is a general technique that has been applied across diverse applications, most notably in audio recording. DeForest adapted the technique to astronomical observations and demonstrated its promise as a noise filter for solar imagery.

Noise-gating works best with a sequence of images over a given time interval. Temporal coherence can then be used to more effectively distinguish physical image features from noise. In modeling the noise spectrum, DeForest makes a distinction between shot noise which depends on the image itself and background noise which is independent of the image. An example of shot noise is the Poisson noise that arises from the counting of photons in a CCD detector. For large photon counts, this is approximately proportional to the square root of the image brightness, with Gaussian randomness:

$$N_s(x,\ y,\ t)\ \approx\ \alpha G(x,y,t)\sqrt{B_0(x,y,t)} \tag{5}$$

Here *x* and *y* are spatial position on the image, *t* is time, *G(x, y, t)* is a Gaussian distribution, and B_0 is the "ideal" noise-free data. The coefficient α , as well as G, reflects the properties of the detector.

Estimation of the noise spectrum from the data begins by organizing the data into a 3D data cube of dimensions $N_t \times N_x \times N_y$. Here N_x and N_y represent the dimensions of each image and N_t is the number of images in the sequence. This data cube is then divided into subcubes, or samples, each of size $n_t \times n_x \times n_y$.

Under the assumption that the zero-frequency component dominates the Fourier transform of $\sqrt{B_0}$, the spot noise spectrum can then be approximated as:

$$\hat{N}_s(k_x, k_y, \omega)\ \approx\ median_i\left\{\frac{|\hat{B}_i(k_x,k_y,\omega)|}{\beta_i}\right\}\sum_i \beta_i \tag{6}$$

where

$$\beta_i = \sum_{x,y,t} \sqrt{B(x,y,t)} \quad . \tag{7}$$

The subscript *i* refers to a particular sample and the summation in eq. (18) proceeds over all pixels and images in a single sample. The median and summation operations in eq. (17) are carried out across all samples. Apodization is performed on each sample to mitigate spurious edge effects.

Similarly, an image-independent background spectrum can be approximated as follows

$$\hat{N}_b(k_x, k_y, \omega) \approx median_i[\hat{B}_i(k_x, k_y, \omega)] \tag{8}$$

The assumption for both equations (17) and (19) is that a significant fraction–approximately half or more–of the Fourier space at the sampled scales is noise-dominated. If that is not the case then the median operations can be replaced by a lower percentile.

After an estimate for the noise spectrum is obtained, a filter is applied that either removes or attenuates Fourier components that have an amplitude less than a specified factor of the noise level.

Deforest has implemented his noise-gate algorithm and made it available in an open-source repository[2]. The code is written in cython; python with C extensions to improve numerical efficiency. Once installed, a sample python invocation looks like the following:

```
import noisegate as ng
ngdata = ng.noise_gate_batch(datacube, cubesize=(18,18,18), model='hybrid',
                             factor = 4.0, dkfactor = 1.5)
```

Here datacube is a 3D numpy array with dimensions (N_t, N_x, N_y), and the cubesize argument defines the dimensions of each sample (n_t, n_x, n_y). The model parameter is here set to hybrid, which includes both spot and background components. The factor and dkfactor arguments set the filter thresholds for the spot and background components respectively. This is the most appropriate option for unprocessed coronagraph images, though feature enhancement could complicate the estimation of a consistent Poisson noise spectrum. The background-only component, selected by setting the model parameter to constant, may be more robust in this case, as addressed in Section 4. Other options for model include spot and multiplicative.

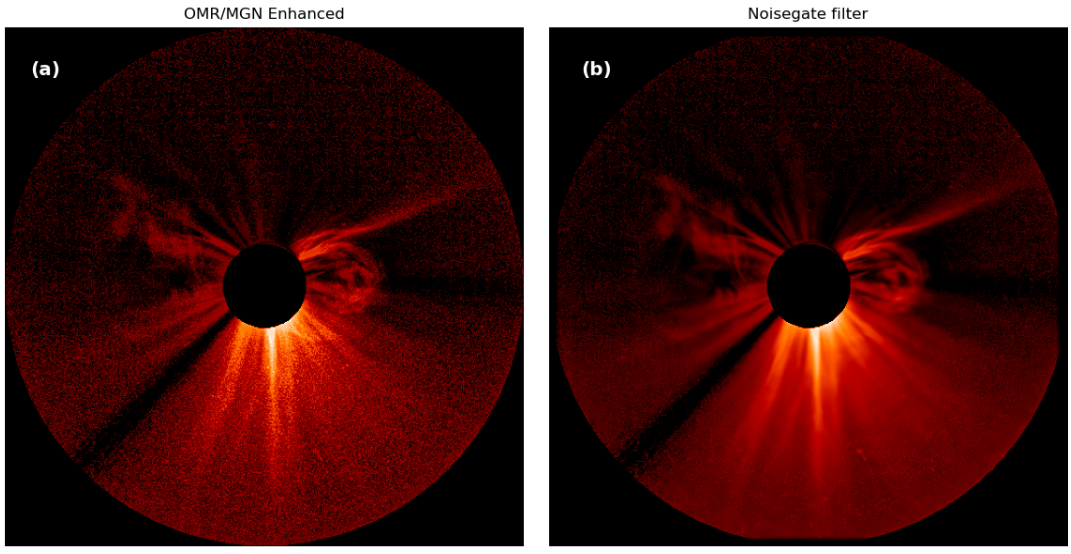

Figure 4. (a) Enhanced LASCO/C3 image from Fig. 3b. (b) Image (a) subject to a hybrid noise-gate filter (DeForest 2017).

[2] https://github.com/drzowie/image-noise-gate

Figure 4 shows how the noise-gate filter performs on the enhanced LASCO/C3 image from Fig. 3b and demonstrates how key features within the image, such as CMEs, are retained, while also clearly removing noise and cleaning the image.

Since the noise-gate filter is applied in Fourier space, the output image sequence produced by noise_gate_batch is apodized in time to mitigate edge effects associated with the finite time series. In practice, this means that N_a images near the beginning and end of the sequence are darkened, where N_a is ⅔ of the temporal cube size n_t. For example, N_a = 12 for the value of n_t = 18 used above. The apodization function can be inverted, but this is at the expense of somewhat higher noise levels near the edges of the image sequence.

A straightforward way to handle this apodization at the beginning of the image sequence is simply to include N_a more images in the sequence and exclude them from the result. However, this cannot be applied to the end of the sequence for operational applications, where viewing the most recent image is essential. A simple solution is to replicate the last N_a images of the sequence and add them to the input data cube in reverse order. Then the last N_a images can be excluded from the result. With this approach, the output sequence ends with the most recent image. An example python implementation of this suitable for operational use is as follows:

```
import numpy as np
import noisegate as ng
def noisegate(images, cubesize = (12, 12, 12)):
    nap = int((2*cubesize[0])/3)
    nt, nx, ny = images.shape
    dcube = np.zeros((nt+nap, nx, ny), dtype = 'float')
    dcube[0:nt,:,:] = images
    dcube[nt:,:,:] = np.flip(images[nt-nap:,:,:], axis = 0)
    dcubeng = ng.noise_gate_batch(dcube, cubesize=cubesize, model='constant', factor = 6.0)
    return dcubeng[nap:-nap,:,:]
```

The noise-gate filter is the only part of the L3 processing pipeline that requires the processing of multiple images simultaneously. This in itself should not pose problems for operational execution: the noise spectrum for the latest image can be approximated from previous images as described above. Furthermore, the use of the median as a robust statistical measure in equations (6) and (8) makes the estimation of the noise spectrum insensitive to the potential presence of corrupted images.

4 References

C. E. DeForest 2017, “Noise-Gating to Clean Astrophysical Image Data”, *Astrophysical Journal*, **838**, 155 (10pp).

H. Morgan & M. Druckmüller, 2014, “Multi-Scale Gaussian Normalization for Solar Image Processing”, *Solar Physics*, **289**: 2945–2955.