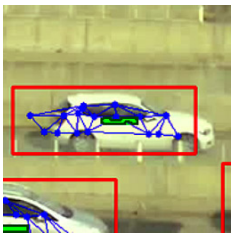
Real-time car tracking and counting



Description	Tracking and counting cars for automatic traffic statistics
Period	Summer 2014
Languages & Libs	Matlab
Tags	Computer Vision

From my office window I've got an unblocked size-view to the Ring Road I (Kehä I) in Espoo, Finland. It is one of the busiest roads in Finland, having up-to 100.000 cars / day. I wanted to create a program which would receive a video feed from a webcam and would process images in real time on common hardware.

Object tracking is fairly well studied problem already, but I wanted to take advantage of the special nature of this problem. Namely cars are known to move in a purely horizontal direction, and I didn't want to have complex background learning and separation code. I also didn't need to know cars locations precisely, but the main interest was on the number of cars and their velocities.

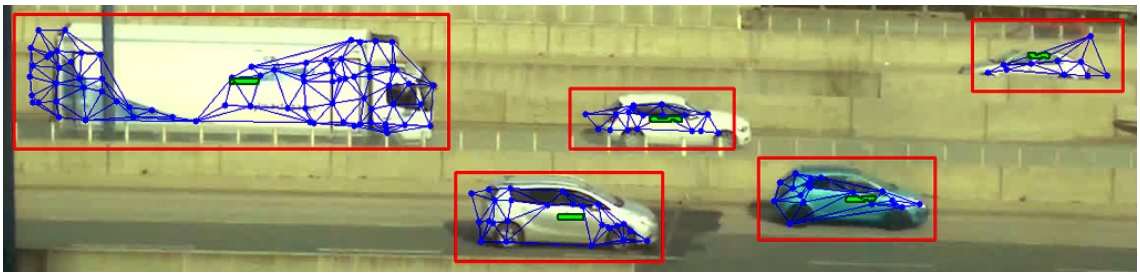


Figure 1: An example result of detecting and tracking cars from a side view. Green vertical lines indicate cars' estimated movement between adjacent frames.

The final output of the algorithm is visualized in Figure 1. Video frames are converted to gray-scale and high-pass filtered (the result can be seen in Figure 3). Then adjacent frames are subtracted from each other, and this is used to detect image locations which had significant changes in their brightnesses. These points are then used to generate a Delaunay triangulation. Too big triangles are removed from the mesh, and resulting disconnected sub-graphs would ideally cover a single car.

This process can merge cars together if the distance between them is too small, as seen in Figure 2, Luckily this problem can be mitigated in data post-processing via standard statistical methods and tracker stability analysis. Additionally too big regions can be ignored all-together. Points and graphs are plotted in blue in Figure 1, and red boxes bound separate "regions of interest" (ROIs). These are then individually analyzed.

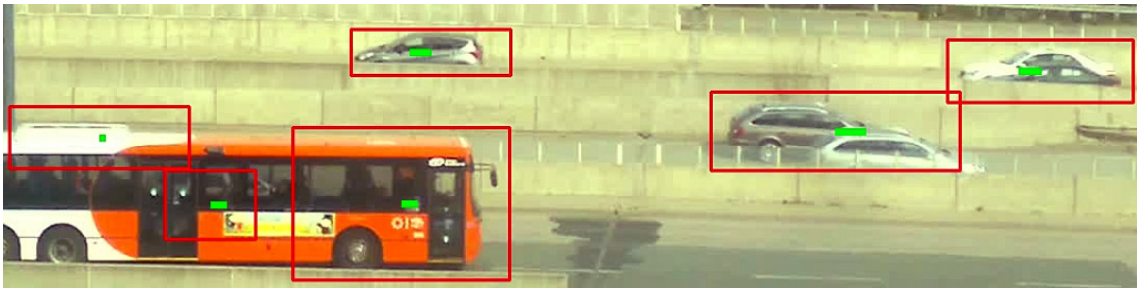


Figure 2: A problematic video frame with a bus counted as three separate cars, and a few cars are merged together. The bus problem wouldn't occur with a better background separation algorithm.

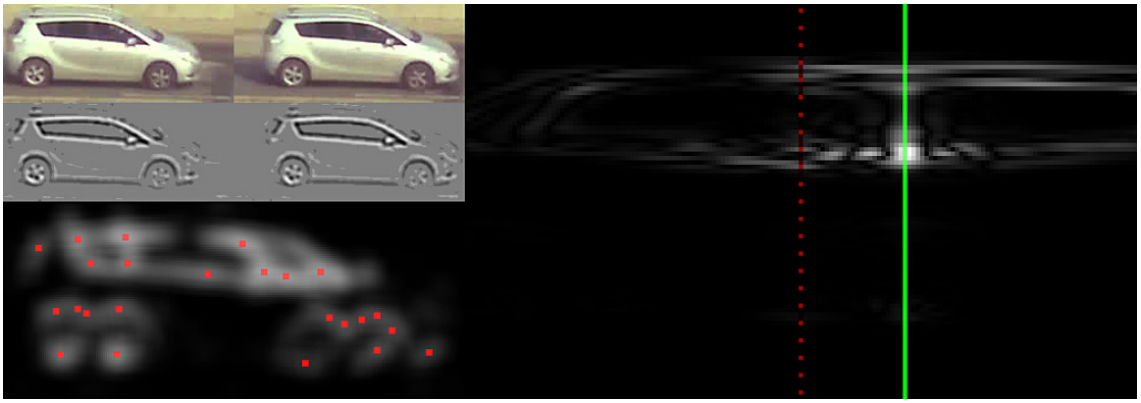


Figure 3: A single car detection and velocity measurement example. The difference between frames is visualized in lower left corner, and red points indicate local difference maxima. The image on the right visualizes the cross-correlation for different image rows and offsets, mid-point is indicated in red and the maximum correlation is indicated by the green line.

Region of interest analysis steps are shown in Figure 3. Once a ROI boundaries are determined, the corresponding region is extracted from current and previous video frames. Previously generated images (converted to gray-scale and high-pass filtered) are re-used. Then the translation between these frames is efficiently, accurately and robustly determined by applying the Phase correlation method in 1D. The Fourier transform is calculated both images rows, these are multiplied together element-wise and the inverse Fourier transform is calculated. The outcome of this is shown in the right side of Figure 3.

To determine the translation amount between images, row-wise phase correlations are multiplied together, and its maximum value is determined. The location of the maximum directly determines the amount and direction of the translation. This signal is shown in Figure 4, and it has a single very distinct peak. This method doesn't rely on any interest point extraction, it isn't fooled by any partial repetitive patterns and no voting scheme is needed for robust outcome. Interest points were only used to detect and segment individual cars, not to actually track them.

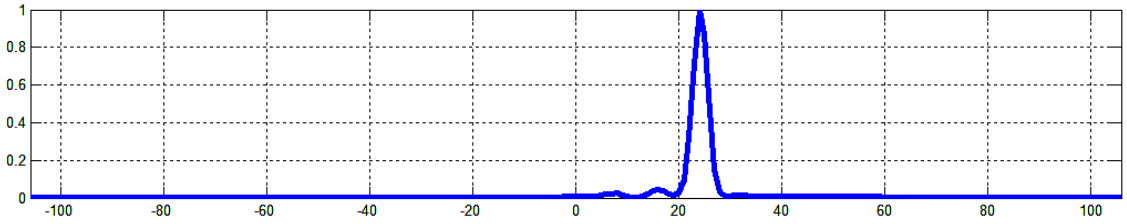


Figure 4: A column-wise summation of correlation values. It is clear that the motion of this car between two frames is about 24 pixels. This could be converted to km/h by a simple calibration.

Overall the system would benefit from a better background model and background separation algorithm, but this simple code worked surprisingly well and it is only about 200 lines of Matlab code. It isn't 100% accurate, but most common errors can be detected and corrected by analyzing its output across multiple frames to detect outliers and incorrect detections. If the traffic gets heavy and car speeds drop, then this method would see it as an empty road since there is no movement. Also this issue would be fixed by better background separation code.