# Speed estimation for vehicles using SVM classification and edge detection

Mathias Dierickx, Tim Ranson, Maarten Tindemans, and Bjorn Vandenbussche

Ghent University, Faculty of Engineering and Architecture
Valentin Vaerwyckweg 1, 9000, Ghent

**Abstract.** This is the abstract: TODO

**Keywords:** SVM classification, Local Binary Patterns, line detection

## 1 Introduction

TODO

## 2 Background

### 2.1 Texture analysis using Local Binary Patterns

Local Binary Patterns (LBP) [6] is a non-parametric descriptor whose aim is to efficiently summarize the local structures of images. As a non-parametric method, LBP summarizes local structures of images efficiently by comparing each pixel with its neighboring pixels. The most important properties of LBP are its tolerance regarding monotonic illumination changes and its computational simplicity. LBP was originally proposed for texture analysis [7], and has proved a simple yet powerful approach to describe local structures.

The original LBP operator labels the pixels of an image with decimal numbers, called Local Binary Patterns or LBP codes, which encode the local structure around each pixel (see figure 1). Each pixel is then compared with its eight neighbors in a 3x3 neighborhood by subtracting the center pixel value. The resulting strictly negative values are encoded with 0 and the others with 1. A binary number is obtained by concatenating all these binary codes in a clockwise direction starting from the top-left one and its corresponding decimal value is used for labeling. The derived binary numbers are referred to as Local Binary Patterns or LBP codes.

One limitation of the basic LBP operator is that its small 3x3 neighborhood cannot capture dominant features with large scale structures. To deal with the texture at different scales, the operator was later generalized to use neighborhoods of different sizes [8]. The histogram of LBP labels can then be exploited as a texture descriptor.

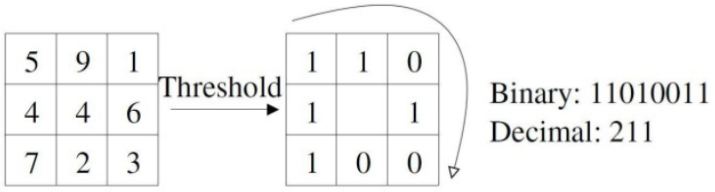



Fig. 1: An example of the basic LBP operator

### 2.2 SVM classification

Support Vector Machines (SVM's) [3] are a learning method used for binary classification. The idea is to find a hyperplane which separates the $d$-dimensional data perfectly into its two classes. However, since example data is often not linearly separable, SVM's introduce the notion of a "kernel induced feature space" which casts the data into a higher dimensional space where the data is separable. Typically, casting into such a space would cause problems computationally, and with overfitting. The key insight used in SVM's is that the higher-dimensional space doesn't need to be dealt with directly, which eliminates the above concerns.

Basically, we want the hyperplane that maximizes the geometric distance to the closest data points (as shown in figure 2).

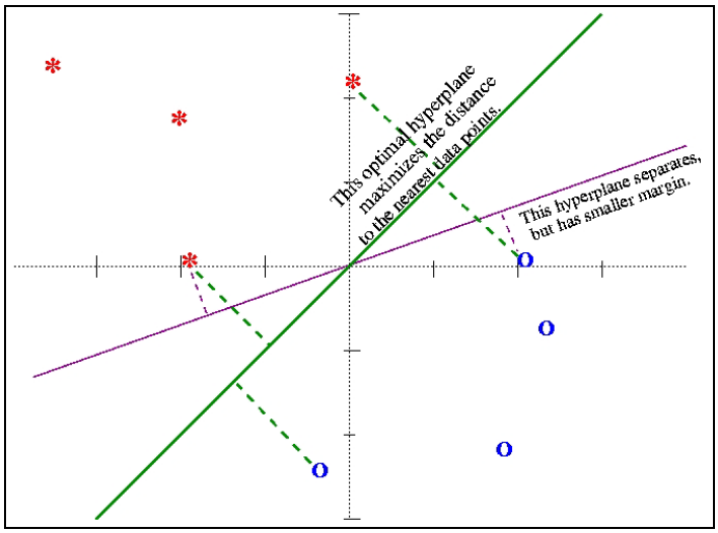



Fig. 2: Choosing the hyperplane that maximizes the margin.

Only the closest data points contribute to deciding the optimal hyperplane, which is why these are called the *support vectors*. They are the only ones needed in defining (and finding) the optimal hyperplane.

The equation for the hyperplane is subject to a constraint concerning the parameter $C$. One can think of $C$ as a tunable parameter: higher $C$ corresponds to more importance on classifying all the training data correctly, lower $C$ results in a more "flexible" hyperplane that tries to minimize the margin error for each example.

*Kernels* The original equation for the hyperplane to linearly separate the data contains a linear kernel. [5] points out when the use of a simple linear kernel is appropriate: if the number of features is large, one may not need to map data to a higher dimensional space. That is, the nonlinear mapping does not improve the performance. Using the linear kernel is good enough, and one only searches for the parameter $C$.

We mentioned that SVM's can use a "kernel induced feature space" which casts the data into a higher dimensional space where the data is separable, in case the data is not linearly separable in its original space. Pending on the type of data and application of the SVM, there are several types of kernels which could be used. The Radial Basis Function (RBF) kernel is a often a reasonable first choice, because it can handle the case when the relation between class labels and attributes is non-linear, while only adding one more parameter $\gamma$ to search for.

*Parameters* There are two parameters for an RBF kernel: $C$ and $\gamma$. It is not known beforehand which $C$ and $\gamma$ are best for a given problem; consequently some kind of model selection (parameter search) must be done. The goal is to identify good ($C$, $\gamma$) so that the classifier can accurately predict unknown data (i.e. testing data).

One method for comparing results for a set of parameter values is cross-validation. In $v$-fold cross-validation, we first divide the training set into $v$ subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified.

### 2.3 Precision, recall, F-scores

TODO: Bjorn

### 2.4 Line detection

TODO: Mathias

# 3 Method

In order to determine the maximum speed of a vehicle two image processing techniques are used: SVM classification and edge detection. We first try to rely on edge detection to find the contours of the lanes where driving is permitted (i.e. where no obstacles are found). We estimate the reliability of the edge detection and incorporate road detection using the SVM classifier when needed.

For the implementation of our method, we use OpenCV 3.1 [4] and C++. Four different datasets were used to either train or test the solution. To train out SVM classifier, we used a function of the OpenCV SVM library which selects the optimal parameters using cross-validation.

## 3.1 Parameters for SVM classification

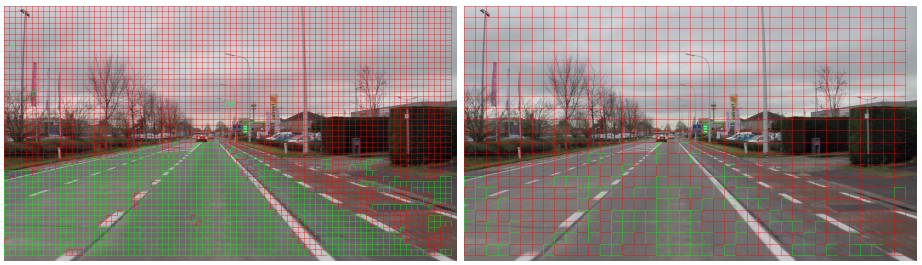
The goal of this classifier is to determine what pixels are part of the road and which are not. We will describe multiple tunable parameters in order to get optimal classification for road detection. These parameters determine both the length and values of the feature vectors, which are then fed to the SVM classifier. We compare classification results varying these parameters for the feature vector in section 4.1.

**Block size** Each frame is split into multiple blocks of predetermined size. We chose to vary the block sizes between 8x8px, 16x16px and 32x32px. On the one hand, larger block sizes would too easily contain multiple structures of different classes, resulting in poorer classification. For example a block on the edge of the road containing part of the street and grass. On the other hand, smaller block sizes would omit too many details to be descriptive enough to use for classification.

The block size affects the level of detail taken into account while calculating the feature vector, as the values for the feature vector operate within one block size. Some examples of different block sizes can be found in figure 3.

**LBP** With LBP we try to characterize the structure of roads within a block. The histogram of LBP values within that block can be added to the feature vector in order to improve classification. These histograms are calculated per channel (e.g. 3 channels for RGB) and are added consecutively to the feature vector.

**Color** Because working with LBP values only doesn't take color intensity into account [9], histogram values of color intensity per block and channel are added to the feature vector.



(a) Block size 16x16px (b) Block size 32x32px

Fig. 3: SVM prediction using only LBP values and including road marks. Green blocks were predicted to be road, red to be non-road.

### 3.2 Kernels for SVM classification

Judging from the OpenCV documentation[1] there are two SVM kernels fit for a two-class classification problem. A linear kernel is faster, but the RBF kernel will outperform the linear kernel in case the data is not linearly separable. We tested and compared the results using each kernel to decide which is best fit for our classification problem.

### 3.3 Training the SVM classifier

For each combination of the previously mentioned parameters, the SVM classifier must be trained. For a training set (a subset of the datasets available) each frame is divided into blocks. Each block must be manually labelled with the expected value for classification. For practical reasons we put a margin between each block in the frames of the training set and only used the $i$-th frame of each dataset (i=0,10,...,40). Figure 4 shows a visualization of pre-labeled blocks in a frame. The classifier is trained using the label and calculated feature vector for each block in each frame of the training set.

### 3.4 Road marks

The training dataset was labelled considering blocks containing road marks once as part of the road class and once as part of the non-road class. Initially road marks were labelled as non-road, in order to get a good classification of road (i.e. different types of asphalt). As these road marks aren't really considered an obstacle for a car driver, ideally they should be classified as road. This makes up for a more difficult classification problem. To quantify this effect we apply classification both with and without road marks.

[1] http://docs.opencv.org/3.1.0/

Fig. 4: Visualization of a pre-labelled frame of a training set. Block size 32x32px, excluding road marks

### 3.5 Removing details

Detecting edges is very straightforward using an edge detection algorithm like Canny Edge Detection [1]. However, there are a lot of edges on the road itself, thinking of road markings and cobbles. The edges on the road, considered as noise, and the edges of the road are not unambiguously separable. The width of the patterns that cause the edges on the road is rather small. Using morphological image processing operations, it is possible to remove these small patterns. In figure 5b, a smoothed cobbled road is shown.

(a) Original frame (b) Eroded cobbled road

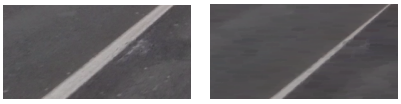
Fig. 5: Perform eroding to remove the edges on the road.

Eroding is a morphology operator to make the objects on the foreground, which are the brightest, smaller [2]. Figure 7b shows an example of a thinned white line. When the kernel is large enough, white lines can be filtered out completely. The same technique is used to smooth the surface of cobbled roads. A cobblestone consist of a bright center, surrounded by darker joints. By eroding the cobbled road, the erode function will minimize the center of the cobble, which will cause enlarged joints. With a large kernel, the joints will spread until they overlap. The best results were achieved with an ellipsoidal kernel of 100 x 30 px.

Unfortunately, the erode function will cause the darker objects to expand. The original size of the objects has been modified, as seen in 8b. This will falsify

(a) Original frame (b) Bigger joints after eroding

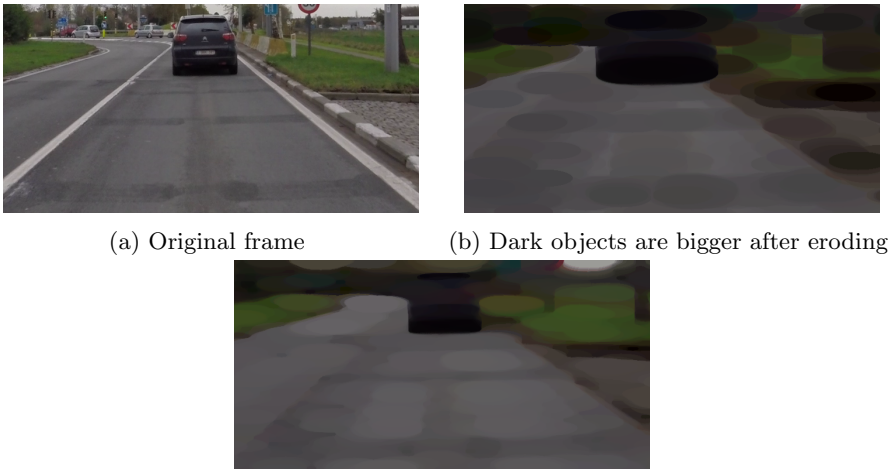
Fig. 6: The effect of eroding cobbled roads.



(a) Original frame (b) Smaller lines after eroding

Fig. 7: The effect of eroding road markings.

the eventual edge detection. In this example, the observed car is reported to close. In order to restore the original measures, dilation is executed. This is the opposite morphology operator of eroding. The combination of eroding and dilation is called opening. When dilation is executed with the same kernel size of the erode function, the original size of the dark objects will decrease to their original size, as seen in 8c. Note that, after eroding, some details of the shape of the objects are lost. In order to provide a safety margin, the dilate function is executed with a slightly smaller kernel.



(a) Original frame (b) Dark objects are bigger after eroding

(c) Restore original sizes with dilation

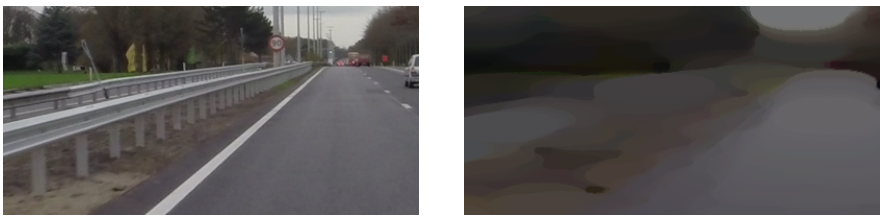
Fig. 8: Perform eroding and dilation to remove details.

### 3.6 Canny Edge Detection

To detect the edges of the road, the Canny Edge Detection algorithm is used. Figure 9 shows an example.

Fig. 9: Canny edge detection

This algorithm requires a minimum and a maximum threshold value, which were determined experimentally. The best results were achieved with a maximum threshold that is twice the minimum threshold. The minimum threshold is determined by the required minimum sensitivity. When the algorithm is configured too sensitive, some imperfections on the road will be detected as edges. On the other hand, a certain level of sensitivity is necessary to detect less clear road edges. For example, a transition from road to dirt has a very small distinction of color and brightness, as seen in 10b. The minimum threshold should be high enough to detect this transition as edge.



(a) Original frame (b) Eroded road

Fig. 10: Unclear transition from road to non-road.

### 3.7 Zebra crossings

This last combination based on edge detection is able to cope with zebra crossings. This is shown in figure 16. In the first one the road markings are small because the car is far enough. Edge detection is therefore able to filter out these markings. In the next frame the markings are closer to the care and are too large to be filtered out. Because the road classification suggests the markings are road, they are considered road.


(a) Zebra crossing filtered out by edge detection.

(b) Zebra crossing classified as road.

Fig. 11: Zebra crossing

### 3.8 Combination of classification and edge detection

The combination of road classification and edge detection starts with removing outliers from the road classification. The goal of this is to eliminate the effect an outlier would have when it's close to the car. A non-road block is considered an outlier when it has less than 4 non-road block neighbours. This is repeated until we become a stable condition without outliers.

Next, the edges are checked if they aren't on the road. This is done by checking that an edge is on a non-road block of the road classification. For safety reasons there is an additional check. The above, left and right blocks of an edge should also be considered as non-road.

When the edge is not classified on the road, it is added to a collection. The intersections of the edges in the collection and the masks are computed. For each frame the masks give the projected trajectory of the car. The edges are visualised as blue lines on figure 16. The intersections are shown green. The closest intersection to the car is shown in red. At this point the maximum speed is derived from the intensity.

## 4 Results

### 4.1 SVM classification

We iterated through several parameter values (as discussed in 3.1) to find optimal classification for road detection. For each combination of parameters we train and test on different combinations of datasets. For each combination we calculate the F-scores. These scores are used to compare the classification results (higher F-scores indicate better classification).

Of the four datasets, different combinations of training sets were selected (each time shown as x-axis on the graphs). The trained classifier was then tested on the remaining datasets.

**Kernels** As previously mentioned in section 3.2, there are two SVM kernels we can use for this classification problem. On average, we tend to get a nearly 11% increase in F-scores using RBF kernel instead of linear kernel. It is safe to say the RBF kernel outperforms the linear kernel for this classification problem in all of our test scenarios.

**Block size** In general, block sizes 16x16px and 32x32px tend to show the better results. As an example we present the F-scores for LBP and color with road marks in figure 12. Other combinations follow the same trend. For the goal of this application, a smaller block size will be more practical (i.e. better accuracy). This is why prefer the block size of 16x16px.

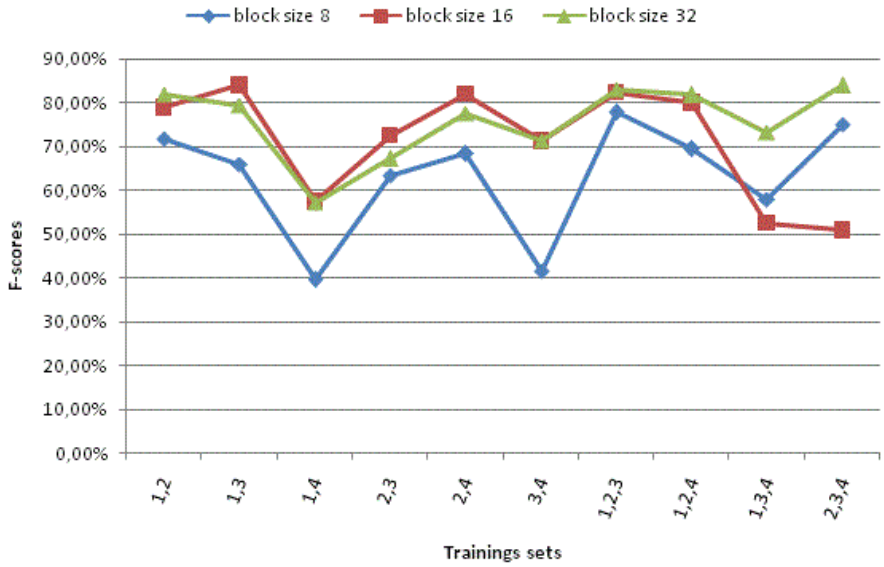



Fig. 12: F-scores for LBP and color with road marks using RBF kernel

**LBP** The use of LBP is crucial to our implementation in order to get a viable classifier for road detection. If we compare results (block size 16x16px) of feature vectors on LBP only to the combination of LBP and color, we get different results depending on whether or not we include road marks.

If we do not include road marks in the training data, we get slightly better results when combining LBP and color for the feature vectors (as shown in figure 13). If on the other hand we do include these road marks, using only LBP shows equal or better results than the combination of LBP and color (as shown in figure 14).

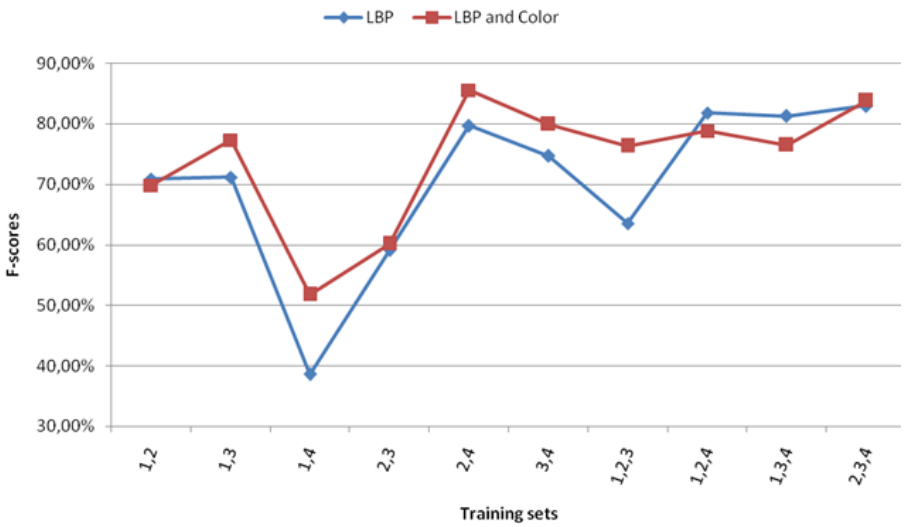



Fig. 13: Comparision of F-scores for 16x16px block size without road marks using RBF kernel

A possible explanation for this phenomenon is that when only asphalt is considered to be road, color is a much more consistent factor to weigh in to the classification. However when including road marks as part of the road, color intensity is no longer an added value and even tends to mislead the classifier.

Overall we can conclude that using only LBP with road marks shows the optimal results for our classification problem.

Considering the results of the different scenarios, we can conclude that training on dataset four results in poorer classification overall. This is because another type of road is used in this dataset, conflicting with the more common road type present in the other datasets. This distorts the classifier increases misclassifications.

**TODO** check if list not f***ed up.
In retrospect, the optimal parameters for our classification problem are:

- RBF kernel
- Block size 16x16px
- Using only LBP histogram values for the feature vectors
- Including road marks as part of the road

### 4.2 Driving results

To combine road classification and edge detection we should first look at the performance of these two methods separately. Therefore, the evaluation was run on each of these two methods. The result is shown in figure 15. The first bar

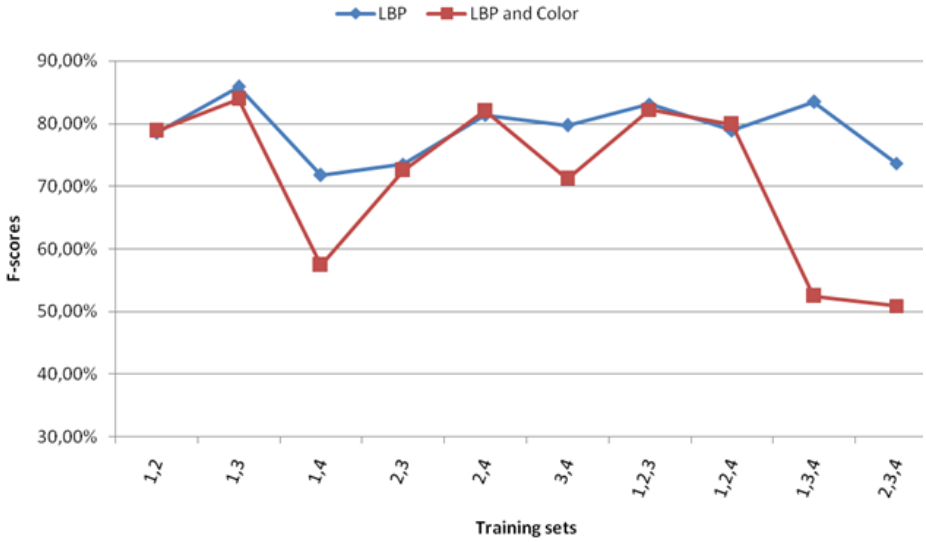



Fig. 14: Comparision of F-scores for 16x16px block size with road marks using RBF kernel

of the histogram shows for every dataset the performance of road classification. Because of a crash in the first dataset there can only be travelled 317m. The second bar shows for each dataset the distance travelled only relying on edge detection. There are no crashes in the four datasets so the results are better.

A first and very safe way to combine these two methods is the following. A region is considered road only if it's identified as road by the road classification and there is no edge. The performance of this combination is shown in the third bar. As said this is very safe, therefore the travelled distance is lower than only applying edge detection.

An other way to make a combination, witch still results in zero crashes is to focus on edge detection. A region is considered road if there are no edges. When there are edges, road classification is used to decide whether it's road. This strategy gave us the best results and are shown as the fourth bar.

This last combination based on edge detection is able to cope with zebra crossings. This is shown in figure 16. In the first one the road markings are small because the car is far enough. Edge detection is therefore able to filter out these markings. In the next frame the markings are closer to the care and are too large to be filtered out. Because the road classification suggests the markings are road, they are considered road.

## 5 Conclusion

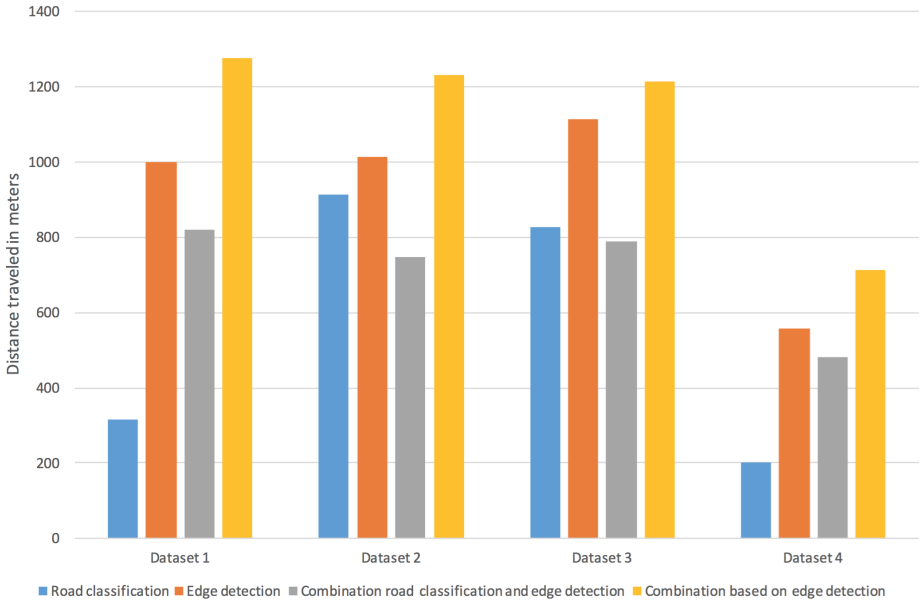



Fig. 15: Results of road classification, edge detection and combinations.

## References

1. Canny edge detection OpenCV 3.1.0 documentation.
2. Eroding and dilating OpenCV 3.1.0 documentation.
3. Dustin Boswell. Introduction to Support Vector Machines. 2002.
4. G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
5. Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. 2016.
6. Di Huang, Caifeng Shan, Mohsen Ardabilian, Yunhong Wang, and Liming Chen. Local Binary Patterns and Its Application to Facial Image Analysis: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6):765–781, nov 2011.
7. Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture analysis algorithms in textile inspection applications. *Pattern recognition*, 29(1):51–59, 1996.
8. Timo Ojala, Matti Pietikainen, and Topi Mäenpää. Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Trans. Pattern Analysis and Machine Intelligence,*, 24(7):971–987, 2002.
9. M. Pietikainen, T. Maenpaa, and Jaakko Viertola. Color texture classification with color histograms and local binary patterns. *Workshop on Texture Analysis in Machine Vision*, pages 109–112, 2002.