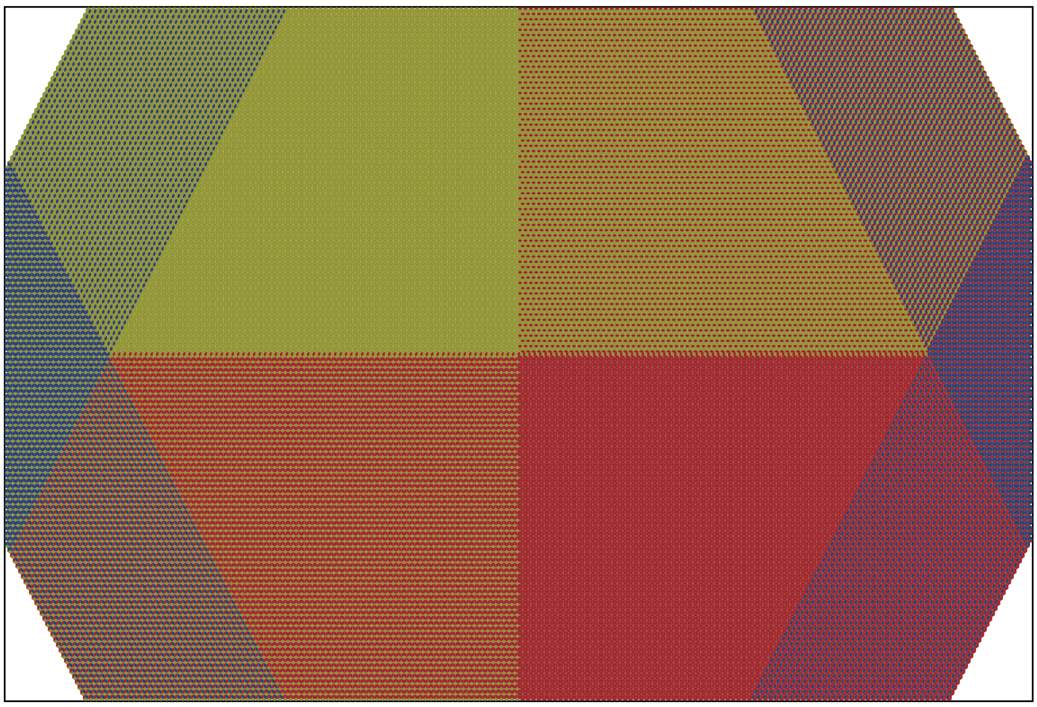

Figure 1: A diagram given to me by the Polish collaborators via Kristian.

author: Ocean Wong
collaborator: Kristian Haverson
date: 2023-05-01
Organization: Sheffield Hallam University
Culham Centre for Fusion Energy

Abstract

This pdf explans and documents the rationale of the design choices behind the trail extraction algorithm. (currently named `fitter.py`.)

1 The problem

We have three sets of strip-detectors pointed in the u-, v- and w-, offset from each other by 60° (Figure 1. Each event consist of tracks made by anywhere between one to four particles, which are then drifted towards the strips by an electric field. Any strip that intersect with the track detects a signal (seemingly giving a reading = one of the integers between 0-20) at every time bin. For tracks in the $u-$direction, we can plot the strip ID number in the vertical direction, and time bins in the horizontal direction(i.e. the z-direction profile of the track as projected onto the $u-$strips). This can be plotted in the red channel of a 510×510 .png file. Coincidentally, the number of time bins for $u-$strips = that of $v-$strips = that of $w-$strips = 510, and the number of $u-$strips = that of $v-$strips = that of $w-$strips = 510. Therefore, we can simply place in the same 510×510 matrix from the $u-$strips in the green channel and the 510×510 from the $w-$ strips in the blue channel as well, forming a matrix of shape (3, 510, 510), forming an image like Figure 2.

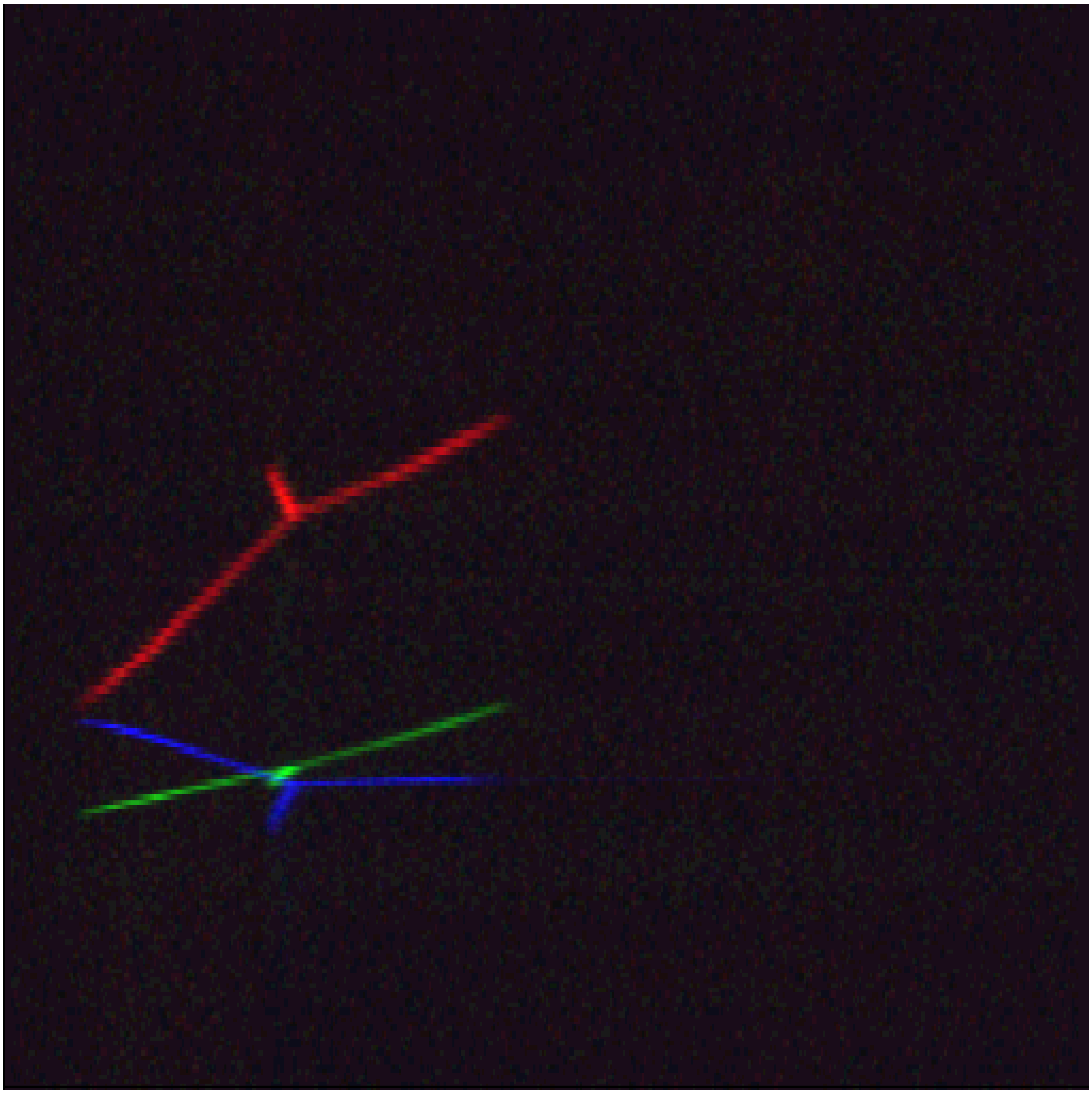
Figure 2: A sample of a 3-prong event. Vertical direction = strip number, horizontal direction = time bin.

We want to find out the energy of the the particles by the amount of charge they left behind and the angle at which events occur. Therefore, we have to:

1. Determinte the type of event that occured, for each event triggered and captured;
2. Find out the energy (scalar quantity) and momentum (vector quantity) of each particle, by deciding:
    - the amount of ionization left behind by each particle,
    - the length of the track left behind by each particle,
    - the angle that the track makes with other particle's tracks.

To do this, I have chosen the following workflow:

1. reconstruct the 3D shape of the track
2. find the bi-/tri-furcation points of these tracks (for events where > 1 particples are involved and multiple tracks are left), so that the direction of travel for each particle can be determined.
3. [optional] improve upon this guessed tracks by fitting (using these guessed tracks as the starting points), so that the $\frac{d\text{energy deposited}}{d\text{unit length}}$ at each point along the track can be recovered.

1.1 The strategy

To extract the 3D shape of the track, we must first identify where the tracks are on the 2D plots in the red, green and blue channels of the png picture, in the form of a lists of coordinates for the "backbone" of each strip-direction. Then we can decide the type of event that this belongs to, so that we can combine these three 2D backbones into a single 3D backbone.

2 2D backbone extraction

Given the 510×510 pixel image (each pixel can take a value between 0-20), we need to extract the backbone of (and possibly outline the area left behind by) each track. This is a deceptively difficult problem. Luckily we are not the first people to encounter this type of problem where "ridges" need to be extracted from an image. Similar problems had occurred in medical imaging, such that there are plenty of tools available in open-source libraries such as `scikit-image` to perform these types of extraction.

My intuition is that the backbone can be extracted more easily if we first determine the area of interest, i.e. highlight the blob of pixels that clearly belongs to the tracks and ignore the pixels that belongs to the background; my intuition says that any doing this in the reverse order would be much more difficult.

Feature extraction like this is a trivial enough task, and could be done with a convolutional neural network; but in the spirit of reducing the number of points of failure (i.e. reducing the number of unknown blackbox parameters that can go wrong) and making sure that the data analysis process is transparent and physically intuitive (so that we can continue to wrangle with the data confidently further down the chain of analysis), I have opted to do this using manually chosen model and manually tuned parameters.

2.1 Determining the background

I noticed that in some cases, some strips have elevated background (see Figure **??**. My intution is that applying a global threshold instead of a per-row threshold would cause the entire streak to be identified as "part of the track", which is obviously not the case. To avoid the misclassification of these types of events, I have written a simply filter that performs the following algorithm:

1. Determine the brightness percentile curve for each row(i.e. for each strip across the entire 510 time bins),
2. The "background" for the entire row is set as the $n\%$ percentile of brightness.
3. Pixels with brightness $>$ the $n\%$ percentile are highlighted as the foreground.
4. "Connected Components" are identified - i.e. all pixels belonging to the same discrete "blob" is given the same ID (This step considers pixels as neighbours if they share at least one edge (Von Neumann neighbourhood)).
    - But any cluster with less than m pixels highlighted would be considered too small to keep, and will be discarded.

Via manual fitting (trial and error), $n = 95\%, m = 40$ was found to be the best settings for the current problem.

In this section we shall use one of the most pathological example (Figure **??**) to demonstrate the extraction process. After applying the steps demonstrated in this sub-section (Section 2.1), Figure 4 plot only the extracted connected components in the $u-$direction strips vs time plot.

2.2 Cleaning the foreground

Once the background is determined, it is fairly easy to determine what is in the foreground using simple negation of boolean variable. But these often include some regions of noise, large enough to not have been rejected in the last step of Section 2.1 (false positives), while some parts of the trail may have been ignored because individual pixels within the trail may have dipped below the $n\%$ percentile due to statistical fluctuation (false negatives).

To reduce these false positives and false negatives (i.e. increase the area under the ROC curve), the

1. After the discarding step above, Connected Components are expanded by re-identification using a lowered percentile threshold of $n'\%$. The definition of "neighbouring" pixels is also relaxed to "sharing at least one vertex (Moore neighbourhood)". This picks up any previously ignored pixels on the trail (the false negatives), and coalesces jagged, broken, but neighbouring components together.
2. And then any blobs (i.e. "connected component") with "mean prominence" $< p$ is also discarded, where we define

   $$\text{prominence} = \text{raw pixel value} - \text{background threshold } (n'\% \text{ percentile}). \quad (1)$$

   This removes the false positives.

Via manual fitting (trial and error), $n' = 85\%, p = 1.5$ was found to be the best settings for the current problem.

After applying the steps demonstrated in this sub-section (Section 2.1), Figure 5 plot only the extracted connected components in the $u-$direction strips vs time plot.

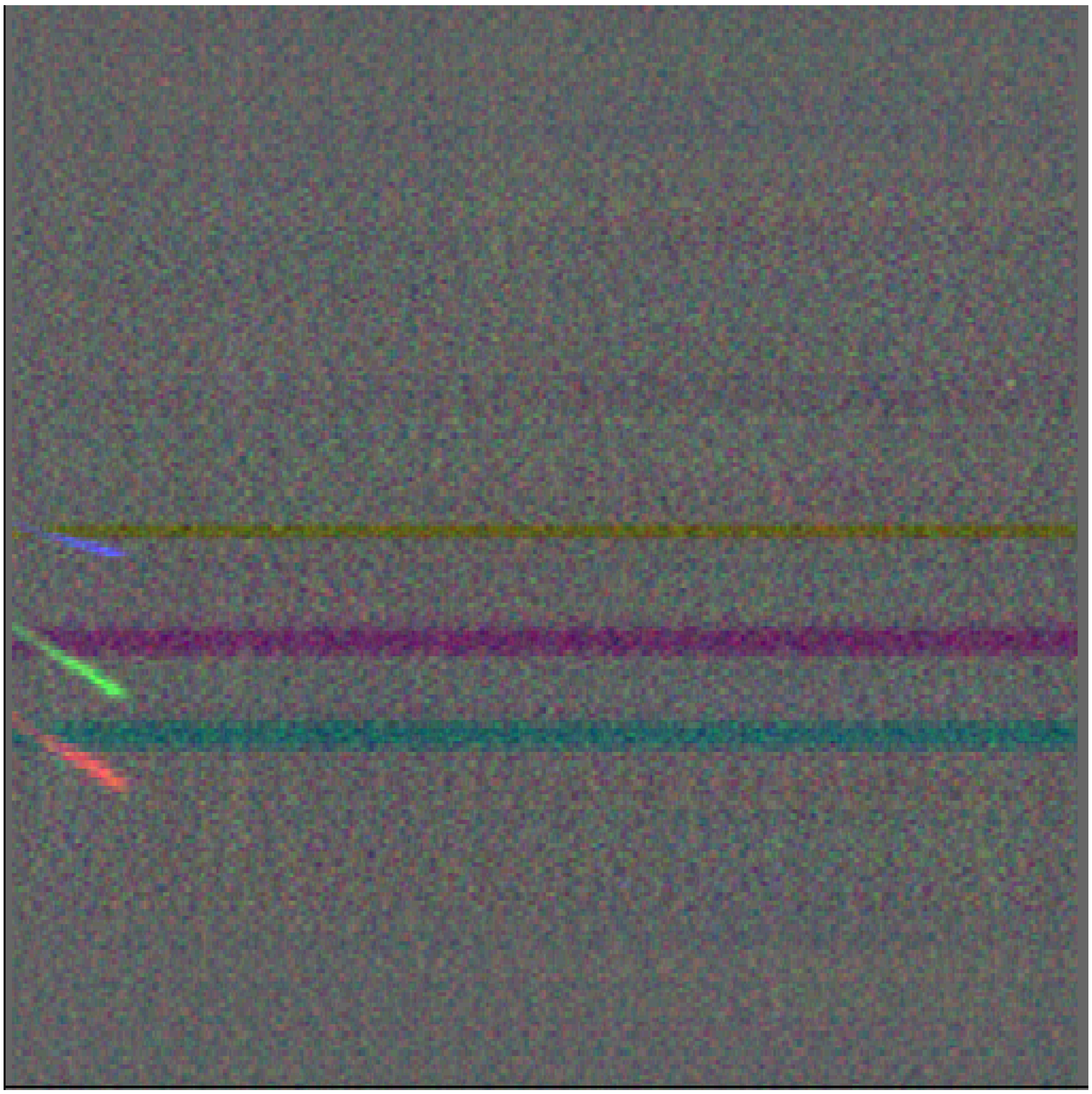

Figure 3: A sample of a 1-prong event. Note the horizontal streaks which may have been caused by the incorrect bias subtraction as the event was triggered too late, was overlapping with the tail of another event, or any other electronics error.

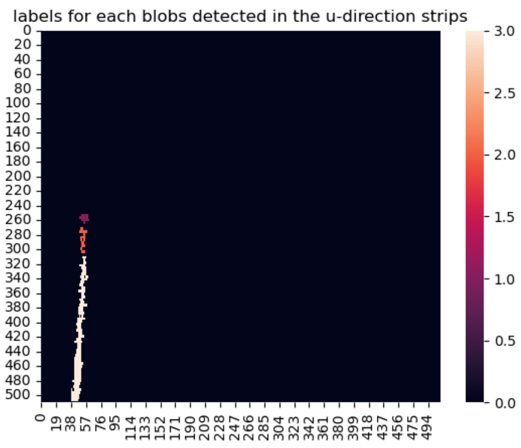

Figure 4: Notice how there are 3 components (3 colours are used) even though all three of them should have belonged to the same trail.

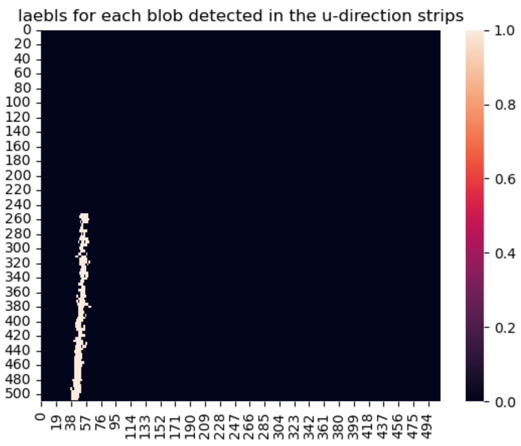

Figure 5: Now after the cleaning step, there is only 1 single colour (1 component, corresponding to the entire trail) in the entire graph as we expect.

2.3 Extracting the outline

Determining the outline for each blob the pixel is conceptually easy but computationally untidy. After some experimentation, I have chosen to do it the lazy but fast way, which is to "keep walking clockwise while touching the interface between the background-foreground element until we return to the starting point". (I have not noticed any faster way of doing this, but if there is any out there I would be happy to implement it.)

2.4 Extracting the backbone

skeletonize

2.5 Development log

I was initially unaware of `scikit-image`'s extensive image processing library applicable for these types of feature extraction purposes, but after stumbling across a YouTube video about cleaning a LiDAR-acquried floor-plan using the `skimage.morphology.skeletonize` function, I attempted to use this library and discovered many more useful functions. If I were introduced to the `skimage` library earlier, I would have taken a slightly approach and used other `skimage` functions in Section 2.1 to reduce the workload and increase the time available for experimentation with manually tweaking parameters in the functions to optimize extraction.

In fact, I have looked through and experimented wiht the complete `skimage` library. While some of them do not fit my use case at all, I have honed down on a few that

- Step 2 of Section 2.1 would have been replaced by `skimage.filters.frangi(sigmas=range(4,18,2)` `skimage.filters.meijering(sigmas=range(4,18,2), black_ridges=False)` or `skimage.filte` These are shown to be excellent at identifying the ridges through manual experimentations.

- Step 3 of Section 2.1 would have been replaced by `skimage.morphology.remove_small_objects`. I'm sure the results would have been similar, but the performance may have been improved.

- An optional step 4 may be added to Section 2.1 using `skimage.filters.apply_hysteresis_thresho`

This is not intended to be a complete documentation so only a few demonstrative plots will be inserted. But for example, for the particular difficult case showing the data collected by the $u-$strips in one of the events (Figure 6): The Frangi (Figure 7), Meijerling (Figure 8), and Butterworth (Figure 9) filter managed to highlight the track sufficiently, so that when a simple global-threshold algorithm or the hysteresis-threshold algorithm is applied, it can extract the outline of the tracks left behind by the two particles.

But upon further examination, they exhibit poorer performance in some other cases (Figure 10 to 12). This is because only acts to destroy existing information, without adding new information. It is clear that any thresholding done on these plots would yield a worse result than thresholding on the raw data.

Other thoughts and notes for myself:

- I won't try `graph.MCP` etc. because it would be quite difficult to use minimum(/maximum)-cost-path in this scenario where bifurcation points arises.

- I won't use `transform` (including inverse radon and hough line) transform because the paths aren't always straight.

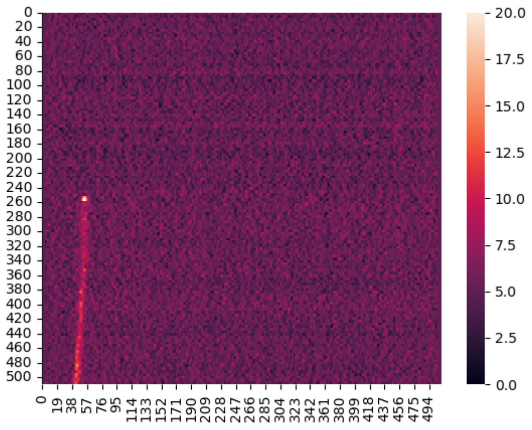
Figure 6: The raw data collected by a single direction of strips (the $u-$direction).

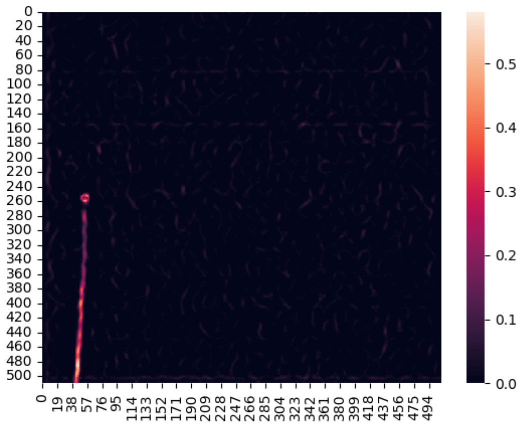
Figure 7: Frangi filter applied

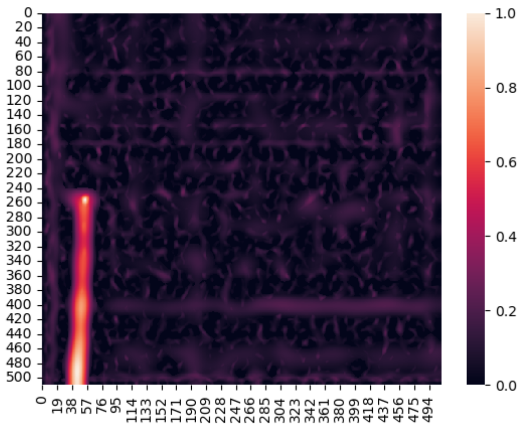
Figure 8: Meijerling filter applied

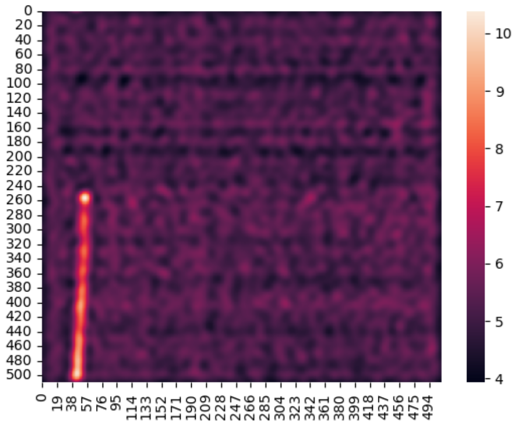
Figure 9: Butterworth filter applied

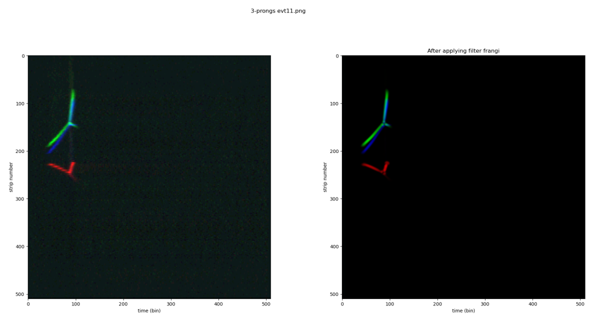

Figure 10: Case where the Frangi filter lose luster.

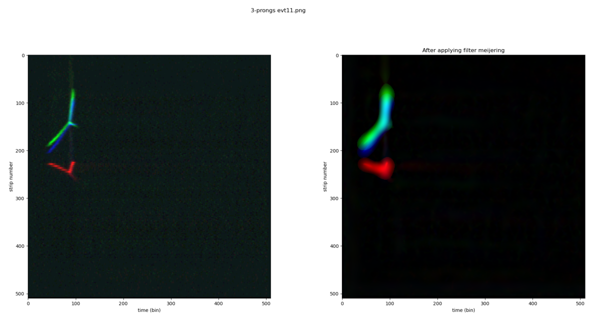

Figure 11: Case where the Meijerling filter lose luster

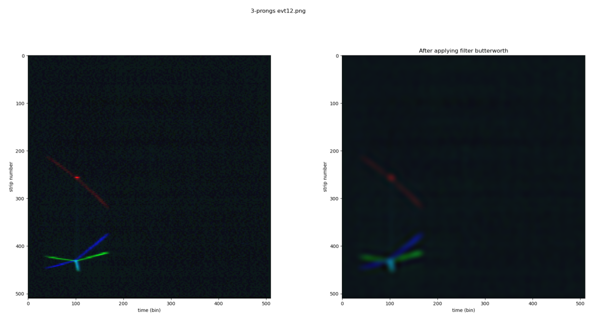

Figure 12: Butterworth filter applied