Computer Vision Project 1

Piotr Franc and Paweł Charkiewicz

November 2025

1 Algorithm Overview

The algorithm we built can be split into four main steps:

1. **Preprocessing with Fourier Filtering**
   First, we apply a Fourier transform and keep only the 80–120 Hz frequencies. After testing different ranges, this one turned out to work best: it cuts out most of the noise while still keeping the part of the signal that is actually useful for our method.

2. **Finding Vehicles on Specific Channels**
   Next, we detect vehicles using simple thresholding. We do this on channel 25 (127 m), because during our experiments this channel consistently showed the strongest and cleanest response. In other words, it's where vehicles are easiest to spot without getting confused by noise.

3. **Clustering the Thresholded Points**
   After thresholding, we group the detections into clusters, where each cluster corresponds to one vehicle. From each cluster we take two important values:

   - the time at which the vehicle appears, and
   - how long it stays visible in the channel.

   On the plot this basically shows up as an x, y position plus the width of the segment.

4. **Main Line Detection Step**
   Finally, for every detected vehicle we generate a bunch of candidate lines with different slopes. We check which of these lines score the highest across all channels. After that, we remove the lines that don't make sense physically (for example, ones that would require unrealistically fast or slow speeds).

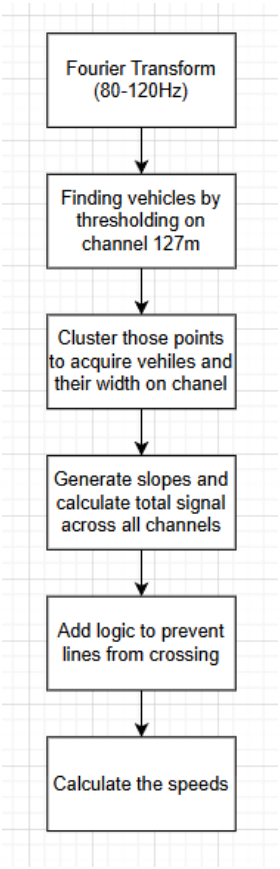

Figure 1: Algorithm flowchart image

2 Methodology

2.1 Fourier transform

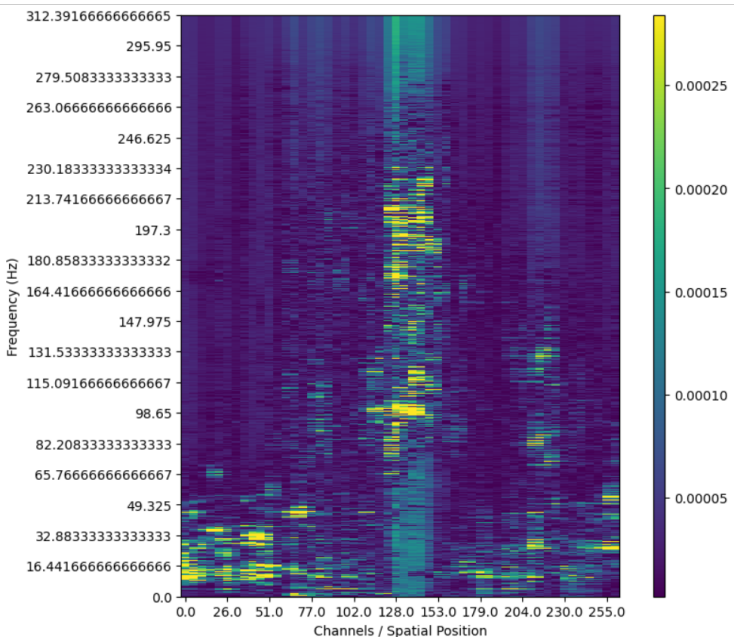

Figure 2: Frequency domain image

We tried to use fourier transform in many different ways - from filtering out noise to filtering out the lines themselves. The experiment with emphasizing the lines by using 1/time for each channel seemed promising but we just couldn't make it work - we decided to settle for the simple but as it turned out, very effective algorithm we came up with.

2.2 Canny and Hough

We also had a version of the algorithm that used bucketing the data (basically almost blurring it with resizing) and canny filter to run the Hough algorithm - for many hours we tried to make the values work, and when they finally did, it turned out that they failed for different time intervals.

3 Results

Below are the results of our algorithm - we ran it on our index's files and on the first file from the page (the random one)

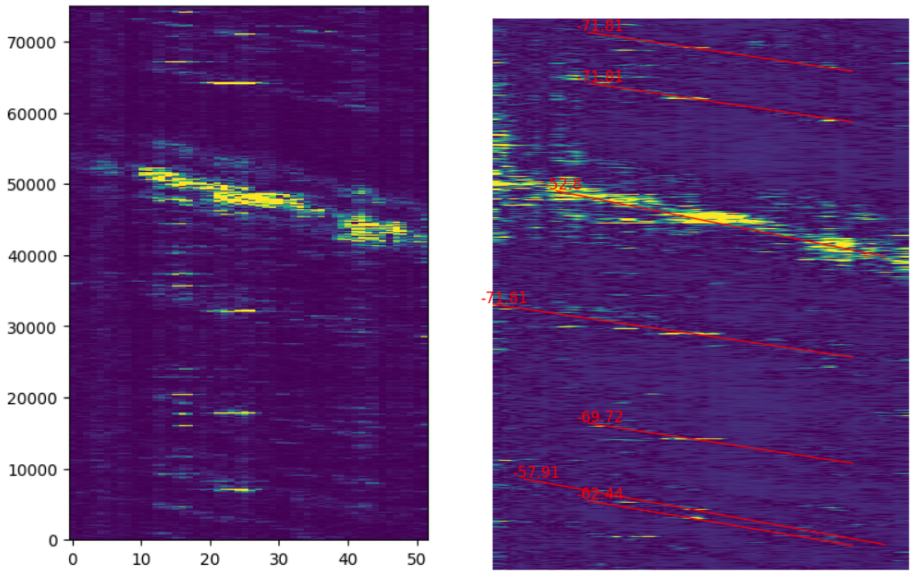

Figure 3: 160288 file

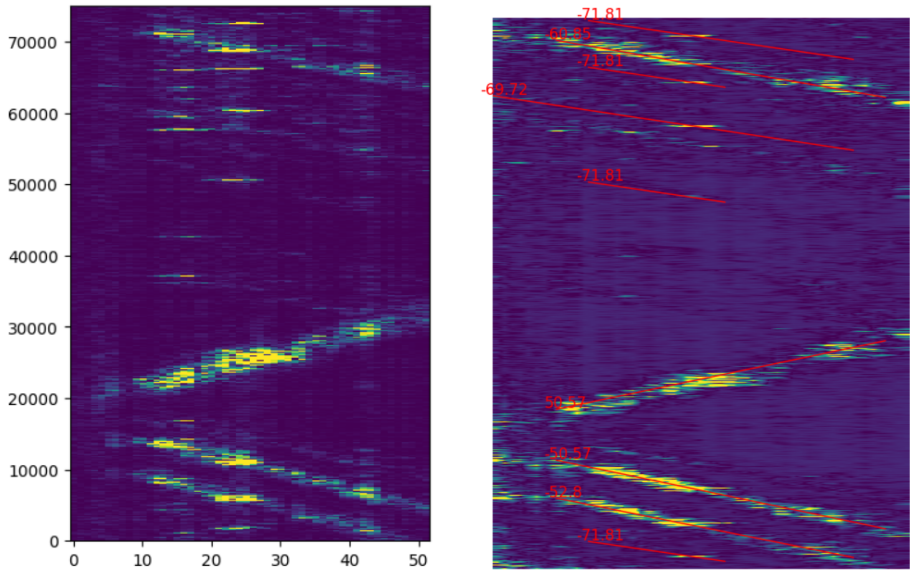

Figure 4: 160306 file

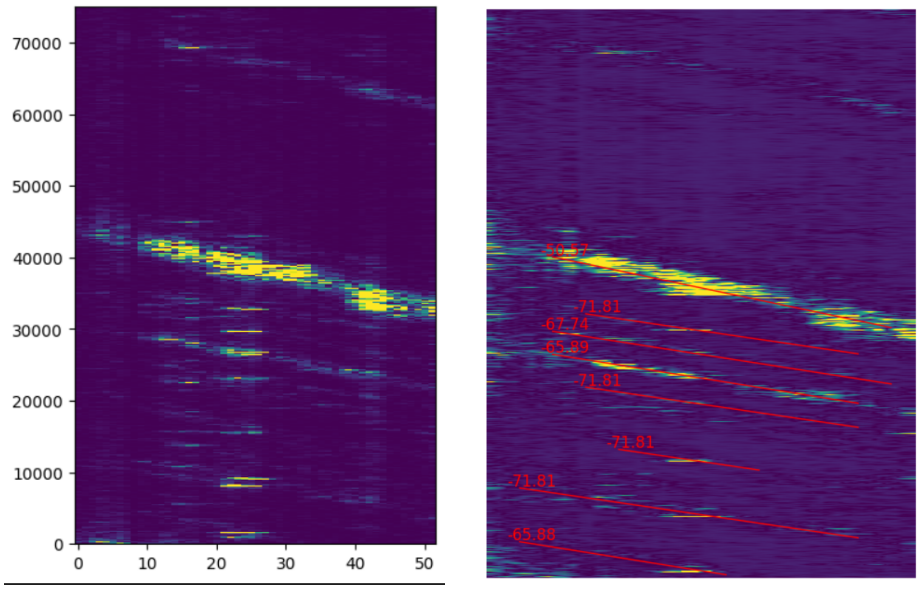

Figure 5: Random file