Detecting Moving Objects in DAS Recordings

Description of our Approach

Aleksander Hański, Jakub Radziejewski

Segments we analyzed:

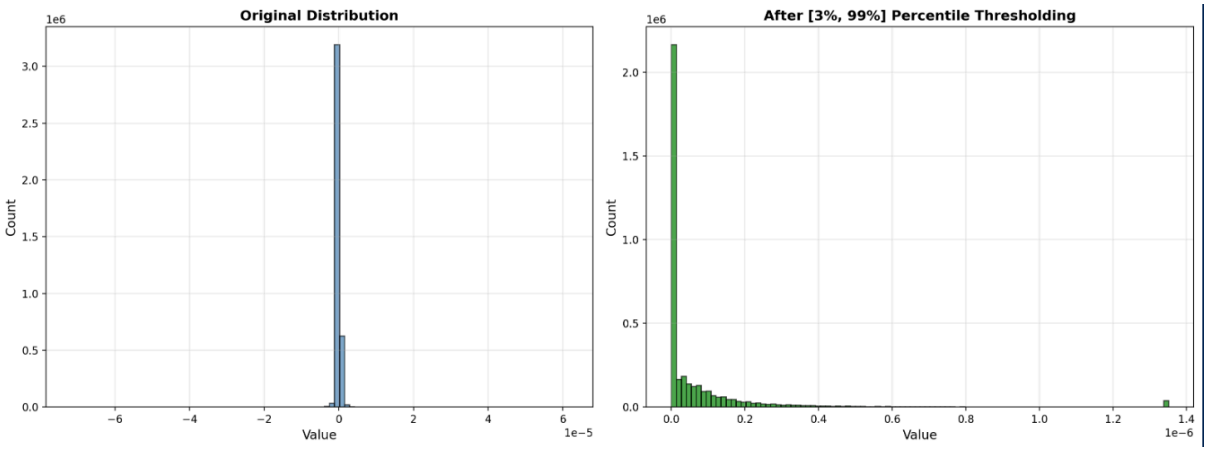
1. Time: 2024-05-07, 09:05:22 to 09:07:12
2. Time: 2024-05-07, 09:32:52 to 09:34:42
3. Time: 2024-05-07, 09:21:12 to 09:23:02

Dataset analysis for Segment 1

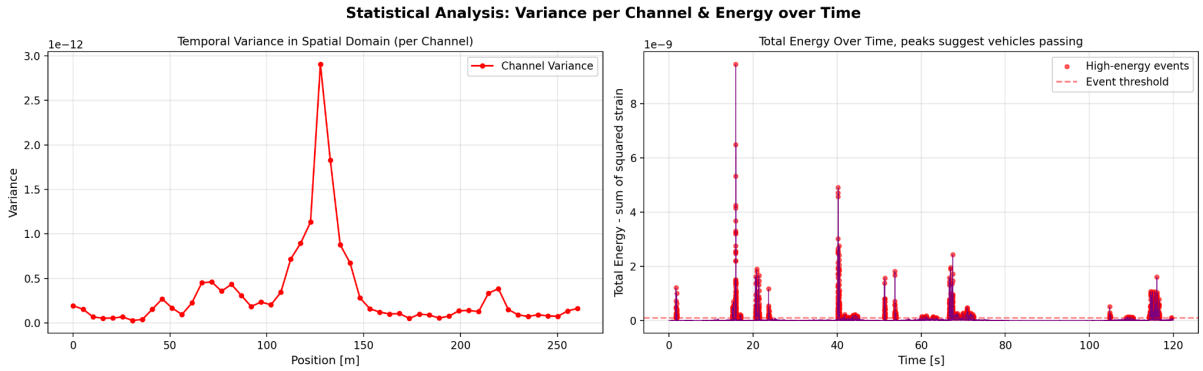
Basic statistics

Basic statistic for a segment:

- Mean: -6.268834e-11
- Std: 5.668068e-07
- Range (min-max values): [-7.18e-05, 6.14e-05]
- Median: 0.000000e+00
- 95th percentile: 4.29e-07
- 99th percentile: 1.35e-06



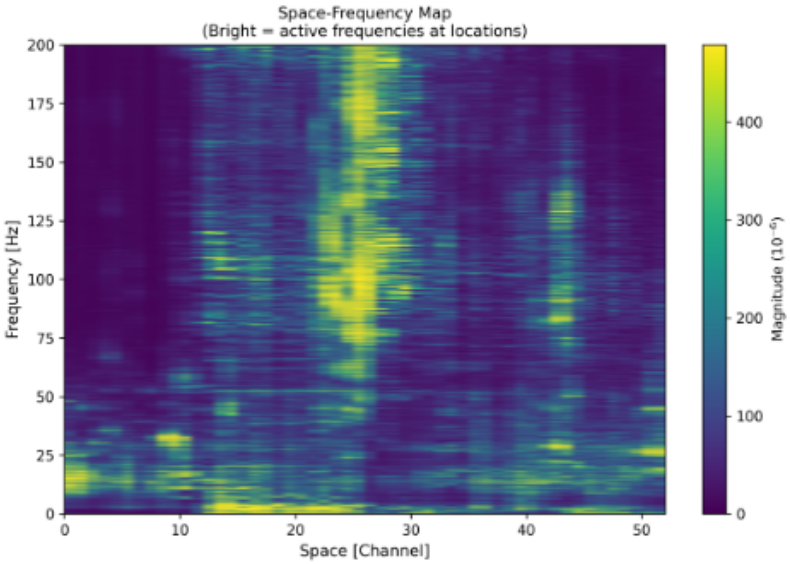
Analysis of the distribution reveals extreme outliers far from the mean. Even after applying [3%, 99%] percentile thresholding, data retains significant variance.



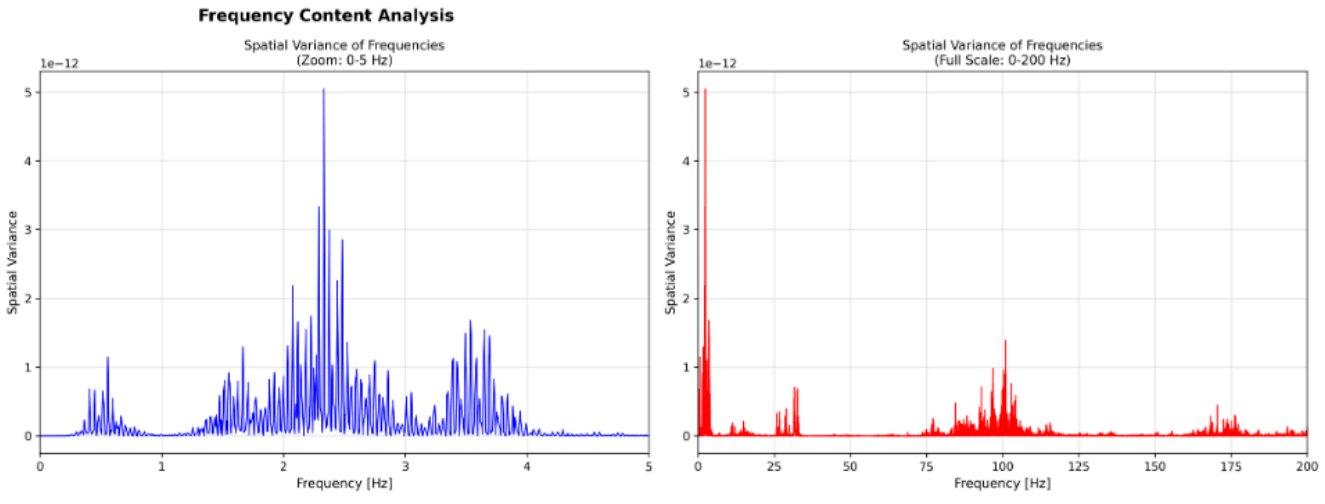
These two plots show spatial variance across fiber channels and temporal energy evolution with detected events marked with red dots. The left one reveals which sections of the cable may contain noise (like around 125m), while the right one shows precisely when significant events may occur in the time series. However we didn't use it when finding where the events are, it was some of our initial ideas which may have been useful.

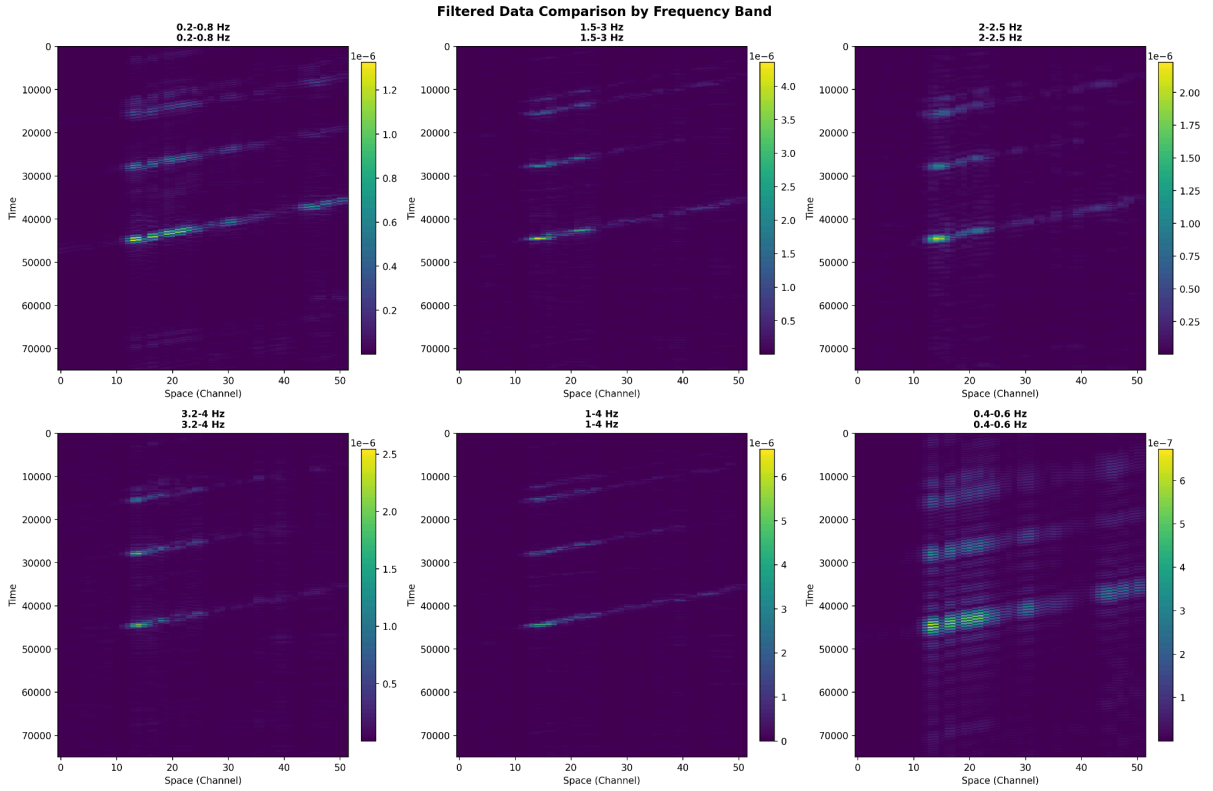
Frequency Content Analysis

The space-frequency map (on the right) displays where different frequencies are present along the cable. To compute it, FFT was applied along the time axis for each channel, transforming temporal signals into their frequency components.



The spatial variance spectrum (below) shows which frequencies have the highest variance as far as frequencies are concerned. This also was one of the ideas to use when doing preprocessing, but high variance here is not only connected to moving vehicles, more often it connects to noisy channels. The spatial variance spectrum calculates the variance of FFT power across all channels at each frequency.



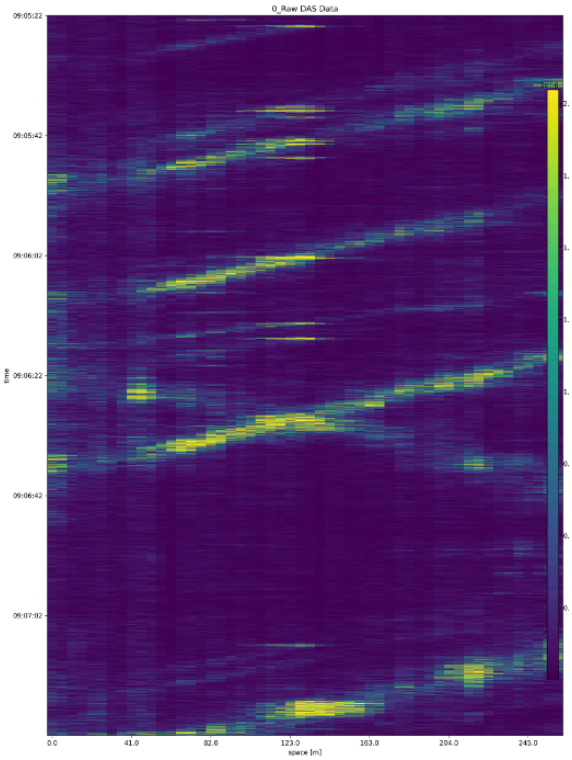


These six side-by-side visualizations show the same data filtered to different frequency ranges.It applies FFT-based bandpass filtering to isolate specific frequency ranges (1.5-3 Hz, 1-4 Hz, etc.), then transforms back to the time domain via inverse FFT. The point was to see whether all lines will be clearly visible in the highest variance ranges in our data. Unfortunately only 3 strongest signals are shown, not others. That's why we didn't use it in actual preprocessing.

Preprocessing

The objective of this preprocessing pipeline is to isolate diagonal lines created by moving vehicles and filter out the noise present in raw DAS recordings. This is achieved through a multi-stage approach:

Raw DAS 09:05:22 - 09:07:22 recording

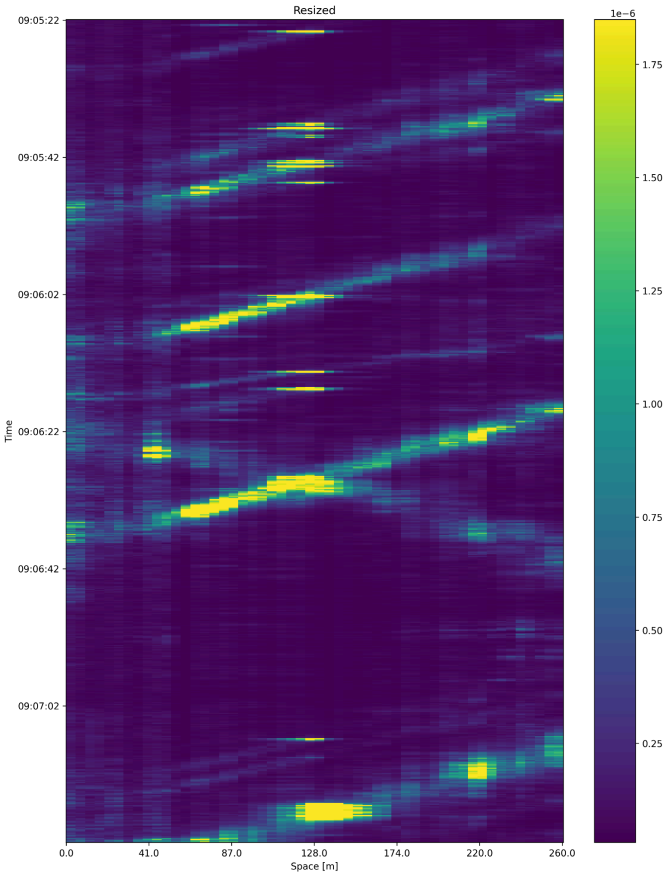


Centering Data and Taking Absolute Values

We subtract mean from the data, and convert all negative values to positive. This preprocessing symmetrizes the data around deviations from the mean, allowing us to focus on magnitude of difference from the mean.

Resizing to square

The Hough line detection algorithm performs optimally on images with similar dimensions. We squeeze the image along the time axis using averaging (combining multiple adjacent values into a single value by taking their mean), which reduces data size while preserving most information and minimizing distortion.

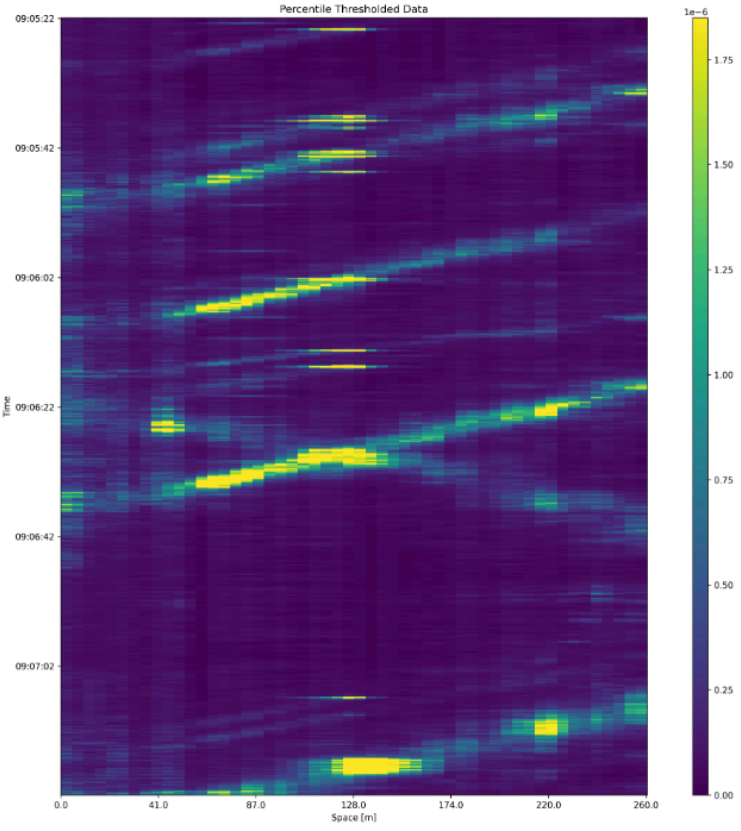


Percentile-Based Thresholding

We calculate the 3rd and 99th percentile thresholds.:

- The low threshold (3rd percentile) zeroes out all remaining low-amplitude background activity, establishing a dark, noise-free background.

- The high threshold (99th percentile) clips extreme amplitude outliers, ensuring that the dynamic range is maximized, and we don't have any spikes caused by noisy channels.



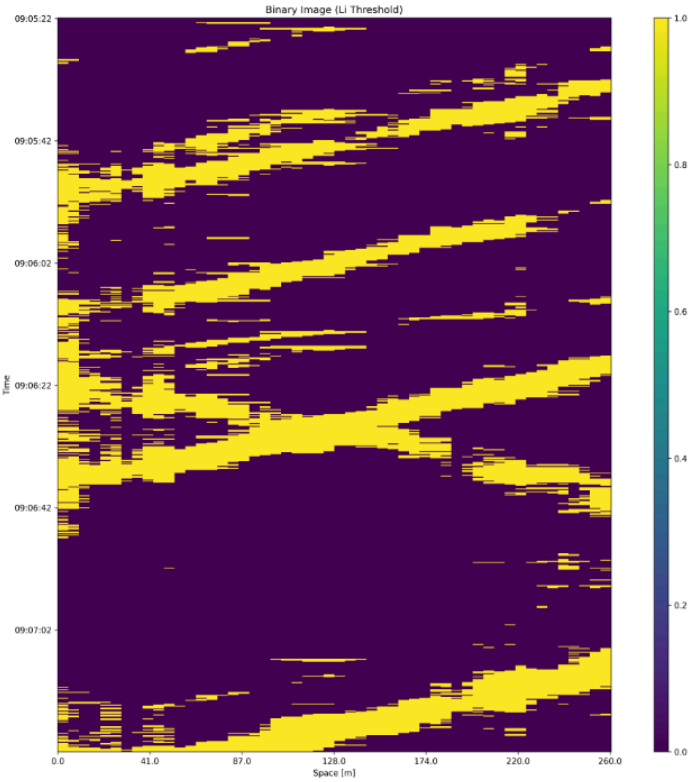
Normalization to [0,1]

Normalization is performed after outlier removal to ensure it operates on a meaningful intensity range rather than being skewed by extreme values.

Applying Adaptive Thresholding

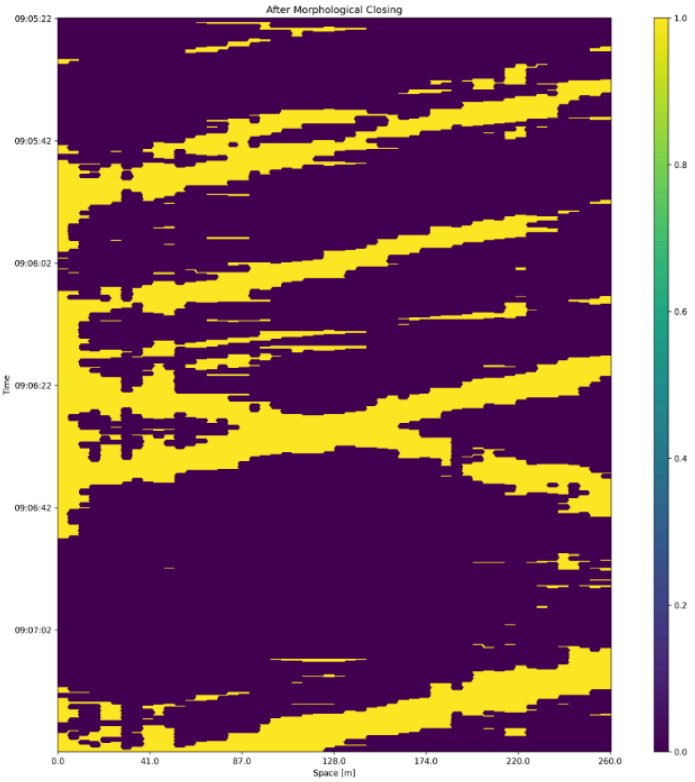
We use Li adaptive thresholding - for each candidate threshold, Li's method divides pixels into foreground/background, and calculates information loss between original image and binary result. It selects threshold that loses least information when converting to binary.

To increase sensitivity to fainter lines, we multiply the Li threshold by 0.8 (threshold_multiplier), making the algorithm more aggressive in detecting subtle features.



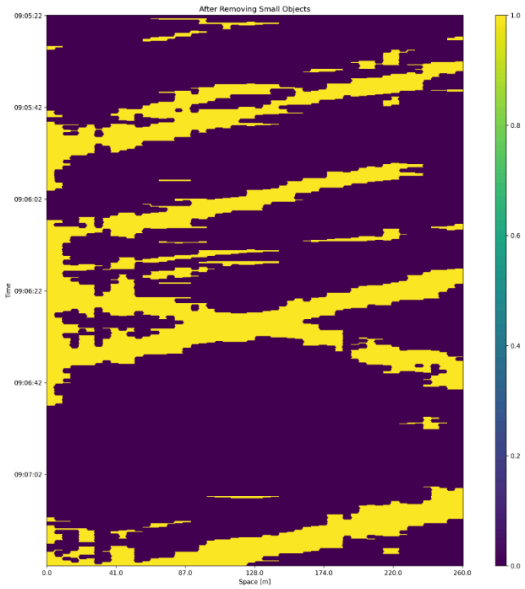
Applying Morphological Closing

This step fills in small gaps and connects broken pieces of vehicle tracks. By performing a dilation followed by an erosion, closing fills small gaps caused by momentary signal dropouts or intermittent noise and applied thresholding, making it easier to extract the lines in the next stages of development.



Removing Small Objects

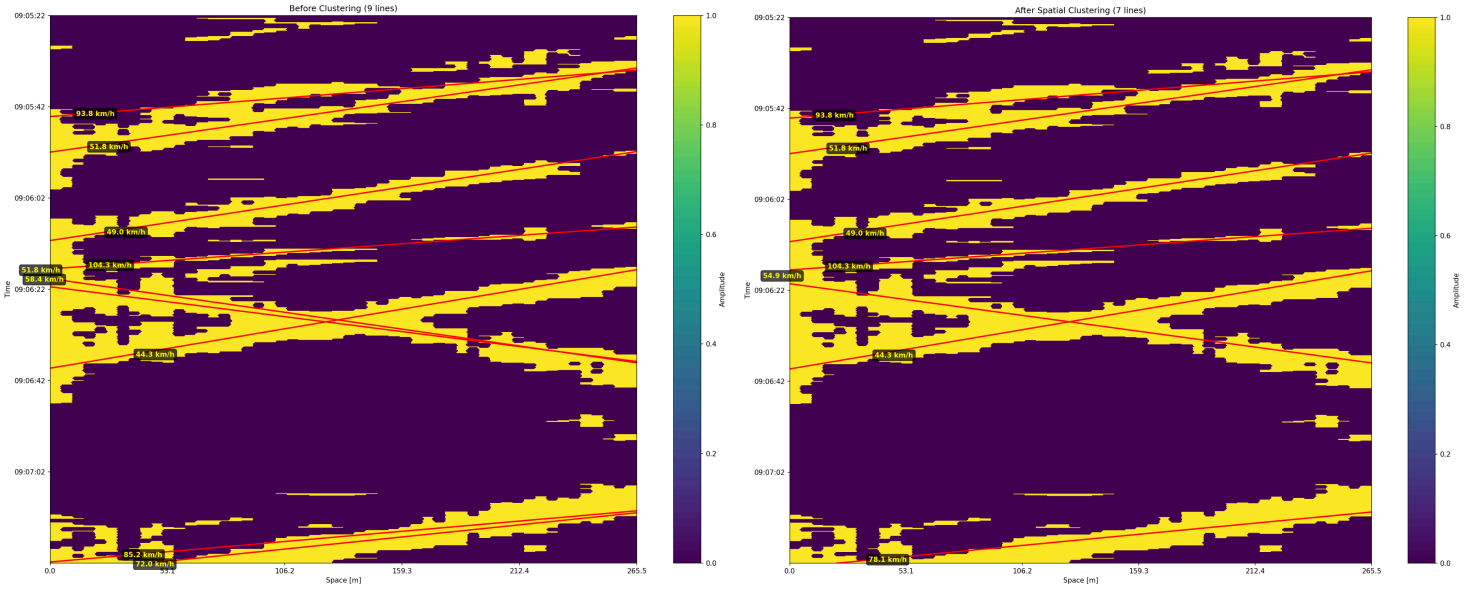
We remove small connected components from the binary image that contain fewer than 100 pixels. This eliminates noise artifacts and isolated pixels that are not part of genuine line structures, cleaning the image before final line detection.



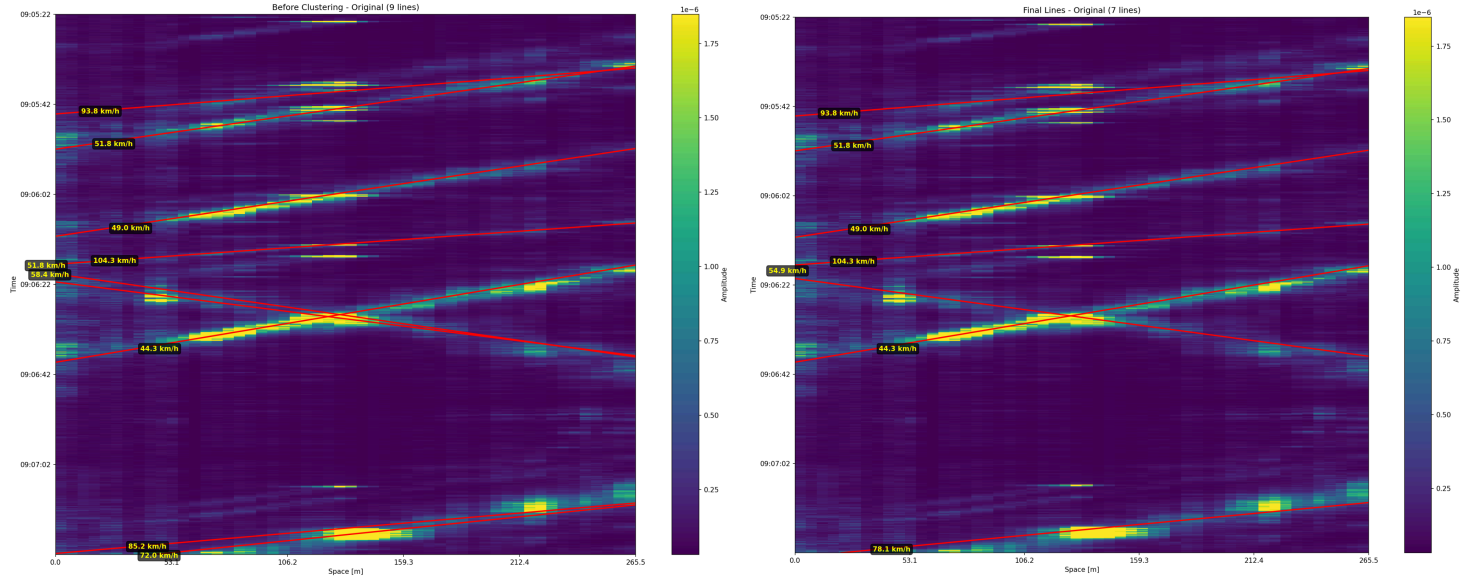
Line detection

We detect lines using Straight Line Hough Transform.
For each line we calculate velocity, and filter out lines by realistic velocity range.
Then we perform spatial clustering on lines - if two or more lines lie close to each other, we cluster them into one line. If a cluster has an odd number of lines, we return a line with a middle angle value as representation of the cluster. If a cluster has an even number of lines, we return the mean of two middle lines as representation.

Detected lines before and after spatial clustering on binary images:

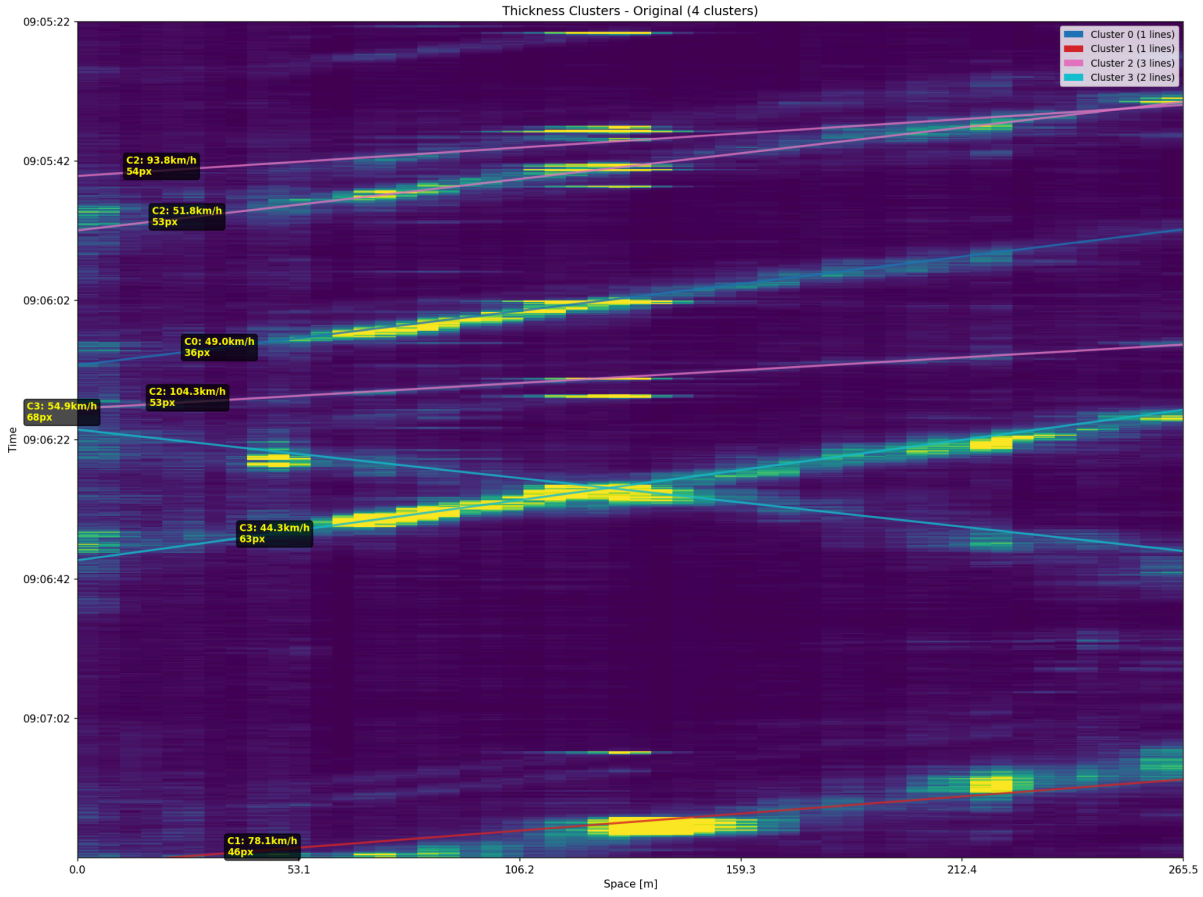


Detected lines before and after spatial clustering on original image:



Clustering Lines by Thickness

We create 500 evenly spaced sample points along line length. Each point measures thickness perpendicular to the line.
Each point looks at binary values of 100 pixels perpendicular to the line at that point, and finds the longest consecutive sequence of white pixels. This represents line thickness at that location.
Then we take the 90th percentile of all thickness measurements as line thickness - this makes it robust against noise and occasional gaps in the line.
We then normalize thickness values and perform clustering using Agglomerative Clustering.

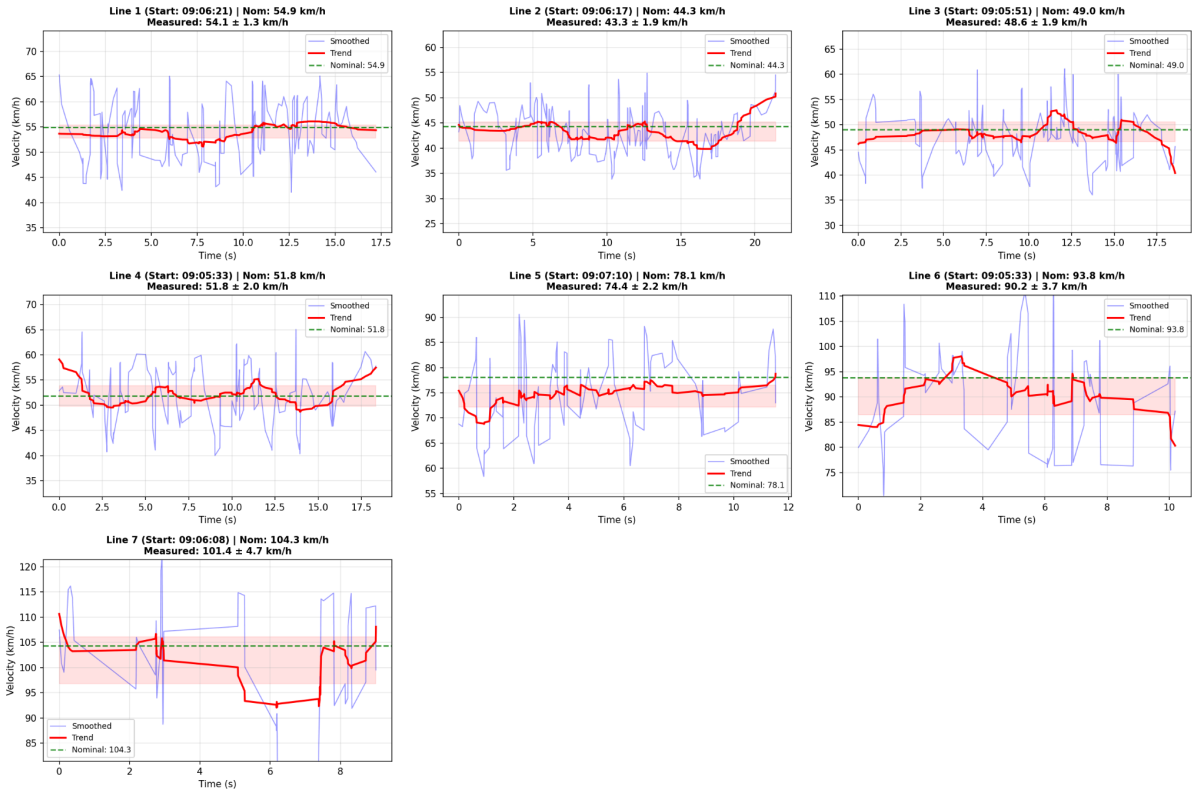


Velocity of the vehicles over time

The velocity analysis uses two main functions to measure how fast objects are actually moving in the DAS data and to verify that the detected lines make sense.

The *compute_velocity_field* function looks at how bright patterns move through the image over time. It smooths the data first, then calculates gradients with Sobel filters to see how things change spatially (across space) and temporally (across time). These gradients get combined into tensor components, and the velocity comes from their ratio using the Structure Tensor method, which basically measures how many pixels something moves per time frame. This gets converted to real units like kilometers per hour. The function also calculates a coherence score that shows if each velocity measurement is trustworthy or just noise.

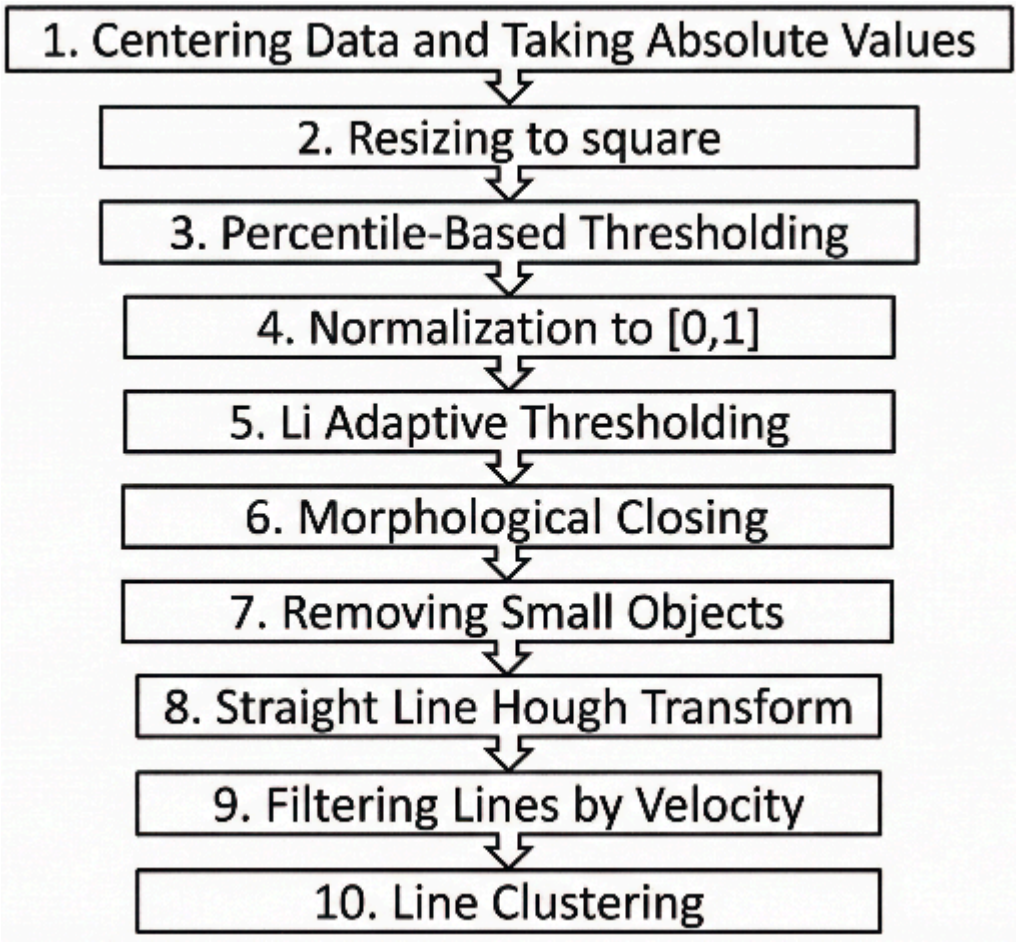
The *sample_velocity_along_line* function checks each detected line by pulling out actual velocity measurements along its path. It maps the line back to the original data coordinates and samples velocities in small windows along the way, using the binary mask to avoid background noise. It filters out measurements that don't match the expected speed (within thirty percent) or have low confidence scores, then averages what's left with higher weight given to reliable points. Finally, it smooths everything with different filters to get clean velocity profiles over time.



On the plots we can see each of the lines velocity over time, to detect which line from clustering is shown, we look at the starting time and nominal speed from the title and match it with the velocity of the clustered lines, one plot above this one.

Potential smoothing bias: gaussian smoothing with sigma=15 in time direction blurs fast-changing velocity patterns, potentially biasing measurements toward average speeds and missing rapid changes of velocities, also as we put a constraint on the velocities, to be detected within the range of ± 30% of the Hough-derived nominal velocity, to get rid of outlying pixels which were giving unreal measurements. The goal was to show a trend and then to show an approximate which is not an accurate value in each of the points.

Flow diagram:



Failed approaches

Resizing Image After Preprocessing

We knew that Hough Transform can work properly only on rescaled images, but we were performing resizing the image right before detecting lines, after processing the image.

Probabilistic Hough Transform

We wanted to combine Straight Line Hough Transform with Probabilistic Hough Transform. While Straight Line Hough transform returns lines in representation of angle and dist, probabilistic version returns lines in representation of two endpoints.

We had the idea to combine lines detected by both approaches in one picture. We would detect longer, thick lines using ‘normal’ Hough and shorter lines using Probabilistic Hough. But then in longer thick lines detections from both methods doubled. We checked what is angle, dist representation of Probabilistic Hough line segments, and excluded line segments that doubled lines detected by ‘normal’ Hough Algorithm. However, the results were not satisfactory as we detected mostly noise, so we abandoned that idea.