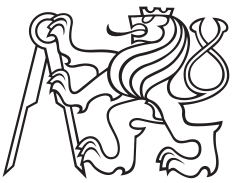
Bachelor thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Car Detection and Tracking from a Vehicle Driving on a Roundabout

Libor Novák

May 2014

Supervisor: prof. Ing. Jiří Matas, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Libor N o v á k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Car Detection and Tracking from a Vehicle Driving on a Roundabout

Guidelines:

1. Familiarize yourself with the problem of detection and tracking of cars from a car driving on a roundabout.
2. Study the WaldBoost and FoT algorithms used in (1).
3. Test and analyze the results of the performance of the algorithms (2).
4. Propose improvements of the algorithms (2) or suggest and implement an alternative approach.
5. Implement and analyze the proposed algorithm.

Bibliography/Sources:

[1] Yoav Freund, and Robert E. Schapire: A short introduction to boosting. Journal - Japanese Society for Artificial Intelligence 14(5), pp.771-780 (1999).(In Japanese, translation by Naoki Abe).
[2] Robert E. Schapire, and Yoram Singer: Improved boosting algorithms using confidence-rated predictions. Machine learning 37.3 (1999): 297-336.
[3] Jan Šochman, and Jiří Matas: Waldboost-Learning for Time Constrained Sequential Detection. Computer Vision and Pattern Recognition, IEEE Computer Society Conference on Vol. 2. IEEE, 2005.
[4] Jiří Trefný, and Jiří Matas: Extended Set of Local Binary Patterns for Rapid Object Detection. Proceedings of the Computer Vision Winter Workshop. 2010.

Bachelor Project Supervisor: prof. Ing. Jiří Matas, Ph.D.

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Libor N o v á k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Detekce a sledování automobilů z automobilu pohybujícím se na kruhovém objezdu

Pokyny pro vypracování:

1. Seznamte se s problémem detekce a sledování ("tracking") automobilů z automobilu pohybujícím se na kruhovém objezdu.
2. Seznamte se s algoritmem WaldBoost a FoT pro řešení problému (1).
3. Vyhodnoťte a analyzujte algoritmy (2) na několika datových sadách.
4. Navrhněte vylepšení algoritmu (2) nebo navrhněte a implementujte alternativní algoritmus.
5. Vylepšení implementujte a vyhodnoťte.

Seznam odborné literatury:

[1] Yoav Freund, and Robert E. Schapire: A short introduction to boosting. Journal - Japanese Society for Artificial Intelligence 14(5), pp.771-780 (1999).(In Japanese, translation by Naoki Abe).
[2] Robert E. Schapire, and Yoram Singer: Improved boosting algorithms using confidence-rated predictions. Machine learning 37.3 (1999): 297-336.
[3] Jan Šochman, and Jiří Matas: Waldboost-Learning for Time Constrained Sequential Detection. Computer Vision and Pattern Recognition, IEEE Computer Society Conference on. Vol. 2. IEEE, 2005.
[4] Jiří Trefný, and Jiří Matas: Extended Set of Local Binary Patterns for Rapid Object Detection. Proceedings of the Computer Vision Winter Workshop. 2010.

Vedoucí bakalářské práce: prof. Ing. Jiří Matas, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Acknowledgement / Prohlášení

I would like to thank my supervisor for many great ideas and inspirational brainstormings. Also, I would like to express the greatest thanks to my family. Thanks to them, I was able to stay focused on the thesis and keep going. Moreover, I would like to thank my girlfriend Hana for moral and technical support during the whole period of writing.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2014

.....

Anotace / Abstract

Tato bakalářská práce se zabývá detekcí automobilů v obrázcích. Situace na kruhovém objezdu byla vybrána, protože je v mnoha směrech výjimečná a dosud na ní dosud nebyla provedena žádná zkoumání. Již vyvinutý WaldBoost detektor je testován proti nově navrženému a implementovanému detektoru, založeném na náhodném lesu, na nové databázi, která byla nasbírána na kruhových objezdech. Tyto testy porovnávají výkonnost a rychlost několika verzí detektoru používající náhodný les a WaldBoost detektoru. Aby se urychlilo procházení obrázku, je představena nová metoda prořezávání detekčních oken, založená na detekci hranových segmentů.

Klíčová slova: detekce automobilů, klasifikace, AdaBoost, WaldBoost, vyhledávací stromy, náhodný les.

Překlad titulu: Detekce a sledování automobilů z automobilu pohybujícím se na kruhovém objezdu

This thesis investigates the problem of car detection in images. The roundabout situation in particular was chosen for the presence of cars at multiple orientations and because it has been neglected in the literature. An already developed WaldBoost detector is tested against a proposed randomized forest detector on a novel data set created on roundabouts. The performed tests compare several versions of the randomized forest detector and the WaldBoost detector in terms of performance and speed. To speed up the scanning window process of the proposed detector, a new method of window pruning based on a line segment detector is presented.

Keywords: car detection, classification, AdaBoost, WaldBoost, search trees, randomized forests.

Contents /

Chapter 1
Introduction

Car detection plays a very important role in surveillance systems, traffic monitoring, car security or support systems in autonomous cars. The surveillance and traffic monitoring systems find use in many fields such as traffic prediction, statistics, navigation or drivers' notification. However, car security systems and autonomous cars have or will have much higher impact on our driving experience. The influence is obvious. Car detection could improve security systems by providing crucial information about surrounding cars in such advance that there would still be time to notify the driver or, in the case of autonomous cars, react. In autonomous cars, these systems are directly connected to the cars' control systems, therefore providing real-time description of the scene.

Car detection from solitary images or image sequences (Figure 1.1) is a very complex problem. As far as surveillance and traffic monitoring is concerned, the task usually is to locate cars in images or video sequences taken from a hard-mounted camera. By assuming the camera is still, the detection algorithms can search for dissimilarities in consecutive images. Whereas in security systems mounted in cars and autonomous cars, the situation is more complicated, given the fact the cars and, therefore, the cameras are moving.

Figure 1.1. The KITTI dataset. Sample images.

Many teams around the world are trying their best to propose a successful solution. There is no time-proven "correct" approach to this problem, unlike for example in the case of face detection. Nowadays, there exist several detectors that show excellent detection results on single-scale side-view and frontal or rear-view cars ([2, 6, 10, 16, 21–22]). The results are usually not as good for multi-scale car detection, even so, the performance is still decent. However, detectors' performance drops when it comes to multi-view or multi-pose car detection.

The traffic situation on a roundabout (Figure 1.2) is a very good example of multi-view car detection and that is why it was chosen for this thesis. The situations that arise there are largely interesting given the variability of the present views as well as the variety of driving situations. Unlike highways, where there is only overtaking and lane changing.

Now, why should we try to detect cars from images? There are reliable car detectors based on 3D point clouds obtained from laser scanners, which can detect very precisely

Figure 1.2. The Dejvicka dataset. Sample images from a roundabout.

the distance and movements of the surrounding cars. They, however, have several drawbacks, which cameras have not.

The most significant one is price. Cameras, even high-definition ones, are very cheap, whereas the price of a suitable laser scanner is approximately four times the price of a regular car. Another advantage of a camera is that it can see in fact much farther than a laser scanner because its CCD chip has a very high resolution. High definition video with the ratio of 1280x720 has 921,600 pixels, whereas the Velodyne laser scanner[1]) used by Google driverless cars gives 260,000 points per rotation and therefore largely less than a camera facing one way or a set of cameras in multiple directions.

An image can be processed quickly, whereas processing a 3D point cloud is usually computationally costly and, in fact, a very complex problem. It would be efficient to use a camera-based detector to localize regions of interest and then process those regions with a more sophisticated 3D-based detector. Also, since cameras see farther, security and collision avoidance systems would be supplied with information about approaching cars earlier, thus leaving more time to react.

Nevertheless, cameras cannot supply data with all features needed for example for driving a car due to its lack of 3D data. We have no information about the distance to the cars as well as their size. These values can only be estimated. Therefore, a car detection system using only cameras can be only used as a subsystem, for example in collision detection or other security systems, or as a primary filter for some more sophisticated detection algorithm. Nevertheless, its role is still very important and, thus, excellent performance and speed is desired.

1.1 Contributions

We employ the well known randomized trees on the problem of car detection because they showed to be powerful in other related applications, for example [1, 12]. We decided to use Local Binary Patterns as splitting criteria, which are used in the reference WaldBoost detector as well, in order to perform the closest possible comparison of the methods.

As a contribution of this thesis, a novel data set was created that covers the newly examined situations on roundabouts. It contains images extracted from video sequences, which makes it suitable for testing car detection as well as tracking.

We introduce a new method of pruning detection windows based on line segment detection. During detection, we use LSD ([20]) to locate lines in the images. These lines are then used to quickly reject windows, which are unlikely to contain cars, thus speeding up the process of sliding window scanning.

[1]) `http://velodynelidar.com/lidar/hdlproducts/hdl64e.aspx`

1.2 Overview

This thesis starts by pointing out the most relevant works. The works discuss the beginnings of boosting up to presenting WaldBoost as well as interesting concepts of object detection using voting techniques and, especially, sliding window approach. Because of the abundance of different methods, car detection was sought specifically. Also, the advantages of the algorithms using search trees and forests are pinpointed.

In Chapter 3, the three datasets used for evaluation are presented. Two of them were used before and one I collected in Dejvická, Prague and several other roundabouts as the contribution of this thesis. This new dataset, specifically built to meet the topic of this thesis, is the most distinct one because it contains only images extracted from sequences that were shot while driving on roundabouts. It is also the most difficult one since its spectrum of car poses is different from the others. The roundabout situation was chosen because it was not yet studied in any preceding work.

Next, we present the reference multi-view WaldBoost detector developed in Czech Technical University in Prague by Matas and Trefný. This detector is a WaldBoost cascade trained in a way that it can distinguish three different groups of views (car poses). An earlier version is described in [2]. Even though it showed to be reasonably fast and its performance was good, it was not sufficient to be used in real applications.

Therefore, we present the randomized forest car detector, the new proposal of this thesis. We decided to propose a new solution because studying and improving the code of the WaldBoost detector would take many weeks. We wanted to start off with a simpler task, which would be easily extendable. The implementation of the basic algorithm is straightforward and, when designed cleverly, updates can be made easily.

In order to carry out the closest comparison possible to the WaldBoost detector, the same training data as for the WaldBoost detector were used as well as the same image features (LBP features). Starting from the most straightforward implementation, several improvements are suggested and implemented to make a better use of LBP features as splitting criteria in randomized trees.

Next, we propose a new window pruning algorithm. Both detectors use sliding window approach to search for cars in images. We designed a new window pruning algorithm based on a line segment detector, which is able to discard irrelevant windows rapidly. The reason why is that the time to decision of the randomized forest detector is much longer than in the case of a boosted cascade. Thus, the number of windows analyzed by the random forest detector needed to be reduced.

Chapter 5 presents several experiments carried out on the detectors. In order for the evaluation to be as objective as possible non maxima suppression was switched off in the majority of the tests because it can have a significant impact on a detector's performance.

In Chapter 6, the implementation of the proposed detector is explained. Combination of MATLAB and C++ was used as in the case of the WaldBoost detector to make it possible to test the detector and display the results easily.

We decided not to implement and test a tracking algorithm since there is still a lot of room for improving the detector itself. Just a brief description of the tracking algorithm used alongside the WaldBoost detector is given.

Chapter 2
Related Work

The car detection task was born in the end of the past millennium. As one of the pilot projects of car detection can be certainly considered the research of Papageorgiou and Poggio [13] from 1999. They used a fixed size window, which was moved across the scanned image. These windows were fed into a Support Vector Machine (SVM) classifier that decided about its class – car or background. Their SVM classifier was based on detecting intensity differences with Haar wavelets. They pointed out several important things such as that they used wavelets because single pixels can be noisy, using intensity differences is convenient because in that case a bright car on a dark background has the same response as a dark car on a bright background or using real world scenes rather than artificial ones improves the performance. Their tests were made with rear-viewed cars.

Object detection made a huge step earlier, thanks to Freund and Schapire [4] in 1995, when AdaBoost algorithm was introduced. The use of an adaptive boosting algorithm that allowed for stacking up simple features into a complex classifier, while decreasing the training error gave birth to many following projects. A breakthrough with AdaBoost was later made by Viola and Jones [19] in 2001. They showed how to train a rapid object detector using a cascade of simple features chosen by AdaBoost. The key idea was to discard as many background windows as possible in the early stages of a cascade, which implied decreasing the time needed for decision. We will try to use this idea in randomized trees as well. Also, the invention of integral image allowed them for a faster evaluation of Haar features, which showed very good performance on face detection.

Many adaptations of AdaBoost were introduced since then. Nevertheless, for the sake of this thesis, only closely related publications are described. The previously mentioned idea was incorporated by Šochman and Matas [24], where WaldBoost was presented. WaldBoost uses the real valued version (RealBoost) of AdaBoost described by Schapire and Singer in [15]. It provides a theoretical justification to the AdaBoost trained cascade classifier by implementing the Wald's sequential probability ratio test, which is important in describing the tradeoff between classifier's complexity and the time to decision. This version of cascade classifier extended by the domain-partitioning weak hypotheses (also described in [15]) was used by Trefný as the WaldBoost car detector, which is used in this thesis as the reference solution.

The advantage of the use of a cascade classifier is also pointed out by Zhu and Zhao [23]. Even though they use different tests and detectors in the cascade, the advantage is clearly presented on quick discrimination of cars from background by simple tests in the beginning of the cascade. A sophisticated SVM detector is then used for a more detailed analysis. It is actually similar to our proposed approach of using a fast window pruning algorithm as a primary phase of a detector.

So far, I introduced only detectors using sliding window method for object localization. However, many state-of-the-art detectors use voting procedures to find object centroids.

This method was used for example by Gall and Lempitsky [6]. A codebook of image patches is learned together with their positions relative to the object centroid, which is then used for voting about an object centroid during detection. The votes are combined by Hough transform to produce clusters belonging to different objects and, from those, final bounding boxes are derived. With this approach they achieve great results on side-view cars.

Similar method was used by Chum and Zisserman [3], where unsupervisely chosen regions of interest have relative position to the object they represent. Also Shotton et al. [16] use this approach. They incorporate an interesting approach of countour-based features, which precisely describe an object. However, a search for this features and matching is time consuming. Altogether, searching for regions of interest or image patches in the whole image is usually time consuming because there is a need of more sophisticated descriptions of these features. Therefore, in this thesis, sliding window approach is used since real-time performance is desired.

Having that in mind, using a boosted cascade classifier is good option given it's speed and performance. Although this works well for single-view object detection, a special approach is needed for multi-view object detection, where there is a high intra-class pose and illumination variation. Divide-and-conquer strategy is a good option, which was used by Wu and Nevatia in [21]. Usually a man-made division of the training set needs to be done, whereas in [21] a very interesting method of unsupervised clustering is used to separate the training data into clusters. Real AdaBoost is used to train a tree classifier, which is forced to split every time the chosen feature does not have a sufficient discriminating ability. This split is then propagated back to the higher levels of the tree and the weak classifiers are retrained using this split, which, as described, interestingly improves their performance. This method was shown to be powerful for multi-view car detection or single-view car detection and pedestrians, where in all cases it outperformed the latest methods.

The previous divide-and-conquer strategy was improved later by Kuo and Nevatia [10]. They divide the problem into two. First, an unsupervised sub-categorization is performed and then a tree classifier on each cluster is trained. For the sub-categorization they use a histogram of oriented gradients (HOG) descriptor, which is then reduced to only 2 major dimensions. What is remarkable is how they clearly show how these descriptors can be used for training set division to different car poses (rotations).

What we can learn from [10, 21] is that a decision tree is a powerful classifier, which can be used for training set division and, possibly, when suitable features are chosen also to discriminate between different car poses (rotations). That justifies our choice of going with randomized decision trees (forest). As described, they enable to use a wide variety of features and the randomization allows for good generalization, which, in the case of cars, is necessary. Considerable features are HOG, extended set of Local Binary Patterns [17], image strip features [22], which describe the car geometry very well, and others.

The last but not the least important approach to mention is the work of Pepik et al. [14]. They considered a completely different approach on car detection, which, instead of learning the class explicitly, models occlusion patterns such as parked cars in a row or in a parking lot. Their detectors have solid performance in a city, the question is how well they would do outside of town or on a highway. The KITTI dataset was used in this work, which is used in this thesis as well.

Chapter 3
Experimental Data

In this chapter the datasets used are introduced as well as methods for training samples' extraction. First, an explanation of the used notation is explained. Then, a thorough description of the datasets is given followed by a description of extracting positive (car) and negative (background) samples.

3.1 Annotation

The cars in the databases are represented by bounding boxes. I used the same representation as Trefný because it allowed me to easily use the same data as used for WaldBoost training and also the annotating software Clicker[1]) from Eyedea Recognition Ltd.

Bounding box is a rectangle, which marks the boundaries of a car (see Figure 3.1). Four numbers are needed to describe the position of a bounding box. In the database, the x and y coordinate of the top left corner and the x and y coordinate of the bottom right corner are used.

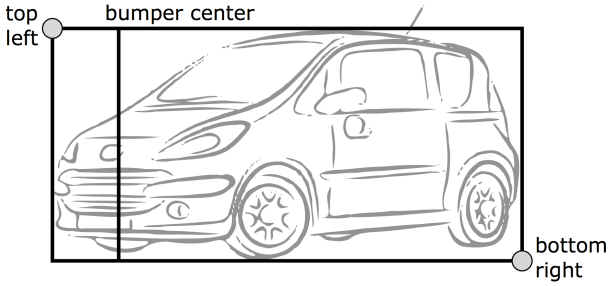


Figure 3.1. Bounding box sketch.

As extra information, the bumper center x coordinate is labelled as well. This is what allowed me to estimate the car rotations in the JT and Dejvicka datasets. The position of the car's bumper can be easily determined from this x coordinate by finding the side of the bounding box that is closer to the bumper center.

The last piece of information is the orientation of the car. It is stored as a flag 1 or 2, representing the frontal and rear views respectively. All this information is stored in a text file, which is then included in each folder containing labelled car samples. The format of each line is `filename:flag left top right bottom bumper_center;...`. An example of such file is shown below.

[1]) The software Clicker serves for annotating various datasets through a web interface. It produces a XML description file for each image, which can be then translated to any other representation.

```
IMG_5433.JPG:1 679.24 524.17 1117.51 928.97 897.91;
IMG_5434.JPG:1 182.09 648.43 714.48 1002.06 366.03;1 782.41 561.64
   1284.36 1010.70 1034.18;
IMG_5435.JPG:1 584.95 476.66 1212.94 1017.61 898.95;
IMG_5442.JPG:1 678.82 501.46 1098.02 897.27 889.62;
```

A different representation is, however, used for the KITTI dataset. I wrote a MatLab script, which translates their representation to the one used here. No information about bumper centers is provided, but it is not a problem because car rotations are labelled explicitly.

What I took from the KITTI dataset was the $\langle -180°, 180° \rangle$ representation of car rotations. The labelled angles correspond to the situations in Figure 3.2. I therefore estimated and translated the angles in the JT and Dejvicka dataset to the same representation. For a thorough explanation see Section 3.4.

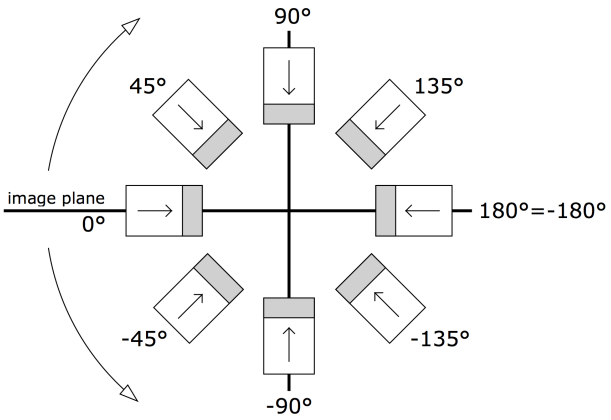


Figure 3.2. Car rotation angles' distribution (top view).

This kind of representation is convenient since it clearly marks whether a car is rotated towards the camera with a plus sign (positive angles) or away from the camera with a minus sign (negative angles).

3.2 Datasets

Three different car datasets are used to evaluate the algorithms in this thesis. The first dataset, which will be referred to as the JT dataset, is a dataset that was used by Trefný to train the WaldBoost detector. The second dataset I collected and labelled myself and will be further called the Dejvicka dataset. The third dataset is a publicly available dataset called KITTI. The description, analysis and comparison of these datasets is provided further in this section.

The first glance to the differences of the datasets provides Table 3.1. It is clear that the newly created Dejvicka dataset is among the most extensive ones. Moreover, the uniqueness of the Dejvicka dataset is obvious also from Figure 3.3. The differences are pointed out in the following sections.

The angles for Figure 3.3a and 3.3b were estimated from the labelled bumper centers because no information about the car rotation was included with the data. The algorithm used for car rotation angle estimation is described in detail in Section 3.4.

	JT	Dejvicka	KITTI_FV	KITTI_OCC
Number of images:	5163	5748	7481	7481
Number of empty images:	5	165	1496	901
Number of cars:	9484	16305	11017	18483

Table 3.1. Comparison of the sizes of the datasets.

The Parzen window estimates in Figure 3.3 were created using normal distribution with zero mean and standard deviation set to 5.

3.2.1 The JT Dataset

The JT dataset is a part of the dataset that was used to train the WaldBoost detector, which is tested here for comparison with the newly proposed randomized forest. It is a private dataset that was created by Trefný by combining images provided by Eyedea, Toyota and several others.

This dataset contains images taken by hand and also from a camera mounted on top of a car, which implies containing images taken from more than one point of view. Some images are parts of video sequences, some are solitary. Images were taken in cities and connecting roads as well. Moreover, various occlusions, lighting conditions and types of cars are present including small trucks. See sample images in Figure 3.4.

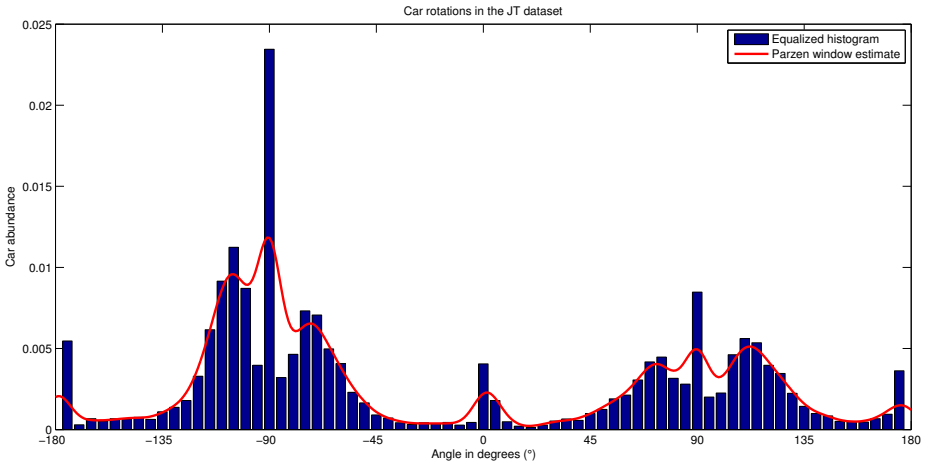
Figure 3.4. The JT dataset. Sample images.

Table 3.2 shows that the JT dataset image count is quite extensive. However, there are less than 2 cars per image, which in comparison with the Dejvicka dataset is not as many. These numbers would correspond to moderate traffic.

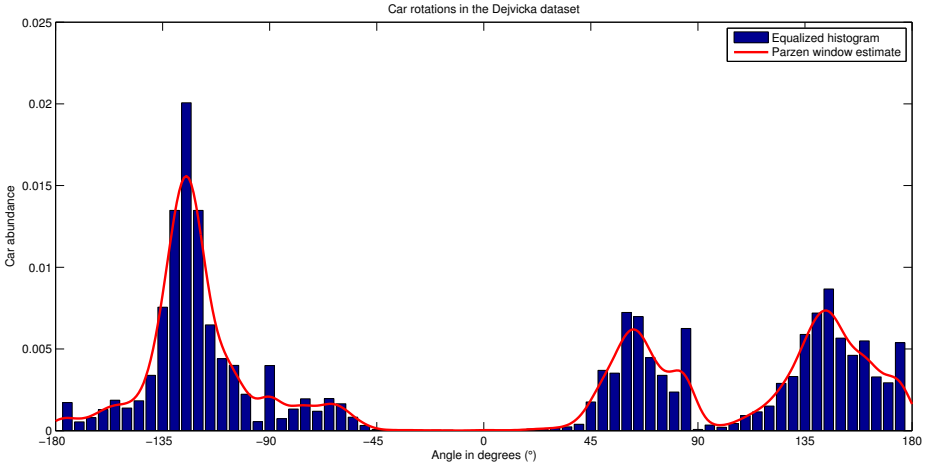
Number of images:	5163
Number of empty images:	5
Number of cars:	9484

Table 3.2. The size of the JT dataset.

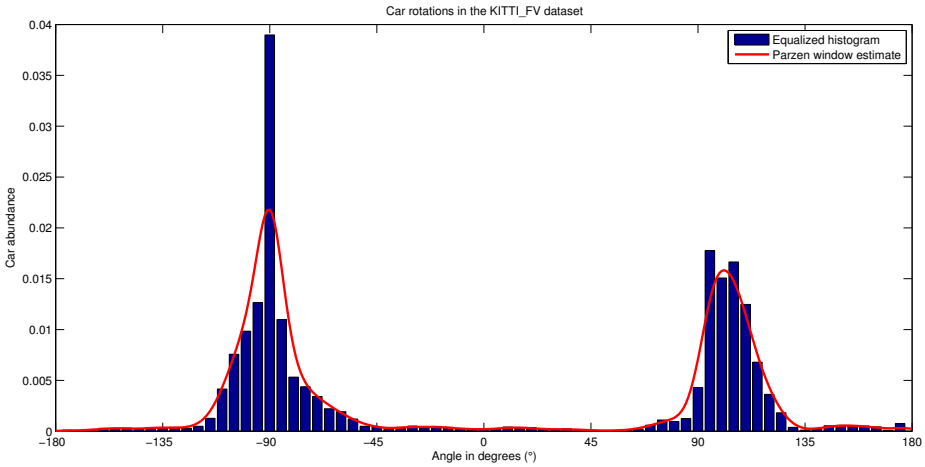
Nevertheless, more interesting statistics are presented in Figure 3.3a. It shows that the variance of car rotations in this dataset is high. All angles are present. The rear views dominate together with frontal and semiside views, however the variance is significant.



a) The JT dataset.



b) The Dejvicka dataset.



c) The KITTI dataset.

Figure 3.3. The distribution of car rotations in the used datasets. The angles correspond to the representation in Figure 3.2. The data for the JT and the Dejvicka datasets were estimated as described in Section 3.4.

This is not true for example for the KITTI dataset. It makes the JT dataset suitable for training.

3.2.2 The Dejvicka Dataset

The Dejvicka dataset consists of sequences of images that I collected on the roundabout in Dejvická, Prague and several roundabouts in Kladno. This dataset was created especially to fit the purpose of this thesis, which is detection of cars on a roundabout.

The images were sampled from a video taken by a camera that was mounted inside of a car behind the windshield on the passenger side. This makes them suitable also for testing tracking. The point of view is the same for all images, which corresponds with the idea of having a fixed camera mounted in a car in the future use. Also, since roundabouts are flat, only a small amount of vertical variance is present in the views. Images contain occlusions, diverse lighting conditions (especially hard when facing the sun) and different car types, see Figure 3.5. However, a smaller number of unique cars is present since the images are extracted from video sequences.

Figure 3.5. The Dejvicka dataset. Sample images.

I labelled the images myself using the Clicker software made by Eyedea Recognition Ltd. Because the database has about 5500 images in total, it was not possible for me to label every single car in every single image individually. Therefore the cars were labelled in approximately every tenth image. Since the images are parts of sequences, it was possible to interpolate between those labels and thus create the labels for the images in the whole sequence.

As we read from Table 3.3, this dataset is more extensive then the JT dataset since there is even higher number of images and there are almost 3 cars per image.

Number of images:	5748
Number of empty images:	165
Number of cars:	16305

Table 3.3. The size of the Dejvicka dataset.

The uniqueness of car detection on a roundabout is obvious from Figure 3.3b. When we compare this figure to Figures 3.3a and 3.3c we see a huge difference in the present car

rotations. When in the JT and the KITTI dataset, there is abundance of frontal and rear views, in the Dejvicka dataset there is a minimum amount of them. The semi-side views predominate here.

It is interesting to notice that there is a lack of cars rotated from $-50°$ to $50°$. This is caused by the fact that we drive on the right hand side, therefore counterclockwise on roundabouts. If this dataset would have been collected in the UK the lack of angles would be on the other side.

The trend here suggest that there needs to be a dataset designed for the roundabout situation specifically if superior results are to be sought.

3.2.3 The KITTI Dataset

KITTI is a publicly available dataset (`http://www.cvlibs.net/datasets/kitti`) created by Karlsruhe Institute of Technology. The images were extracted from video sequences that were taken by a camera mounted on top of a car. It is more challenging then other frequently used datasets like UIUC[1]) or MIT Street Scenes[2]) datasets. Because it is public it can help us compare our algorithms to other submitted solutions.

The ground truth for the KITTI dataset was obtained by hand labeling, supported by laser scanned point clouds, which makes it accurate especially in terms of cars' rotations (positions). There are no images from roundabouts present, however there are many parked cars and mid-sized intersections. The images are very wide because they were stitched from two cameras that were mounted on the top of a car. However, the point of view is the same for all. KITTI also does contain information about pedestrians, cyclists and other object, but I used only the information about cars.

Apart from car positions, the dataset also contains information about 4 levels of occlusion - fully visible, partially occluded, largely occluded and unknown. I created two datasets, where one of them contains only fully visible cars (will be referred to as KITTI_FV) and the other contains unoccluded and partially occluded cars (will be referred to as KITTI_OCC). This allows me to test how much a detector is resistant to occlusions. More information about the KITTI Vision Benchmark Suite can be found in [7] or [8].

The KITTI dataset is in its size similar to the Dejvicka dataset. Even though it has less cars per image (see Table 3.4), the total number of cars is high. The high number of empty images is thanks to the fact that some images contain only pedestrians or bikes.

	KITTI_FV	KITTI_OCC
Number of images:	7481	7481
Number of empty images:	1496	901
Number of cars:	11017	18483

Table 3.4. The size of the KITTI dataset.

When we examine Figure 3.3c, we notice similarities to the JT dataset. Frontal and rear views are in the majority as well as in the JT dataset. This is because when driving

[1]) `http://cogcomp.cs.illinois.edu/Data/Car/`
[2]) `http://cbcl.mit.edu/software-datasets/streetscenes/`

Figure 3.6. KITTI dataset sample images.

up a road, we see mostly cars in the opposite or the same direction. Occasionally we see cars from a side, when we drive through a crossroad. Moreover, this dataset is the most "isolated" one, having the smallest amount of variance of car rotations. This again testifies to the specificity of the Dejvicka dataset and the roundabout car detection problem.

The figure of car rotations for the KITTI_OCC is not shown since it is similar to the KITTI_FV (Figure 3.3c).

3.3 Positive (Car) Training Samples

3.3.1 Definition

Positive training samples are used in a form of a fix-sized grayscale images. However, this is not the case for the bounding boxes in the databases. Therefore, the bounding boxes in the databases need to have their ratios adjusted and be resized in order to be of the same dimensions.

The WaldBoost detector was trained on 26x22-pixel images containing three different groups of poses (see Section 3.3.3 about ALR coefficient for details). Such small images were chosen so that also distant cars were able to be recognized in the images. Nevertheless, from several tests I carried out I concluded that using larger images is advisable since it drastically improves the performance. Therefore I decided to use images of twice these dimensions – 52x44 pixels, having in mind that nowadays, cameras have a higher resolution and thus distant cars would be detected anyway. Also, when we study different works, such as [21, 23], we see the size of the used window is even greater.

3.3.2 Processing

The ratio adjustment process is following. The width w of a bounding box is always kept and the height is changed so the resulting bounding box has the ratio 13x11. After that, the bounding box is enlarged by a margin m in all directions, while the center

point is kept. The situation is depicted in Figure 3.7. The final width w^* and height h^* of the bounding box to be cropped is

$$w^* = mw, \qquad h^* = m\frac{11}{13}w,$$

where margin $m = 1.2$ was used. Such a bounding box is cropped from the original image and then resized to the required dimensions – in my case 52x44 pixels.

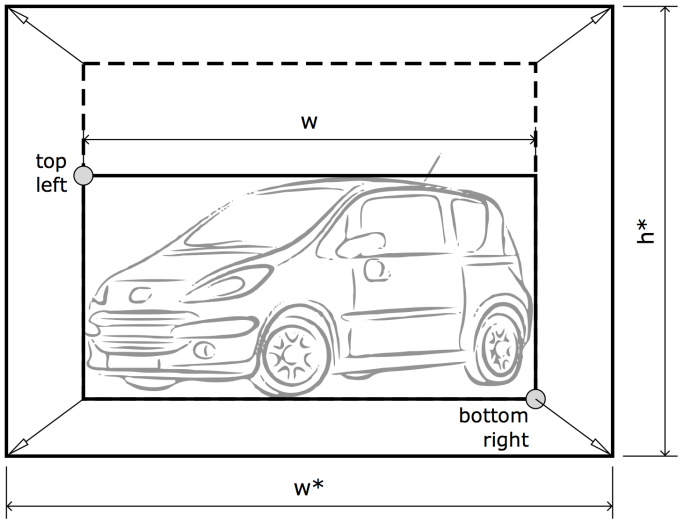


Figure 3.7. Bounding box resizing.

3.3.3 ALR Coefficient

ALR is an abbreviation for absolute value of the ratio of the logarithms of the lengths (J. Trefný, personal communication, November, 2013). The three groups of car poses for WaldBoost training mentioned in Section 4.4 are described by this coefficient. ALR coefficient represents the ratio of the width of the side and the mask of a car. Looking at Figure 3.1, the ALR coefficient can be written as

$$alr = \text{abs}\left(\frac{\log(\|TL_x - BC_x\|)}{\log(\|BR_x - BC_x\|)}\right),$$

where TL stands for top left, BC stands for bumper center and BR stands for bottom right.

The coefficient is a number between 0 and ∞, however the interesting values are between 0 and 2. The greater the value, the wider is the side of the car compared to its bumper width. When a car rotates, the ALR coefficient changes. For a frontal or rear view (f) the ALR coefficient is a small number, for a side view (s) the coefficient is a large number and for a semi-side view (m) the coefficient is a number between the boundaries of the frontal and the side views. This means that there is a need of 2 numbers (boundaries C_1 and C_2) in order to determine the car pose to be extracted from a bounding box:

$$0 \leq \text{f} < C_1 \leq \text{m} < C_2 \leq \text{s} < \infty.$$

This is enough to determine car poses in a training database, but during detection testing, the views very often overlap. This means, for example, that there can be a correct detection of a side view and a semi-side view at the same time. Therefore, the boundaries also have a defined overlap.

3.4 Car Rotation Angle Estimation

In order to compare the present car poses (rotations) in the datasets, there was a need of estimating the car rotations in the JT and Dejvicka datasets because no information about car rotations was included (in KITTI the information about car rotations is provided). Since the car labels in those datasets contain three measurements – the bounding box width, height and the bumper center (see Figure 3.1), it was possible to do so in three different ways. In the end, I combined these three methods to provide a higher accuracy. The three methods use three different ratios:

1. Ratio of the height of a car and the bumper width.
2. Ratio of the height and the length of a car.
3. Ratio of the length of a car and the bumper width.

Their descriptions follow in Sections 3.4.1, 3.4.2 and 3.4.3. This would be useless without assumptions about the average width, height and length of a car. We have to keep in mind that these facts are very place-specific. The average car in Europe is certainly different from the average car in the United States. Therefore, since the data sets used in this project were collected in the Czech Republic, the average car dimensions are determined from the vehicle fleet of the Czech Republic.

The outcome of the analysis is shown in Table 3.5. Studying the most common vehicles of the vehicle fleet, it can be noticed that the length of the cars differ significantly (its standard deviation is high), whereas the width and height vary less. Thus, it is advisable to use these two measurements in the first place, the ratios containing length should have less significance put on.

	Average	Stand. deviation
Car width:	1.7 m	0.07 m
Car height:	1.45 m	0.03 m
Car length:	4.2 m	0.3 m

Table 3.5. The average car dimensions in the Czech Republic. The values were computed from the data available at `http://www.autosap.cz` using eight most common cars in the Czech Republic, which is 33% of the whole vehicle fleet.

I carried out the calculation of the angle in the following steps. First, I computed the angle β of deviation from the direction of view (the direction perpendicular to the image plane) displayed in Figure 3.8. Then, the angle β is translated to the $\langle -180°, 180° \rangle$ representation described in Section 3.1 and Figure 3.2.

3.4.1 Ratio of the Average Car Height and Width

Lets assume that the height of a bounding box always corresponds to the height of the car. This assumption can be made because roundabouts are mostly flat, therefore the height of a car viewed from a camera mounted in a car does not get distorted.

When a real car's height h and width w is known, it is possible to compute their ratio and, knowing the projected height in pixels h_p, it is possible to determine the projected

a) Distortion of the width. b) Distortion of the length.

Figure 3.8. Distortion of car dimensions (top view).

width in pixels w_p. This projected width equals the distorted width in pixels w_{pd} when angle $\beta = 0°$. In other cases, comparing the computed width in pixels w_p to the distorted width in pixels w_{pd} illustrated in Figure 3.8a yields

$$\beta_{hw} = 90° - \operatorname{asin}\frac{w_{pd}}{w_p}, \qquad w_p = \frac{w}{h}h_p.$$

3.4.2 Ratio of the Average Car Height and Length

The principle of this method is identical to the previous one because car's length gets distorted similarly as shown in Figure 3.8b. The deviation angle β is computed

$$\beta_{hl} = 90° - \operatorname{acos}\frac{l_{pd}}{l_p}, \qquad l_p = \frac{l}{h}h_p.$$

3.4.3 Ratio of the Average Car Width and Length

In this method it is not possible to calculate the projected width in pixels w_p and length in pixels l_p explicitly because these measurements do not correspond to the width w and length l of a real car directly. Still, it is possible to use their ratio to compute the angle β. The two equations below hold for the situation depicted in Figure 3.8.

$$\sin(90° - \beta) = \frac{w_{pd}}{w_p} \qquad \cos(90° - \beta) = \frac{l_{pd}}{l_p}$$

Combining these two equations results in

$$\tan(90° - \beta) = \frac{\sin(90° - \beta)}{\cos(90° - \beta)} = \frac{\frac{w_{pd}}{w_p}}{\frac{l_{pd}}{l_p}} = \frac{l_p w_{pd}}{w_p l_{pd}} = K\frac{w_{pd}}{l_{pd}}.$$

The deviation angle from the direction of view β is then computed as

$$\beta_{wl} = 90° - \operatorname{atan}\left(K\frac{w_{pd}}{l_{pd}}\right), \qquad K = \frac{l_p}{w_p} = \frac{l}{w}.$$

3.4.4 Combination and Notation Translation

In the end, the three methods are combined using weighted sum to produce an estimate of the angle β with a higher accuracy.

$$\beta = \frac{3}{5}\beta_{hw} + \frac{1}{5}\beta_{hl} + \frac{1}{5}\beta_{wl}$$

As mentioned before, the first method is the most accurate one, since it is using the average height and width of a car, which vary the least considering the vehicle fleet of the Czech Republic. It makes sense to weigh the method the most and the others less. The weights used for the first, second and third method are 3, 1, 1 respectively.

When the angle β is estimated, it needs to be translated to the $\langle -180°, 180° \rangle$ representation in order to cover the whole range of angles displayed in Figure 3.2. Whether the car has its bumper center closer to the left or right side of a bounding box determines on which side the car bumper is. And, since frontal and rear views are distinguished (labelled) in the database, it is easy to decide in which quadrant the angle falls.

Even though three different computations are used to estimate the rotation angle, one can object to the accuracy of this estimate. It is true that it can differ up to about $\pm 25°$ from the real value, however when all the errors are combined, the statistics say that if they are random, they get mostly cancelled. That is why we can consider Figure 3.8 to be reasonably accurate. Nevertheless, for learning an angle-recognizing classifier it is not ideal.

3.5 Extracting Background Training Samples

Because the training in the case of WaldBoost or randomized trees is discriminative, there is a need of negative (background) samples in order for the algorithm to learn the difference between them and the positive (car) samples.

In the JT dataset, there are special background images included that show anything but cars, see Figure 3.9. Those images were used for negative samples' extraction for training of the WaldBoost detector by Trefný. Nevertheless, from my point of view, these images do not represent the real situation correctly since the trained detector will be used to distinguish between cars and street background images.

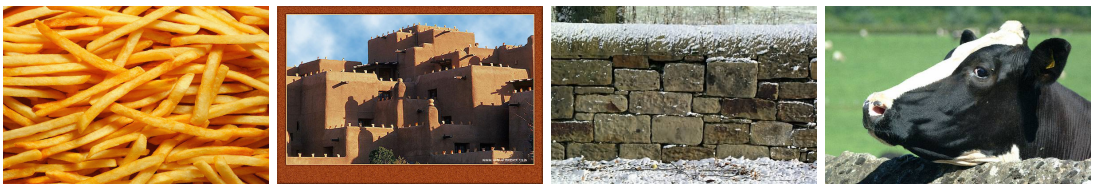


Figure 3.9. JT dataset. Sample images for background extraction.

Therefore, we decided to use randomly generated samples from the same images as the car samples were extracted from. I programmed a script that randomly chooses an area of the required ratio (in my case 52x44) and extracts it as a background sample if the

Figure 3.10. JT dataset. Background images sampled from the car images.

intersection over union ratio with each labelled bounding box is less than 0.1. Sample background images obtained by this method are printed in Figure 3.10.

A test with a randomized forest was carried out, where it was proved that using the new method for generating background is superior to using special set of images to extract background samples from.

But still, even though the performance is better, when we look at the images (Figure 3.10) we see that many of them are sampled as a large area of the original image. Yet, this is not the case during detection, where, when sliding window is used, the small windows predominate. I therefore decided to improve the background samples' extraction to account for this fact.

On each image, all windows that would be scanned using scanning window are generated and several random windows are chosen from this set. This makes the numbers agree. Figure 3.11 shows images extracted using the enhanced method and it is obvious that their nature changed. The test comparing the two new methods can be sought in Section 5.3.3.

The method could certainly be improved even more since during detection a line segment detector LSD (see Section 4.7) is used to prune windows. The same pruning could be used for background training samples before using them for training the detector. I will leave that as a subject of further study.

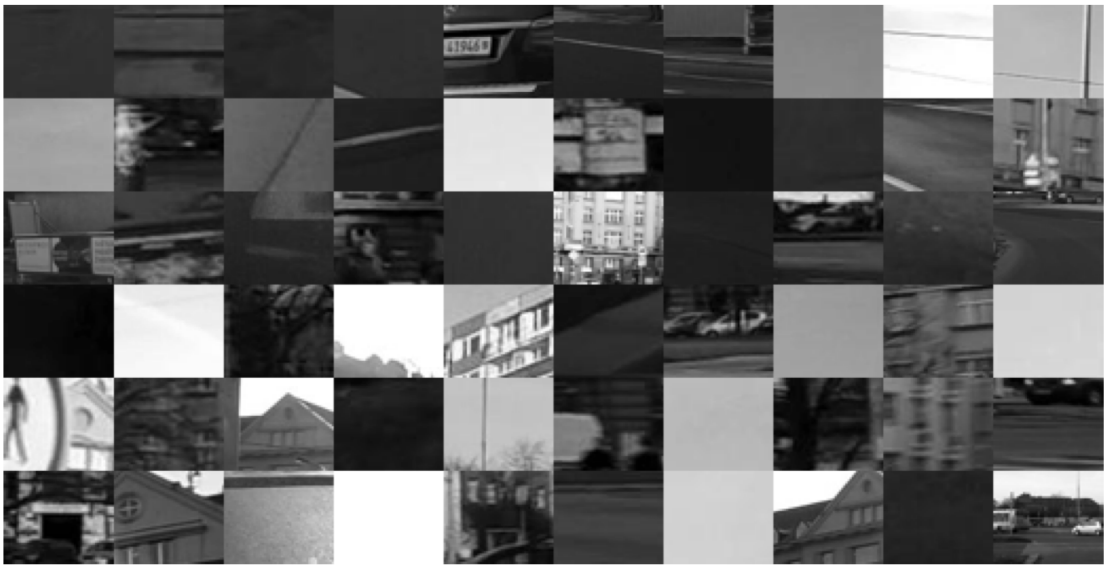


Figure 3.11. JT dataset. Background images sampled using the enhanced method.

Chapter 4
Methods

In this chapter, the methods used are described. First, the difference between detection and classification is pointed out, followed by the explanation of the topic of this thesis – object detection. A short description of the tracking algorithm used in combination with the WaldBoost detector is given. Then, both detectors are presented and explained. In the end, the newly proposed window pruning method based on line segment detection is revealed.

4.1 Detection vs. Classification

In this thesis, it is important to distinguish between a detector and a classifier, that means between detection and classification. Classification is a task, where for an image a class it belongs to is sought. For example, I have a set of images and I want to know which of them show cats and which of them dogs. Whereas in detection, the task is rather different.

Detection does not make any conclusions about the image as a whole. During detection an image is searched for object instances, therefore finding out also their positions comes into play. For example, I have an image and I want to find how many dogs there are and where in the image they are. It is obvious that detection is what this thesis is about.

Despite these differences, when it comes to sliding-window (see Section 4.2.1) detectors, they are closely related. In sliding-window-based detectors, single windows of a fixed size are analyzed. It is decided which class (in this thesis car or background) they belong to, which actually is a classification task. So, the core of a detector using the sliding window approach actually is a classifier.

4.2 Object Detection in General

Detection is the problem of finding object instances, in this thesis in an image. In general, there are two approaches to this problem. One of them is scanning the whole image with a window of fixed dimensions, which is called sliding window (see Section 4.2.1. The other one does not use a window for scanning the whole image, but it detects features, which then vote for the positions of object centroids.

The latter was used in [3, 6, 16]. It is convenient in a way that there is no need for scanning a huge amount of windows, when in reality only hundredths of a percent of windows contain objects. In the usual approach, the image is searched for feature instances. Then, the centroid votes of the found features are collected and grouped by some clustering algorithm. A very often used one is Hough transform. Around the found centroids, the bounding boxes of the objects are estimated.

Nevertheless, there are several drawbacks of this method. A previously mentioned one is the need of complex features in order for them to be discriminative enough. This brings the need of more computational power and time, which, for us, is crucial. Also the implementation of these kinds of methods is more difficult and thus time demanding, which in the case of thesis is not achievable. Possibly, also a deeper knowledge of the subject is needed, which in my case is not sufficient since I am dealing with this kind of task only for the second time in my life.

The former method – sliding window, however, is easier to implement and does not require that much post-processing, only non maxima suppression, described in Section 4.2.2. It was successfully used many times, for example in [4, 19, 21–24], and its good performance is time-proven.

The way the detection is carried out in this case is the following. First, the image is scanned using sliding window and the windows are evaluated using a classification algorithm. The windows containing objects are collected and, finally, grouped using non maxima suppression. Sliding window is described in more detail in Section 4.2.1.

We chose a sliding window approach since it is easier to start with and also we wanted to have the comparison of the new randomized forest detector and the WaldBoost detector, while eliminating the influence of the pre and post-processing algorithms.

4.2.1 Sliding Window

When the core of a detector – a classifier analyzes windows of a fixed size, there needs to be a way of splitting the image the detection is performed on into smaller windows of the dimensions required by the classifier. The method is called sliding window. The basic method works in the following way.

A window with fixed dimensions moves across the image with a given step (for example 2 pixels) and creates many sub-images of the required size. These images are then fed to the classifier. When the whole image is searched, the image is resized (reduced) by a given multiplier and the whole process of moving the window is repeated. This continues until the reduced image is smaller than the dimensions required by the classifier.

When the search is done, what usually happens is that there is more than one detected window on the desired object. Therefore, the windows are pruned using a non maxima suppression algorithm. The version used here is described in Section 4.2.2.

Sufficient is the method for simple tasks. However, for difficult and time constrained tasks this is not fast enough. A real-time performance is desired for car detection. Even though the WaldBoost detector performs only 5 tests on one window in average, real-time performance is still not achieved. Randomized forests perform even more tests per window. This suggests that a pruning algorithm should be used on the widows before any analysis by the classifier is made.

We propose a new method for rapid window pruning based on line segment detection (LSD). It is presented in Section 4.7.

4.2.2 Non Maxima Suppression

When the image is searched by sliding window, many detections appear in the places where cars are present and there is a need of filtering these multiple detections to output the most reliable ones.

A basic non maxima suppression algorithm takes all detections from an image and orders them by confidence. Then, starting from the highest confidence, the detections are taken one after another and when there is a detection that overlaps (different overlap measurements can be used) with a detection with a higher confidence, the detection with lower confidence is discarded.

For details about non maxima suppression implemented in this work see Section 6.4.4.

4.3 Tracking

The tracking algorithm used with the WaldBoost detector is enhanced Flock of Trackers. A more robust version was used in order to track the detected cars more reliably. Unfortunately, time didn't allow me to study this part thoroughly because I was focused more on the problem of car detection. That is why I only give a short description of the basis of the tracking algorithm – Flock of Trackers.

4.3.1 Flock of Trackers

The Flock of Trackers (FoT) estimates the position of the tracked object by combining several independent trackers covering the object. Each independent tracker covers a certain area of the tracked object, such as a part of a grid (see Figure 4.1).

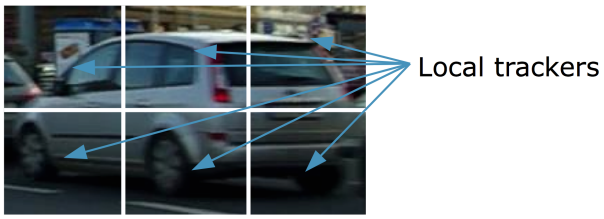


Figure 4.1. Grid placement of local independent trackers.

The local trackers then independently determine the estimated future position of the tracked area. The estimated future position of the tracked object is obtained as the median of a subset of local trackers' responses. A subset is used because the whole set of trackers is preprocessed to eliminate the unreliable trackers.

The pinpointing of unreliable trackers can be performed in many ways, such as Neighborhood consistency or by Markov predictor introduced by Matas and Vojíř [11], which are used in this application.

In the original FoT, the local trackers are restarted in each frame to their initial positions. However, a different approach is used here. The positions of the trackers are maintained as predicted, only the trackers that drift away from the grid too far are restarted. This makes the tracker more robust and outperforms the original version as described in [11].

4.4 The WaldBoost Detector

The WaldBoost car detector has been developed at Czech Technical University in Prague by Matas et al. In this thesis, I use the latest version, which was worked on

by J. Trefný. A description of an earlier version, which works very well on rear-viewed cars, can be found in [2]. Nonetheless, the current version can distinguish between three different groups of car poses. The first group contains frontal and rear views, the second one left and right side views and the third one semi-side views, which are the situations in between the previous two.

Even though it is possible to sum up the detector's principle briefly, in detail, each part is profound. WaldBoost is essentially an AdaBoost trained cascade classifier. Cascade classifier is used to speed up the detection process, because the time constraint in car detection is large, especially when used for security or control in traffic. AdaBoost orders the weak classifiers used in the cascade by their training error (from the most effective to the least effective one), which allows the cascade to quickly reject the false windows and therefore save time during the detection process. The explanation follows in Sections 4.4.1, 4.4.2 and 4.4.3.

The multi-view ability of the detector can be described as having multiple WaldBoost detectors that use the same cascade of features. Each of them, however, uses different weights and thresholds for the features. This is achieved by extending the basic WaldBoost detector to using domain-partitioning weak hypotheses, described in [15]. The features (hypotheses) used in the WaldBoost detector are various adaptations of local binary patterns (LBP), see [17] or Section 4.6 for further details.

In short, the detector works in the following manner. Each input image is screened using the sliding window method. The windows are evaluated by the WaldBoost classifier and a collection of windows that were assigned the car class with corresponding confidences is returned. The collection is filtered using a non maxima suppression algorithm, which groups the detection windows and outputs the final detections.

4.4.1 AdaBoost

Boosting in general is a method, which allows for combining a set of weak classifiers into a stronger one. A weak classifier is usually a fast and simple test that has a discriminative ability between classes slightly better than random. The "slightly better" is enough for the boosting algorithm to produce a strong classifier by combining the decisions of the weak ones. The prove can be sought in [4, 15] or in Section 4.4.2.

AdaBoost is a boosting algorithm introduced by Freund and Schapire [4] in 1995. Since then, many improvements and generalizations have been proposed such as RealBoost, which is used in this application. Whereas AdaBoost outputs only two different values, usually $+1$ or -1, RealBoost is generalized in a way that it outputs any real number. Since AdaBoost was described and analyzed many times before, see Freund and Schapire [5] or Schapire and Singer [15], I will describe only the basics of the algorithm.

Let's have a training set $(x_1, y_1), ..., (x_m, y_m)$, where x_i belongs to a domain or instance space X and label y_i takes one of the values from label space Y. In the basic algorithm, the label space consists of only two values $Y = \{+1, -1\}$. AdaBoost trains a set of weak classifiers H_t with the training set $(x_1, y_1), ..., (x_m, y_m)$ and a distribution D_t over $\{1, ..., m\}$ repeatedly each round $t = 1, ..., T$. Each of these weak classifiers has the form $h_j : X \to \{+1, -1\}$. The weak classifier with the smallest weighted training error

$$\epsilon_j = \sum_{i:h_j(x_i) \neq y_i} D_t(i)$$

Given: $(x_1, y_1), ..., (x_m, y_m)$, where $x_i \in X$, $y_i \in Y = \{+1, -1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, ..., T$:

- Train weak classifiers using the distribution D_t.
- Get the weak classifier $h_t : X \to Y$ with the lowest weighted training error

$$\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i).$$

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  where Z_t is the normalization factor chosen so that D_{t+1} will be a distribution.

Output the final strong classifier:

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

Figure 4.2. The boosting algorithm AdaBoost.

is chosen from each round to form the strong classifier $H(x)$. The whole algorithm is shown in Figure 4.2.

The earlier mentioned version RealBoost differs from the basic algorithm in the output format of the weak classifiers $h_t : X \to \mathbb{R}$ and so the final strong classifier $H(x)$ is

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x), \qquad H(x) = \text{sign}(f(x)).$$

This gives us the opportunity to measure the confidence of a prediction by $|f(x)|$ and that is the reason why RealBoost is used here. The detections need to have confidences in order to be further analyzed by a non maxima suppression algorithm described in Section 4.2.2.

4.4.2 Why Does It Work?

In order to show why AdaBoost works, I will be following Schapire and Singer [15]. Using the notation from the previous section, one of the theorems of Freund and Schapire says that the upper bound on the training error is

$$\frac{1}{m} |\{i : H(x_i) \neq y_i\}| \leq \prod_{t=1}^{T} Z_t. \tag{1}$$

This can be proven in a following way. Using the assumption, that the distribution D_1 is equiprobable and by examining the update rule in Figure 4.2, it is obvious that

$$D_1(i) = 1/m,$$

$$
\begin{aligned}
D_2(i) &= \frac{D_1(i)\exp(-\alpha_1 y_i h_1(x_i))}{Z_1} = \frac{\exp(-\alpha_1 y_i h_1(x_i))}{mZ_1}, \\
D_3(i) &= \frac{D_2(i)\exp(-\alpha_2 y_i h_2(x_i))}{Z_2} = \frac{\exp(-\alpha_1 y_i h_1(x_i)) \cdot \exp(-\alpha_2 y_i h_2(x_i))}{mZ_1 Z_2}, \\
D_4(i) &= \ldots, \\
D_{T+1}(i) &= \frac{\prod_{t=1}^{T} \exp(-\alpha_t y_i h_t(x_i))}{m \prod_{t=1}^{T} Z_t} = \frac{\exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i))}{m \prod_{t=1}^{T} Z_t} = \frac{\exp(-y_i f(x_i))}{m \prod_{t=1}^{T} Z_t}.
\end{aligned} \tag{2}
$$

And, given the fact that when $H(x_i) \neq y_i$, then $y_i f(x_i) \leq 0$, which means that $\exp(-y_i f(x_i)) \geq 1$ and when $H(x_i) = y_i$, then $y_i f(x_i) \geq 0$, meaning that $0 < \exp(-y_i f(x_i)) \leq 1$, the following can be written:

$$
[\![H(x_i) \neq y_i]\!] \leq \exp(-y_i f(x_i)), \tag{3}
$$

where $[\![\pi]\!]$ represents 1 if the predicate π holds and 0 otherwise. In other words the left side of (3) is 1 if the training sample i is incorrectly classified and 0 if it is correctly classified by the classifier H. The equation (3) can be transferred to cover the whole training se into

$$
\sum_{i=1}^{m} [\![H(x_i) \neq y_i]\!] = |\{i : H(x_i) \neq y_i\}| \leq \sum_{i=1}^{m} \exp(-y_i f(x_i)). \tag{4}
$$

Now, plugging in (2) into the right side of (4) and multiplying the whole equation by $1/m$ results in

$$
\frac{1}{m} |\{i : H(x_i) \neq y_i\}| \leq \frac{1}{m} \sum_{i=1}^{m} \left(D_{T+1}(i) \cdot m \prod_{t=1}^{T} Z_t \right) = \prod_{t=1}^{T} Z_t \sum_{i=1}^{m} D_{T+1}(i) = \prod_{t=1}^{T} Z_t. \tag{5}
$$

The reason why $\sum_{i=1}^{m} D_{T+1}(i) = 1$ is that it is defined as a distribution. The prove is finished since (5) is identical to (1). This theorem proves that AdaBoost works because by adding more and more weak classifiers to the strong classifier, the training error will be decreased.

It is important to point out that for (1) to be valid, Z_t needs to be lower than 1. I am not going to show the proof here, even though I have done it, since its outline can be found in [15]. The fact that $Z_t < 1$ holds comes from the Freund and Schapire's choice of α_t. The mentioned theorem also says that it is desired to seek the lowest Z_t in each round. Therefore, this assumption is adapted on the choice of α_t by seeking the minimum of Z_t. The obtained choice

$$
\alpha_t = \frac{1}{2} \ln \left(\frac{1 + r_t}{1 - r_t} \right),
$$

where $r_t = \sum_{i=1}^{m} D_t(i) y_i h_t(x_i)$, gives the following bound on Z_t:

$$
Z_t \leq \sqrt{1 - r_t^2}.
$$

This bound is sufficient to show that $Z_t < 1$.

4.4.3 WaldBoost

The idea of boosting a cascade of weak classifiers was first introduced by Viola and Jones [19] and later justified by Šochman and Matas [24]. WaldBoost incorporates a requirement on evaluation time as another criterion while training a classifier. The optimal sequential strategy (classifier) S^* according to Wald's sequential probability ratio test (SPRT) is the one with the shortest average decision time subject to constraints on error rates (false positive β and false negative α rates). Defined as

$$\begin{aligned} S^* &= \arg\min_S \bar{T}_S \\ \text{s.t.} \quad & \beta_S \leq \beta, \\ & \alpha_S \leq \alpha, \end{aligned}$$

where $\bar{T}_S$ is the average decision time of a sequential strategy S and the sequential strategy is SPRT defined as

$$S^* = \begin{cases} +1 & R_t \leq B; \\ -1 & R_t \geq A; \\ \text{continue} & B < R_t < A, \end{cases}$$

where A, B are computed from the constraints α, β and

$$R_t = \frac{p(x_1, ..., x_t|y=-1)}{p(x_1, ..., x_t|y=+1)} \approx \frac{p(h_1(x), ..., h_t(x)|y=-1)}{p(h_1(x), ..., h_t(x)|y=+1)} = \frac{p(H_t(x)|y=-1)}{p(H_t(x)|y=+1)}$$

is the likelihood ratio. $x_1, ..., x_t$ are measurements on one test sample x. WaldBoost estimates the likelihood ratio by using weak classifiers' outputs $h_i(x)$ as measurements on the test subject. The method is described in detail in [24].

This has a logical implication that when classifying an sample, one measurement (one weak classifier), the most distinctive, is taken and if the likelihood ratio (SPRT) is significantly biased to one or the other side a decision is made. If not, another measurement of the sample is taken. This continues until a decision is made or all measurements are taken.

4.5 The Randomized Forest Detector

Randomized forests are structures that combine the output of several randomized trees to a single decision. Randomized tree is a decision tree, which, while built, chooses the split criteria in its nodes randomly. Also the training set used to built each tree is a random subset of the whole training set. This ensures the diversity of the chosen trees. Randomized trees were described several times, for example in [1, 9, 12].

There are many variants of decision trees that can be used for classification. They differ in branching factor and mainly in the splitting criterions and tree building. In this thesis, randomized trees were chosen, because they are quickly implementable, the concept is not difficult to understand and they are easy to customize. Also, we wanted to compare their performance to the WaldBoost detector and decision trees allowed us to use the same features (LBP) as in the case of the WaldBoost detector.

In the following sections I introduce the theory of decision trees followed by randomized trees and forests, where I also describe the algorithms implemented.

4.5.1 Decision Trees

Mostly because there are two classes in the case of car detection – car and background, binary decision trees were used in the randomized forest detector. Binary decision tree is a tree structure where each node has exactly two or zero (leaf) descendants, see sketch in Figure 4.3.

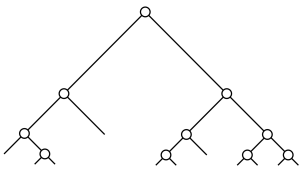


Figure 4.3. Binary decision tree.

Each leaf node is assigned a class during training. To build a decision tree, a split criterion is chosen, which splits the training set into two subsets. One subset is passed to the left child node and the other is passed to the right child node. The subsets are split again until a stopping condition is met. Usually the condition is to have samples of only one class in the node. The node is then assigned this class.

While using the tree for classification, test are being evaluated on the sample until it reaches a leaf. The class of the reached leaf is then assigned to the sample.

Decision trees are based on the principle of bagging (bootstrap aggregating). They split the training set into subsets using splitting criterions (functions), which partition the training set (see Figure 4.4). The more partitions are made (more splitting criterions chosen) the more the training set is shattered. And, if the training set is shattered a lot, then a very good fit to the training set is made. Interestingly, this directly points to the fact that decision trees are prone to overfitting.

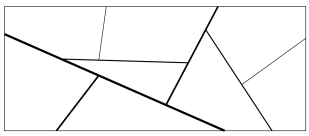


Figure 4.4. Bagging. The thinner the line, the later the split.

4.5.2 Randomized Trees

Special cases of decision trees are randomized trees. Randomized because the splitting criterion in each node is chosen randomly. In the case of the programmed detector, the splitting criterions are Local Binary Patterns (see Section 4.6). The random selection causes the trees to grow differently every time they are built. This fact is conveniently used in randomized forests.

The first version of the detector I programmed used randomly chosen LBP features as splitting criteria in the nodes. This soon turned out not to be sufficient because the trees grew too deep, therefore taking too much time to train and to classify images as well. That is why I decided to generate several random features in each node and choose the best one.

The method of choosing the test with the best split from many randomly generated tests is mentioned in [12]. As the criterion for choosing the feature that best splits

the training set I chose Shanon entropy gain, which is frequently used to measure the quality of a split in decision trees. It was used for example in [1]. The entropy gain $GAIN$ is given by the weighted sum of the entropies in the child nodes:

$$H(Node) = -\sum_{i=1}^{m} p_i \log(p_i), \qquad GAIN = H(Parent) - \sum_{i=1}^{c} \frac{N_i}{N} H(Child_i),$$

where $H(\cdot)$ denotes Shanon entropy, m is the number of classes (in my case 2 – background, car), c is the number of child nodes (in my case 2) and N is the number of samples in a node. The probability p_i of a class i in a node is given by

$$p_i = N^i/N,$$

where N^i denotes the number of samples of the class i in the node N.

The suitability of this criterion to this task is, however, questionable since it turned out to produce very unbalanced trees. Incorporating a different criterion could certainly affect the balance of the tree as well as its depth and, therefore, its average time to detection and performance. Yet, we would have to take in account the fact that the detector is going to classify a huge amount of negative samples and several positive, thus, completely balanced tree is not going to produce the results in the shortest time. This can be a subject of further study.

Incorporating choosing from several features improved the detector's classification performance and speed. But still, a better performance and speed is needed to get closer to the state-of-the-art methods. Thus, we decided to allow for the tree to make final decisions about the samples' class not only in the leaves, but in all nodes of the tree.

An LBP feature is a domain-partitioning classifier (256 or 512 bins). The decision making during training was improved to not only assigning a class to each bin, given by the majority vote of the present samples, but when the bin itself provides a sufficient split, the images from this bin get discarded and not used in training any more. In classification, the images that fall into these bins are immediately assigned a class and not propagated deeper to the tree. This approach assimilates SPRT used in WaldBoost. To see how this method affects the detector's performance see Section 5.3.2.

4.5.3 Randomized Forests

Randomized forests combine the decisions of many randomized trees to produce a better decision. Apart form randomly selecting node tests, each tree is trained on a randomly chosen subset of the whole training set, which ensures even higher variability between the trees. Randomized forests have a much better performance than single trees because they are not so prone to overfitting.

The discriminant function for combining the trees' outputs is described in [9]. In short, it is decided for the class, whose probability sums to the highest value.

Each tree's output is the probability $P(c|l_i(x))$ of the class c in the leaf node l_i, where the sample x that is being classified falls. The probabilities are summed to

$$g_c(x) = \frac{1}{t} \sum_{i=1}^{t} P(c|l_i(x)),$$

where t is the number of trees. It is chosen for the class c for which $g_c(x)$ is the highest. The value of $g_c(x)$ is the confidence measure.

In my application I can set the minimum confidence threshold of $g_c(x)$ for a sample to be accepted as positive. Since the confidence measure is a probability, it ranges from $\langle 0, 1\rangle$ for each class. I translate this representation to the range $\langle -1, 1\rangle$, where I use negative numbers for background and positive numbers for cars.

4.5.4 Training Parameters

When training the randomized forest detector, for each tree, the training set gets shuffled and a half of the training samples is chosen for training. The form of the resulting forest is given by three parameters. The number of trees in the forest, the number of random features that is chosen from in each node and the minimum split in each leaf that has to be reached.

A higher number of trees causes longer training, but a better performance. A higher number of random features that is chosen from shortens the training time, however its impact on the performance is questionable. And, the better the split is required, the longer the training time and deeper tree. Logically, the better split, the higher precision and the lower recall.

4.5.5 Car Rotation Detection

Our idea was to use the fact that a decision tree partitions the training set to detect the rotations of the detected cars. During training, in each leaf, there would end up samples that have only a small variance in rotations. This would happen because the cars in different rotations do not have many similar features.

For each leaf, I used Parzen-window estimate to create a histogram of the rotation angles of the training samples. The Parzen-window estimate $PWE(x)$ for angle x is given by

$$PWE(x) = \sum_{i=1}^{n} p(x, \alpha_i, \sigma); \qquad p(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

where n denotes the number of training samples (in a node) and α_i is the angle of the sample number i. Normal distribution with $\sigma = 5$ was used.

Then, during detection, the histograms returned from the trees in the forest are summed and the angle with the highest abundance is returned as the detected angle.

This method, however, showed basically no correct angle detections because the distribution of rotation angles is basically the same for all nodes (see Figure 4.5). The problem is that the features chosen by the entropy gain criterion do not provide sufficient discrimination between car samples with different rotations. If we wanted this method to work, we would have to incorporate some measure of how well the feature separates different rotations in the feature rating criterion.

4.6 Local Binary Patterns

Local binary patterns (LBPs) are features used for texture description in images. They describe the relations between several pixel intensity values inside of a defined area

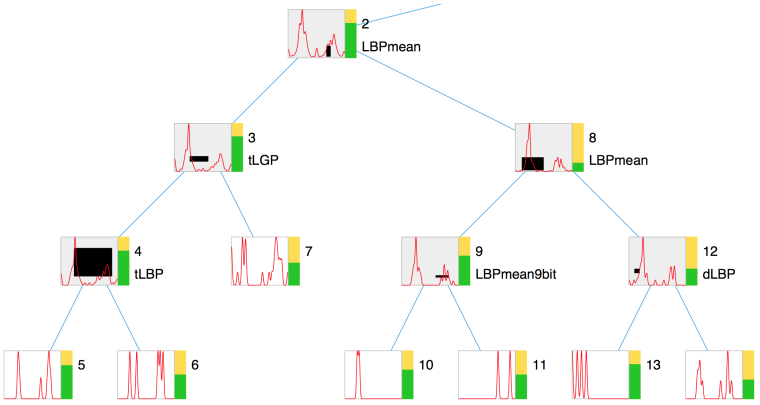


Figure 4.5. Car rotation angle distributions in nodes (red line).

1	2	3
4	5	6
7	8	9

Figure 4.6. 3x3 local binary pattern grid.

in an image. The LBPs used by Trefný in the WaldBoost detector and in this thesis correspond to a grid of 3x3 areas. A prototype is shown in Figure 4.6.

An usual implementation of this pattern compares some kind of measurement on the border areas to the area in the middle (number 5). The result of each comparison is 0 or 1, which makes it, in this case, an 8-bit number. This pattern is placed over some particular area of the examined image or images. In training, the outputs of these measurements on all training samples are grouped to produce a histogram, which is then used for classification.

The 3x3 area does not have a fixed ratio nor fixed size, which means it can spread across any area of the image. The only limit for this pattern is the minimum size of 3x3 pixels. The problem is how to spread the grid over a number of pixels that cannot be divided by 3. In my implementation I specifically set the outer areas to have the same size and the middle one to be different. An illustration is depicted in Figure 4.7.

1	2	3
4	5	6
7	8	9

Figure 4.7. Disproportional (5x3 pixels) local binary pattern grid.

The computation of the number of pixels is

$$col_{13} = \text{round}(w/3),$$
$$col_2 = w - 2 \cdot col_{13},$$

where w is the width of the grid in pixels and col_{13} and col_2 are the widths of the outer and middle areas in pixels respectively. The procedure is the same for row heights. In the case of a 3x3 pixel grid the intensity values of each pixel would be used for further computation, however, when the dimensions of the grid are different, the mean pixel intensity values of all pixels in each area are used.

One can think of many measurements that can be done using this pattern. Eight different types of measurements, whose description follows, were used in this thesis. The types are intentionally the same as in the case of the WaldBoost detector in order to perform a closer comparison of the detection methods.

- **Classical LBP** compares the intensity value of the center area with the intensity values of the border areas from 1 to 4 and 6 to 9, which outputs an 8-bit number.
- **Mean LBP** compares the intensity of each border area 1-4 and 6-9 to the mean value of the intensity computed from all areas 1-9. This local binary pattern also outputs an 8-bit number.
- **9-bit mean LBP** compares the intensity of each area 1-9 (including area number 5) to the mean value of the intensity computed from all areas 1-9. This results in a 9-bit number output.
- **9-bit mean LBP with an incorrectly computed mean** is the same feature as the previous one, but the area number 9 is excluded from the computation of the mean intensity value. This feature was a coincidental finding of Trefný, however because it showed a very good performance he decided to keep it.
- **Local gradient pattern (LGP)** first determines the absolute intensity differences of the border areas 1-4 and 6-9 from the central area 5. Then a mean of these differences is computed. In the end, the absolute differences of the border areas 1-4 and 6-9 from the central area 5 are compared to the mean difference. The output is an 8-bit number.
- **Transition LBP (tLBP)** and **Directional LBP (dLBP)** incorporate also the spatial relations between the areas. They are described thoroughly in [17].
- **Transition LGP (tLGP)** feature is the combination of classical LGP and tLBP.

Using LBPs for classification is straightforward. The trained histogram can be directly used for classification by assigning the majority class from the bin, where the tested image falls.

The 8-bit features have 256 bins and the 9-bit features 512 bins. This is not a problem for AdaBoost training, where the whole dataset is used in each round. However, it is a problem for randomized trees, where the training set gets split in each node. This leads to the situation, in which, after a few splitting rounds, there is not enough training samples for the histogram to be representative. That is for example when there are 30 samples left and an 8-bit feature is chosen, in the best case, there will be 226 empty bins. Which class should be assigned to them?

We decided to use also the data from surrounding bins when there is no sample in the examined bin. However, surrounding bins cannot be taken in terms of the integer number output of LBPs, because close integer values do not necessarily have similar binary representation.

We tackled the problem by incorporating Hamming distance measure, which measures the similarity of binary numbers (see Section 4.6.1). The data from bins having Hamming distance equal to 1 are also used for classification, when there is no sample in the bin where the classified image falls. They are grouped together and the majority class is assigned to the classified image. When there is no data even in the surrounding bins, it is decided for the background class.

This approach partially solves the problem with the large number of histogram bins. Still, when we think of the the nature of LBPs, we should consider whether using texture based features is optimal for car detection since car is, to large extent, described by its shape. In the end, the work Zheng and Liang [22] shows that using shape based strip features leads to excellent results.

4.6.1 Hamming Distance

Hamming distance is a measure of similarity of two binary words (numbers). The definition says that Hamming distance of two binary words is the number of bits in which the two binary words differ. For example (Hd stands for Hamming distance)

$$\mathrm{Hd}(01001110, 00101100) = 3.$$

This measure can be used since it correctly describes similarity of LBPs' output numbers. Used LBPs output 8-bit or 9-bit binary words. Each bit in those words corresponds to one measurement made during LBP evaluation. Having Hamming distance equal to 1 of those words illustrates the situation of only one different measurement. Obviously, it means that those words and therefore the LBP descriptions are in fact very similar, which is the point.

4.7 Line Segment Detector (LSD)

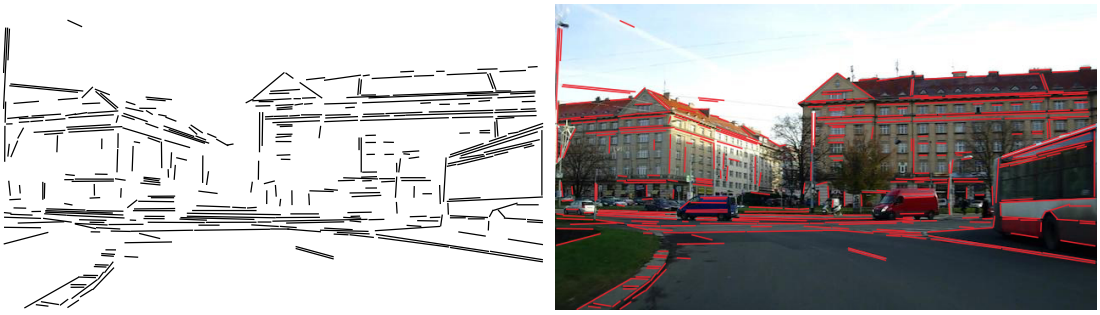
Because randomized forests turned out to carry on too many tests on each scanned window, making it impossible to get even remotely close to a real-time processing, we decided to incorporate a fast line segment detector – LSD. It was recently presented by von Gioi et al. [20]. A huge advantage of this line segment detector is its linear-time performance and parameterless usage. The detection on an HD image on a 2.5 GHz machine takes about 0.5 s.

The principle of LSD is finding line-support regions on a gradient orientation map. A line-support region is a region that contains pixels with the same (up to some tolerance) gradient angle. LSD combines several previously introduced methods to achieve superior results. Its key improvement against other methods considering speed is that the clustering algorithm visits each image pixel only once. For a detailed description I will refer to [20] because I implemented the detector only as a black-box.

The format of the detector's output is displayed below. Each line is represented by two points `x1, y1, x2, y2`, its width `width`, angle tolerance `tol` and kind of confidence `cnf`. Since I used only the first four values I will not be describing the others.

```
x1, y1, x2, y2, width, tol, cnf
```

Sample output of the detector is shown in Figure 4.8. Also a working web demo version of the detector can be found at `http://demo.ipol.im/demo/gjmr_line_segment_detector/`.



a) LSD. b) Original + LSD.

Figure 4.8. Sample LSD output and the corresponding original image with the LSD output.

4.7.1 LSD Window Pruning

We propose a new window pruning method based on line segment detection. By studying the images in Figure 4.8a and 4.8b one can notice the abundance of horizontal lines detected on any present car. By assuming there is at least a certain number of horizontal lines in a window containing a car, we are able to discard a massive amount of scanning windows without even running the randomized forest (or any other) detector.

Using this preprocessing we can quickly skip the image areas containing sky, road, trees or any other area without a sufficient amount of horizontal lines. This leads to skipping up to 4/5 of all scanned windows, which results in a significant speedup.

Horizontal are considered lines within 0.15 rad from the horizontal angle. The absolute value of the direction ϕ of a line is computed

$$\phi = \operatorname{atan}\left(\frac{\Delta y}{\Delta x}\right); \qquad \Delta x = |x_1 - x_2|, \qquad \Delta y = |y_1 - y_2|.$$

The lines not within the angle range of $\langle 0, 0.15\rangle\, rad$ are discarded. Also, the lines longer than the width of the detection window (in my case 52 pixels) would not support any detection window and hence are discarded as well.

For the detail on implementation see Section 6.3.3 and 6.4.3.

Chapter 5
Experiments

In this chapter I first introduce the measures used for evaluation. After that, I present several tests carried out on the WaldBoost detector and on the new randomized forest detector and I elaborate about the results. In the end, a thorough analysis of the randomized forest detector is provided.

5.1 Performance Measures

The measures used for performance evaluation in this thesis are precision, recall, miss rate and the number of false positives per window (FPPW). In the following text I will use abbreviations for true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN).

Precision describes how many samples classified by a classifier as correct (positives) are actually correct. The measure is defined as

$$Precision = \frac{TP}{TP + FP}.$$

Recall in classification describes how many positive samples from a testing set were actually classified as positive. In detection it describes how many labelled cars in an image were found by the detector. Each car must be counted as a TP only once.

$$Recall = \frac{TP}{TP + FN}$$

Miss rate in classification says how many cars from the training set were marked as negative. In detection it is the measure of how many labelled cars in an image were not found. It is defined as

$$Miss\ rate = 1 - Recall.$$

FPPW is a measure for evaluating detection without non maxima suppression. The idea of this measure is to show how many false positive windows a detector produces. It is defined as

$$FPPW = \frac{FP_w}{FP_w + TN_w}.$$

Here, FP_w and TN_w denote the numbers of false positive windows and true negative windows respectively.

5.2 Approving and Rejecting Detections

To approve a detection as a correct detection I use the measure of overlap intersection over union. Intersection over union $iou(x, y)$ of two bounding boxes x and y is defined as

$$iou(x, y) = \frac{Area(x \cap y)}{Area(x \cup y)},$$

where $Area(\cdot)$ is the area of a bounding box. To approve a detection I require $iou(x, y) > 0.5$, which is a value that is most frequently used.

5.3 Performance

In this section I will describe several tests carried on the randomized forest (RF) detector and the WaldBoost detector. I will show how the performance of the RF detector improved with improving the training and detection algorithm. In the end I will compare the RF detector to the WaldBoost detctor.

5.3.1 RF Classification

The first test carried out on the RF detector was a classification test. It was important for parameter tuning – to see how the detector's parameters affect its performance.

Three parameters of the detector can be set – the number of trees (`noOfTrees`), the number of random features that is chosen from during training in each node (`testFeatures`) and the minimum split in each leaf (`minSplit`) as mentioned in Section 4.5.4. The values tested are shown below.

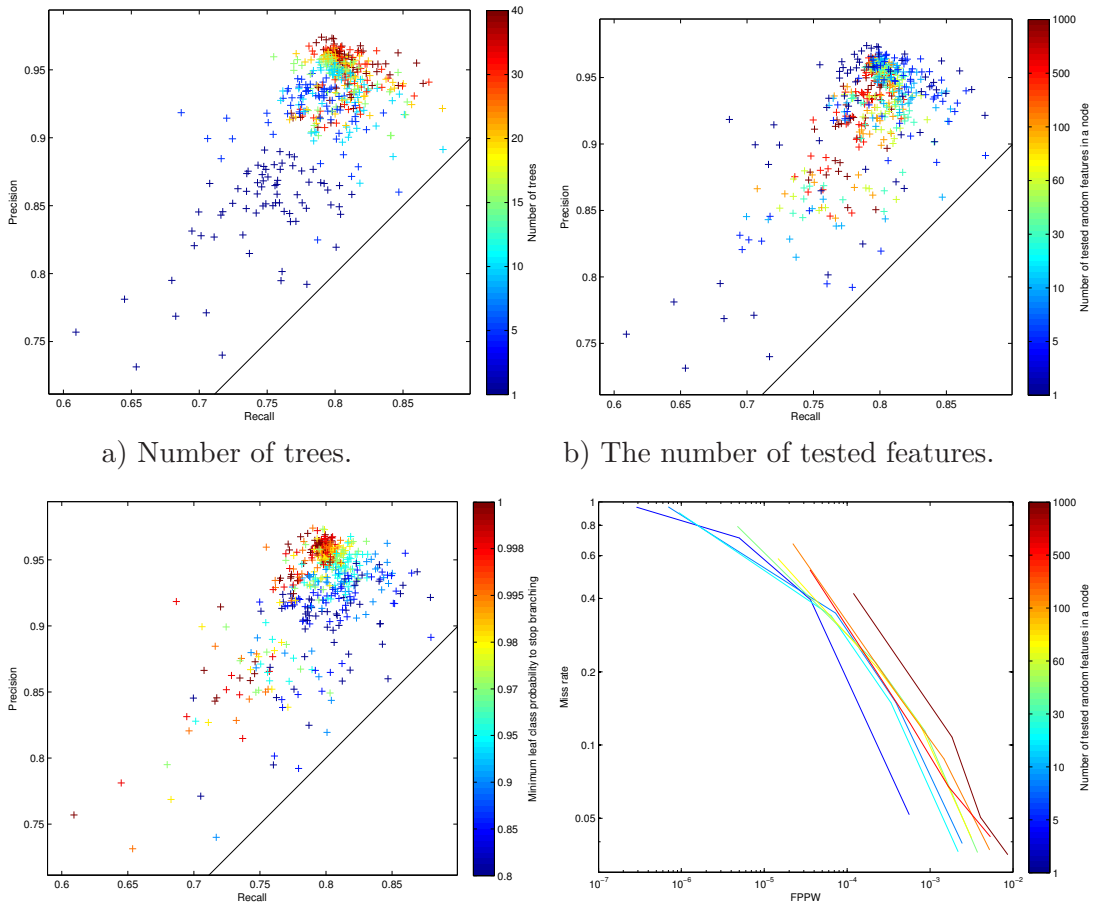
```
noOfTrees       = [1 5 10 15 20 30 40];
testFeatures    = [1 5 10 30 60 100 500 1000];
minSplit        = [0.8 0.85 0.9 0.95 0.97 0.98 0.995 0.998 1];
```

The values were combined each with each producing 504 detectors in total. All those detectors were trained on the JT data set and tested for classification performance on the Dejvicka, KITTI_FV and KITTI_OCC data sets, where the positive samples were extracted as described in Section 3.3 and the background samples extracted with our first method described in Section 3.5. The detectors were trained and tested on data sets having 50% positive and 50% negative samples.

In order to clearly show the detectors' performance, the detectors were tested for precision and recall (see Figure 5.1).

Figure 5.1a shows that the more trees are used in a forest, the better the performance. These results are in accordance with the trend suggested by T. K. Ho [9]. Because the trend is the same for any detector setup, I will fix the number of trees to 20 for all other tests.

Figure 5.1b surprisingly says that the detectors where the feature selection was nearly random have the best performance. Especially considering recall. However, there is no significant trend when increasing the number of considered features, which suggests that the parameter is dependent on other parameters. Nevertheless, some trend is obvious when tested for detection. Figure 5.1d, which was created as described in Section 5.3.2,



a) Number of trees. b) The number of tested features.

c) Minimum split. d) Detection. The number of tested features. Split 0.85.

Figure 5.1. Classification performance on the Dejvicka dataset.

suggests that the more random the feature selection is, the better the performance. Only for a very strict training set split required during training (more than 0.97) this trend reverses.

The trend in Figure 5.1c was expected, as it says that the better split is required to be reached during training, the better the precision and the worse the recall.

Because the detectors (forests) are random, the results could be slightly different if the detectors would have been retrained and the tests ran again. This causes noise in the results. In order for the results to be representative, many detectors were trained and therefore we can observe some trends in the results, which represent the reality well.

5.3.2 Any-Node Decision Making in the RF Detector

As described in Section 4.5.2, the basic version of the RF detector was extended to conveniently discard images that during training fall in a bin with a sufficient split. Similarly, during classification, samples are not forced to reach a leaf, but they can be classified in any node, providing they fall in the bins with sufficient splits.

This test compares the classical RF and the extended one. A detection test was carried out on 100 randomly chosen images from the Dejvicka dataset. Non maxima suppression was not used and therefore miss rate and FPPW measures were used for evaluation.

The detectors were trained as described in Section 5.3.1. The number of trees in a forest was fixed to 20.

On each image a set of all windows scanned by scanning window was generated and the number of all negative windows was computed using intersection over union with bounding boxes of all labelled cars. The windows that overlap less than 0.5 with each bounding box were marked as negative. The bounding boxes were extracted from the labelled cars as described in Section 3.3 for all three groups of poses outlined in Section 3.3.3.

In detection, all detected bounding boxes that overlap less than 0.5 with any labelled bounding box were marked as false positives. True positive was marked any labelled bounding box that overlapped with at least one detected bounding box.

The results in Figure 5.2 suggest that allowing for making a decision in any node of a tree improves the performance of the detector. Moreover, the average time to detection was 2.5 times shorter. That is a very important improvement.

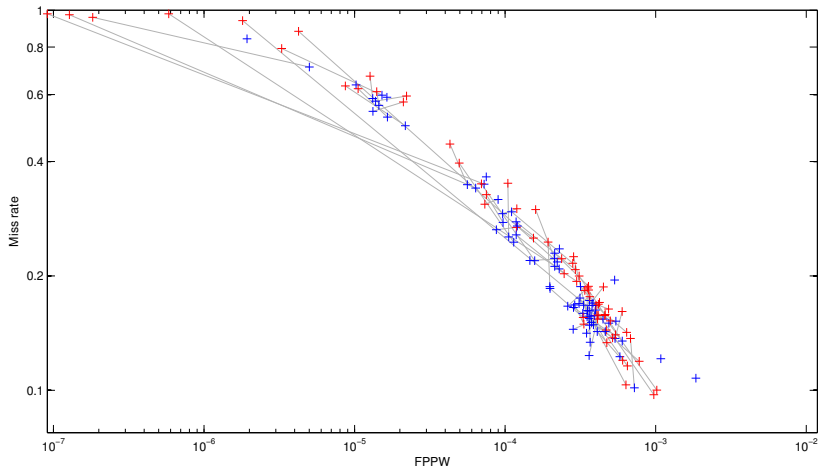


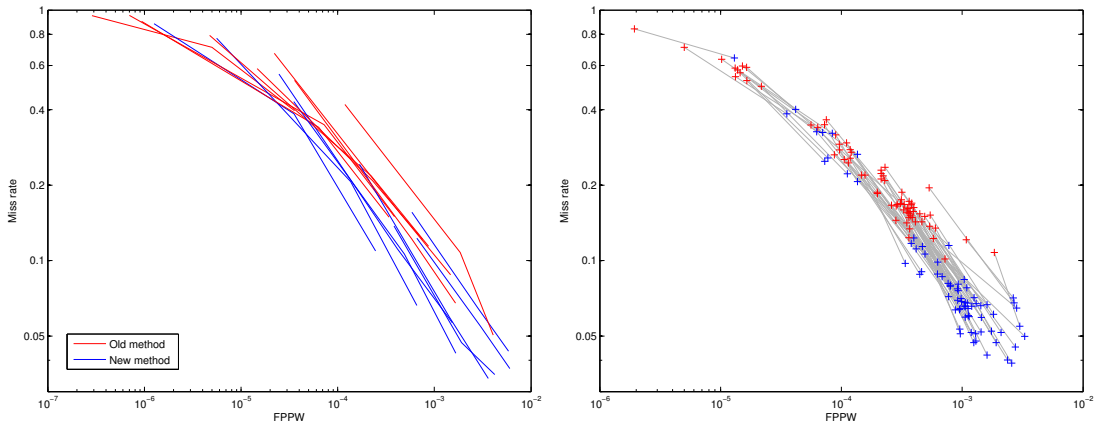
Figure 5.2. Detectors without discarding (red), detectors with discarding (blue). Detectors with the same training parameters are connected with a grey line. Confidence threshold to accept a detection was set to 0.8.

5.3.3 The Influence of Background Training Images

As described in Section 3.5, two new methods for extracting background training samples were suggested. The first proposed method (old method) extracts random background samples from the same images as the positive data are extracted from. The other (new method) takes in account also the distribution of the abundance of windows of different sizes.

This test was again trained as in Section 5.3.1, however, for one set of detectors the new background extracting method was used. Tests were than carried out on a 100 random sampled images from the Dejvicka data set. Three different confidence acceptance thresholds – 0.7, 0.8 and 0.9 were used to obtain curves. The results are shown in Figure 5.3.

Figure 5.3a suggest a slight improvement in the detectors performance when the new method for background extraction was used. The improvement is significant especially in terms of miss rate, whereas the FPPW very often increased as shown in Figure 5.3b.



a) Three thresholds. Min. split 0.85. b) Threshold 0.8 only. All detectors.

Figure 5.3. Background extraction influence on detectors' performance. Red represents the old method, blue the new method.

The increase in FPPW can be partially eliminated by raising the confidence acceptance threshold. This improved the detectors' FPPW, however their miss rate raised so that the performance was very similar to the first method. The results suggest that the new method is superior, however, a further analysis including non maxima suppression would have to be made to see its real power.

5.3.4 LSD Window Pruning

A novel method for pruning windows based on LSD was introduced in this thesis (Section 4.7). This test, which was again carried out the same way as described before, shows the influence of LSD pruning on the detectors' performance and speed. The number of trees was fixed to 20 and threshold of confidence was set to 0.8. Then, the same detectors were ran 3 times, each time with a different window pruning setup.

Figure 5.4 shows the impact of LSD window pruning on the detector' performance. It is obvious that LSD pruning decreases the FPPW of the detectors by an order of magnitude, which is caused by discarding windows that could possibly be evaluated as positive when LSD is not used. However, the window pruning also has a negative impact on miss rate. This behavior could have been expected, nevertheless, the drop in miss rate is significant (about 10%). The drop can be partially caused by the fact that the implementation was not ideal as described in Section 7.

The impact of the pruning on the detectors' speed was enormous. For 2 horizontal lines required within a tested window the average detection time on a HD image shortened from 15.8 seconds to 4.2 seconds, which is 3.7 times shorter time. Actually, if the LSD was ran only once and not each time the image is scaled, the speedup could be even more significant. The comparison of the different setups is shown in Table 5.1.

	Without LSD	LSD, 1 line	LSD, 2 lines	WaldBoost
Det. time	15.8 s	7.1 s	4.2 s	0.9 s
SDPTP	32.7	15.5	13.7	9.1

Table 5.1. LSD pruning impact on detection time and support detections per true positive (SDPTP). Mean values. WaldBoost is shown as a reference.

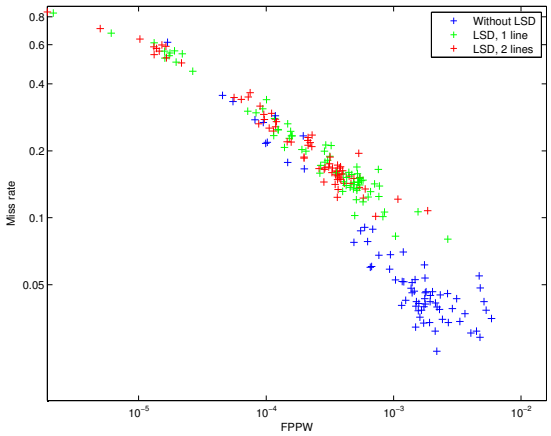


Figure 5.4. LSD pruning impact on detectors' performance.

Even though the window pruning did not cause any improvement in performance in the tests without non maxima suppression, it could be different when NMS would be used. As can be seen in Figure 5.5, when too many support detection are found, it is possible the NMS algorithm combines them in a way they cancel each other and therefore do not show any detection on the found cars. In Figure 5.5c on the left, the small cars in the background were located correctly, however, in Figures 5.5a and 5.5b on the left some of the cars were not located.

5.3.5 WaldBoost vs. Randomized Forest

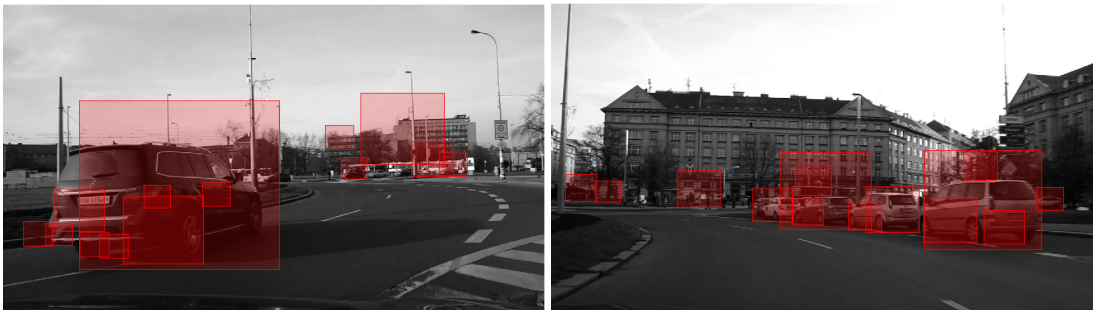
In order to compare the WaldBoost detector to the randomized forest detector I performed a detection test also on the WaldBoost detector.

The curves in Figure 5.6 was obtained by varying its confidence threshold for accepting detections. The test was carried out on a 100 random images from the Dejvicka dataset – the same ones as in the case of the randomized forest detector. The results of the randomized forest detectors were taken from the test in Section 5.3.3. Only the detectors trained on the new background samples and 0.85 minimum split are shown in Figure 5.6.

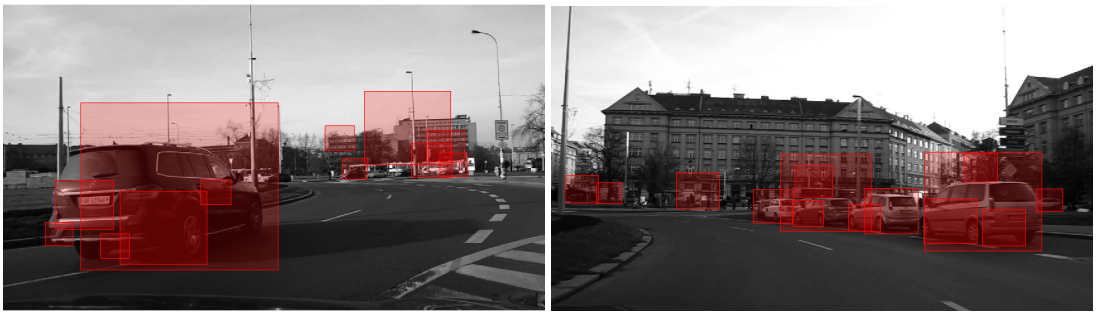
Also, a test with NMS was carried out on the detectors. Multiple detections of the same car were not counted as multiple true positives neither as false negatives. Having in mind that the WaldBoost detector has a much more sophisticated NMS algorithm, the results are only informative and do not represent a fair comparison. Figure 5.6b suggests that the randomized forest detector with NMS has a much worse performance then the WaldBoost detector.

The results truly describe the situation in Figure 5.5, where it is obvious that the randomized forest detector has many false detections. The fact that recall is also very low can be assigned to the fact that some of the cars were detected in for example only one window and therefore the NMS algorithm threw them away for insufficient number of support detections.

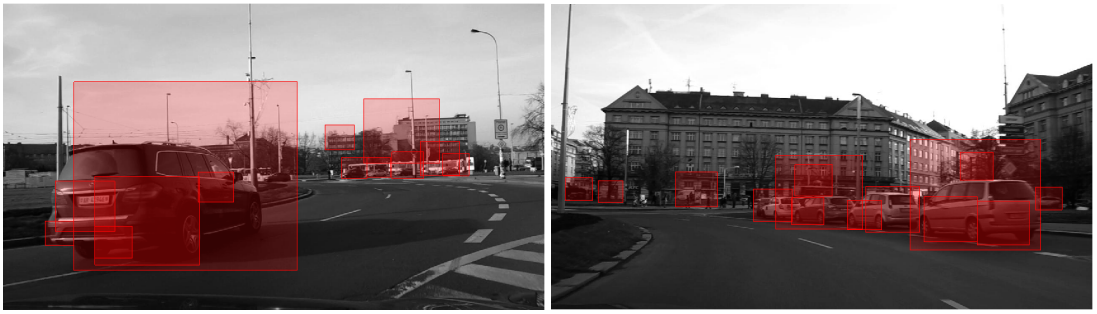
The WaldBoost detector obviously outperforms the randomized forests. With the randomized forest detector we get similar miss rate to the WaldBoost detector, while having FPPW higher by one order of magnitude (Figure 5.6a). Table 5.1 than shows that the WaldBoost detector also dominates while considering speed. Thus, with the



a) No window pruning.

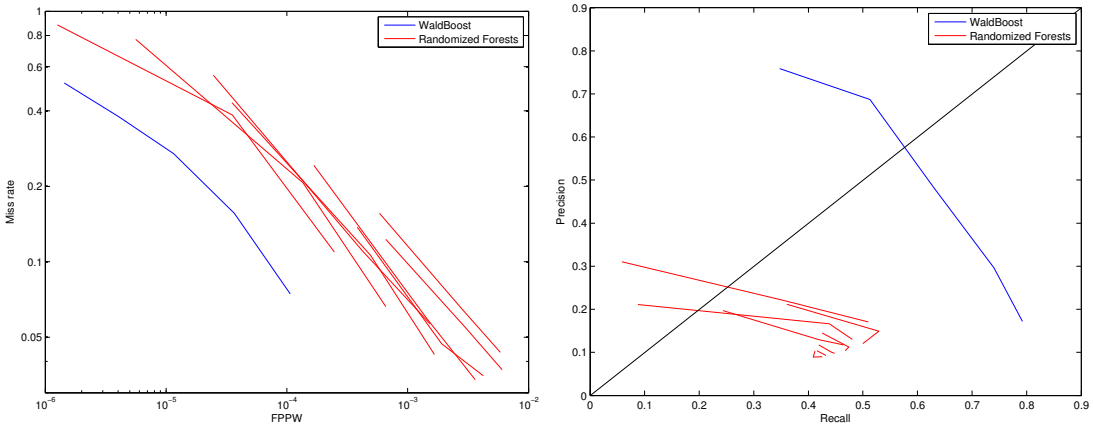


b) LSD window pruning. 1 line within a window required.



c) LSD window pruning. 2 lines within a window required.

Figure 5.5. LSD window pruning with NMS. Sample images with detections (red).



a) Without NMS. FPPW vs. miss rate. b) With NMS. PR curves.

Figure 5.6. WaldBoost versus randomized forests.

randomized forest there is still a long way to go. Yet, there are many ways to improve the detector (see Section 7), which is going to be the subject of future work.

It is important to mention that the information about the detected type of view in the WaldBoost detector was not taken in account during this evaluation because the randomized forest does not have this ability. The detections were evaluated only in terms of the percentage of intersection over union with the labelled cars.

Sample images with displayed detections from both detectors are shown in Figure 5.7. When studied in detail, one can notice similarities in the obtained detections and false detections.

5.4 Data Sets' Difficulty

To test the difficulty of the data sets used in this thesis I used the classification test described in Section 5.3.1. I plotted the classification results on the three data sets in Figure 5.8.

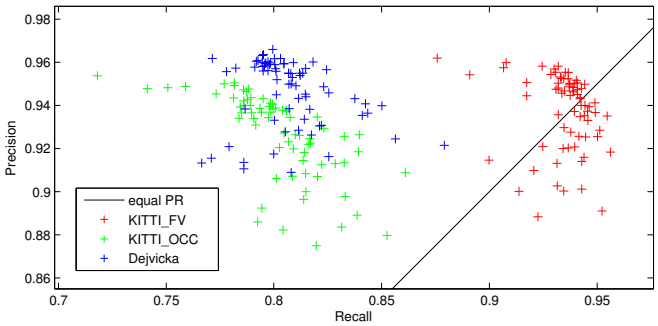


Figure 5.8. The difficulty of the data sets.

It is obvious that the KITTI_FV data set is the easiest one. The lowest performance the detectors showed on the KITTI_OCC data set, however, the difficulty of this data set is given by the fact that many cars are occluded. The Dejvicka dataset is a little bit easier, however not as many occluded cars are present and its difficulty lies in the car rotations present.

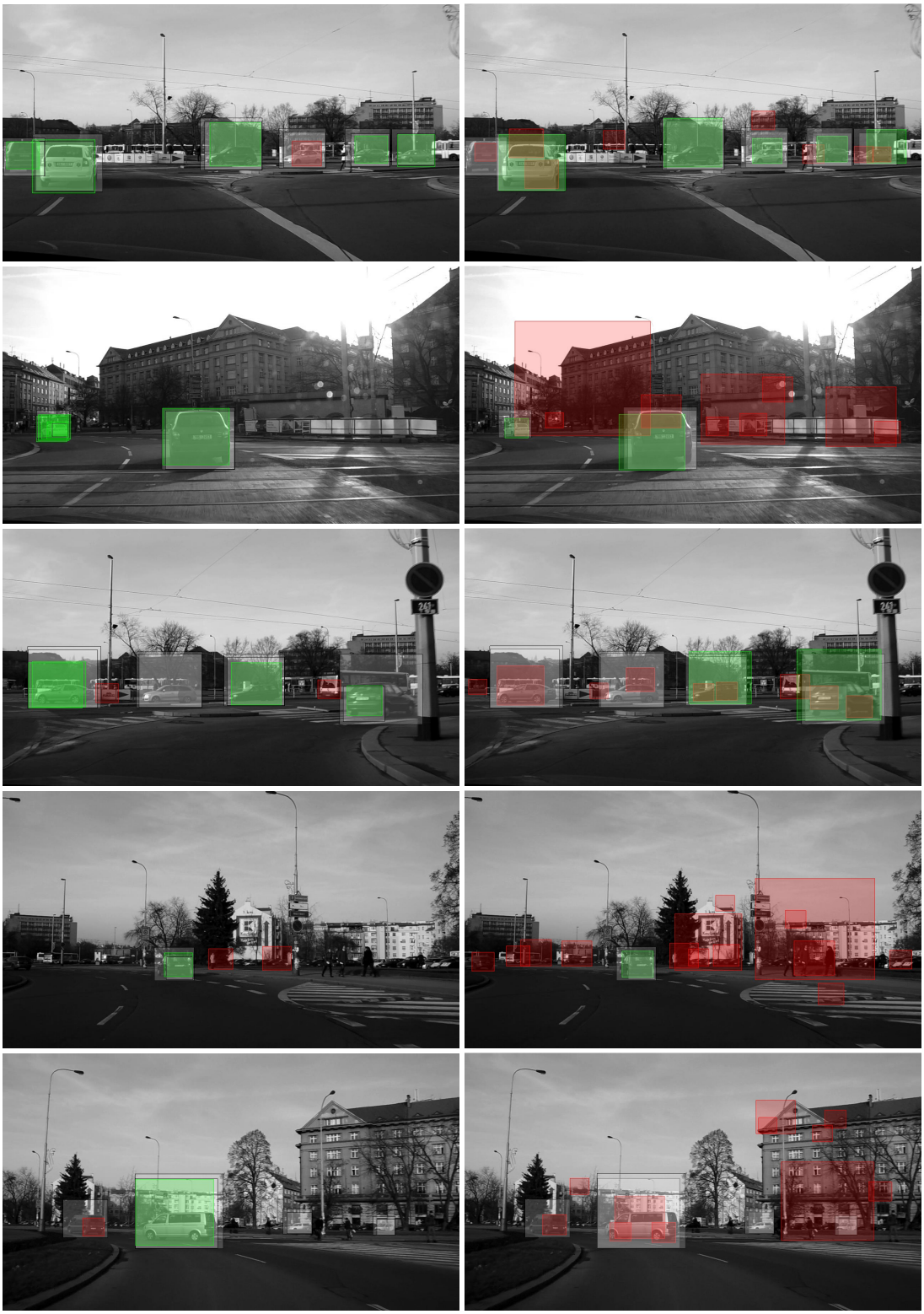
5.5 Time

Since short detection time is desired in real-time applications I provide several tests showing the randomized forest detector speed. Its speed is also compared to the WaldBoost detector.

5.5.1 Impact of the RF Setup

During the classification test in Section 5.3.1 the times it took for the detectors to classify the Dejvicka data set was measured. The detectors were tested with 32,326 image samples from which 50% were positive and 50% negative.

The number of trees was fixed to 20 because this setting has an obvious impact of prolonging the classification time. The other two parameters – the minimum split



a) WaldBoost. b) Randomized forest.

Figure 5.7. WaldBoost and randomized forest detections' comparison on images. Labelled bounding boxes are white, correct detections green and false detections red.

(MS) and number of tested features in each node (NF) were varied. In Table 5.2 we can see trends in changing the classification time.

MS \ NF	1	5	10	30	60	100	500	1000
0.8	0.72	0.54	0.53	0.48	0.46	0.48	0.46	0.48
0.85	0.83	0.60	0.59	0.51	0.71	0.48	0.49	0.45
0.9	1.02	0.73	0.65	0.60	0.58	0.55	0.56	0.55
0.95	2.07	1.02	0.91	0.81	0.76	0.74	0.66	0.68
0.97	1.68	1.38	1.22	1.08	1.02	0.97	0.89	0.86
0.98	1.62	1.58	1.53	1.35	1.19	1.13	1.02	0.98
0.995	1.71	2.20	2.18	2.03	1.94	1.79	1.53	1.49
0.998	1.68	2.28	2.28	2.42	2.20	2.14	2.01	1.76
1	1.86	2.54	3.51	2.57	2.30	2.25	1.95	1.89

Table 5.2. RF classification time on a 2.5 GHz machine.

Table 5.2 suggests that the better the split required during training, the longer the time of classification. It is logical since the trees grow deeper when a better split of the training set is required. For the number of tested features during training is the tendency reversed. It does not have to be immediately obvious because for 1 tested feature the times are short. However, this is given by the fact that, in training, when the selected feature does not split the training set, the branch is not evolving further. Therefore, the trees do not grow as deep as in other cases.

The achieved classification times suggest that detection will take a lot of time because an HD image has approximately 500,000 windows to be evaluated. The times in Table 5.2 represent the time needed to classify 32,326 samples, which when recalculated gives 15 times longer time needed to classify all windows in a HD image. That is the reason for incorporating the new method of LSD window pruning.

5.5.2 Detection Time

An important attribute of a detector is its speed. The average time it took for a detector to detect cars in an image was measured while evaluating the test in Section 5.3.4. The measures were made on a random subset of 100 images from the Dejvicka dataset, which means that the vast majority of the images are in HD. Therefore, I will be talking about the images as if they were in HD.

Three detectors were compared for detection speed as shown in Table 5.3. The randomized forest detector without LSD window pruning, which obviously has the longest running time. The RF detector with LSD window pruning and the WaldBoost detectors that run much shorter time.

	RF without LSD	RF with LSD	WaldBoost
Time	15.8 s	4.2 s	0.9 s

Table 5.3. Average detection time on an HD image on a 2.5 GHz machine.

The WaldBoost detector would probably be a little faster than measured because its measurement were made in Ubuntu in VirtualBox[1]), whereas the randomized forest detectors were tested in OS X.

[1]) VirtualBox is a software for creating virtual operating system environment within a different operating system. The system running in VirtualBox therefore does not have all the power of the computer available.

Chapter 6
Implementation

In this chapter, the implementation of the detectors and the training and testing procedures are described. The stress is put on the randomized forest detector because the WaldBoost detector was used only as a black box.

Two programming languages – C++ and MATLAB[1]) were used in this thesis. The WaldBoost detector is programmed in C++ and can operate as a stand-alone C++ program. However, for the sake of testing, the MEX library[2]) was implemented in order to translate the program's output to the format that can be further handled by MATLAB. This structure, which is displayed in Figure 6.1, is convenient since MATLAB allows for simple processing and evaluation of the detector's output.

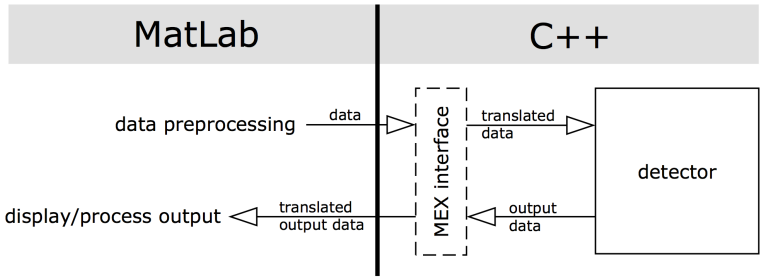


Figure 6.1. MatLab/C++ cooperation schema.

I decided to integrate the same software structure in the randomized forest detector. Moreover, I use it also in training. The structure is convenient because it allows me to preprocess all data easily in MATLAB and then just pass it to the C++ part. And, since the MEX function is used only as an interface for translating MATLAB data types to C++ data types and the other way round, it is possible to leave out this interface in the future and use the detector only as C++ software.

6.1 The WaldBoost Detector Implementation

As already mentioned, the WaldBoost detector is programmed in C++. I do not know its exact structure because I obtained and used it only as a black box. The only changes I could make to the detector were setting up sliding window (window size, shift and image scaling) and non maxima suppression (minimum accepted confidence threshold and minimum number of combined windows for detection). Therefore, I will no be describing this detector any further.

[1]) MATLAB is a short from matrix laboratory. It is a high-level programming language and a software developed by MathWorks. For more details see `http://www.mathworks.com/products/matlab/`.

[2]) `http://www.mathworks.com/help/matlab/mex-library.html`

6.2 The Randomized Forest Detector's Structure

In the case of the randomized forest detector, I designed and programmed the whole detector myself. I tried to design the structure of the classes of the detector to be as easily editable, extendable and legible as possible. Each class can be edited separately without affecting the overall functionality of the detector, therefore, enhancements of the methods and functions can be done just by editing one or two files without changing the rest.

The simplified layout of the classes is displayed in Figure 6.2. Some classes, functions and methods are omitted because they are not important to understand the overall concept.

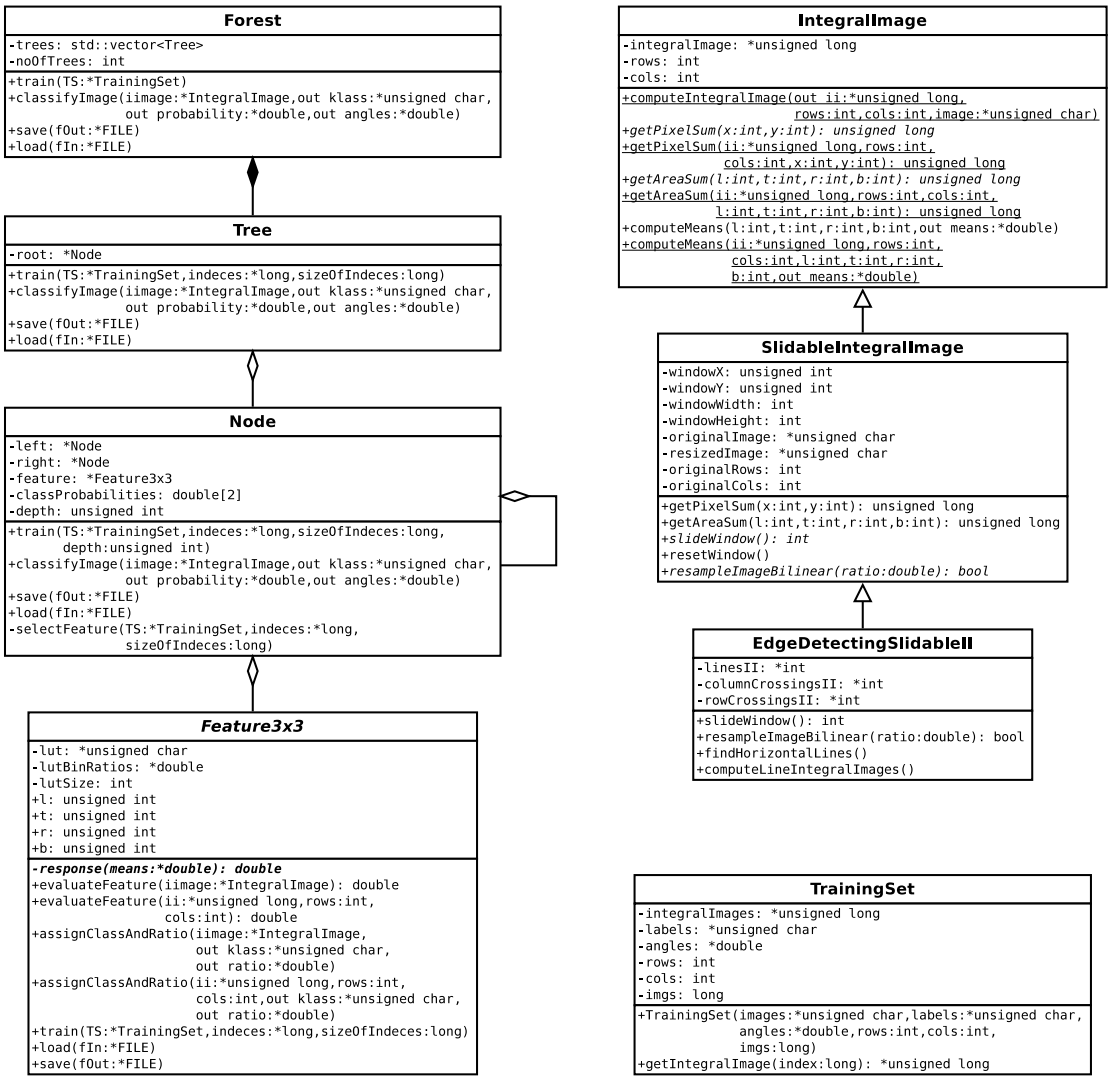


Figure 6.2. The randomized forest classes' structure (simplified).

The access to the whole detector is available through the Forest class. When training or loading a detector, one instance of this class is created and calling its method *train* or *load* builds the whole forest. The forest is made of instances of the Tree classes, which are assembled from instances of the Node class. The Tree class is essentially just a pointer to the root node. The whole tree is then built on this root node.

Speaking of the Node class, each node contains pointers to its children nodes and a splitting test – a feature. The only exceptions are leaf nodes, which do not have any children nor a feature assigned.

The feature is assigned to each node during training. Each feature class (there are eight of them, see Section 4.6) derives from the abstract class Feature3x3. This makes it possible to use one container in the Node class to store a feature of any type. The features differ in the size of their lookup table and their *response* method.

6.2.1 The TrainingSet Class

An instance of the TrainingSet class is used during training. It stores the whole data set in the form of integral images (described in Section 6.4.1) as a single array on heap to save space and computational time.

The instance of the TrainingSet class is created in the very beginning of the training cycle. A set of training images is passed from MATLAB in the form of a single array. The images are immediately translated into integral images using the static method *computeIntegralImage* from the IntegralImage class.

When the instance of the TrainingSet class is used in the future, only pointers to the instance or pointers to separate integral images are passed to avoid senseless copying of the data set.

6.2.2 The IntegralImage Class

The IntegralImage class has its main purpose in computing and storing an image in the integral image representaion (see Section 6.4.1). It can be used in two modes because it contains dynamic methods as well as static ones. The dynamic methods are used during detection, whereas the static methods are convenient during training.

In training, creating of a huge number of IntegralImage instances is avoided using a huge array that contains all integral images one after another. The static methods of the IntegralImage class can be then called on this array. Whereas during detection, only one instance of one of the inheriting classes (SlidableIntegralImage or EdgeDetectingSlidableII) needs to be created, which is fast and convenient for further manipulation.

The SlidableIntegralImage class inherits from the IntegralImage class. It is an extension that allows for processing parts of a big image. The methods *getPixelSum* and *getAreaSum* are reprogrammed to always work only with the part of the integral image that is specified by the window position and size. The window can be moved across the image by calling the *slideWindow* method or reset to the 0,0 position by calling *resetWindow*. Also, resizing of the image is implemented to cover the full functionality of sliding window (details about sliding window are in Section 4.2.1). Specifically, bilinear interpolation was used for image resizing to achieve good results (see Section 6.4.2).

The EdgeDetectingSlidableII class inherits from the SlidableIntegralImage class. This further extension incorporates the LSD-based window pruning described in Section 4.7. When the instance of the class is created, the horizontal lines are found in the image and when the image is resized, the search for lines is carried out again. In the future it would sure be possible to prune the lines instead of running the detection again and save some computational time.

I suggested and implemented a method of storing and evaluating the numbers of lines in a certain area, which is described in Section 6.4.3. This method is then conveniently

used for discarding windows where there are not enough horizontal lines. Every time the *slideWindow* method is called, the window keeps being slid until a window with a sufficient amount of horizontal lines is found.

6.2.3 Saving and Loading the Detector

The randomized forest detector is saved and loaded from a text file. While saving a forest, the *save* methods of the Forest, the Tree, the Node and the Feature3x3 classes are recursively called in the depth-first manner. During loading, the methods are called in the same exact order and thus, the forest is loaded to the same form. The structure of the text file is illustrated bellow.

```
window_width window_height
NumberOfTrees: no_of_trees
--Tree--
depth is_leaf backg_prob car_prob
angles 361 angle1 angle2 ... angle361
feature_type l t r b bin1 bin2 ... binN
carRatios N cr1 cr2 ... crN
depth...
...
--Tree--
...
```

The first line of each text file contains the information about the dimensions (`window_width window_height`) of the training samples the detector was trained on. It is used during detection to set the scanning window minimum dimensions. The second line simply says how many trees (`no_of_trees`) there are in the forest.

The complete listing of all trees follows. Each tree starts with the line `--Tree--`. This makes it easier to view the file manually. All nodes are then listed in the depth-first manner.

Each node listing starts with the information about its depth `depth`, whether the node is a leaf or not `is_leaf`, followed by the Bayesian probability of the background and the car class. Next, the values of the Parzen-window estimates of the car rotations present in this node are written, starting from the angle $-180°$ and ending with the angle $180°$. Lastly, in the case the node is not a leaf, the information about the chosen feature are written out. Starting with the feature type `feature_type`, which is represented by a number 1 to 8. The positions of the top left (`l, t`) and bottom right (`r, b`) corners are saved, followed by the lookup table. The size of the lookup table `N` depends on the feature type. It can be either 256 or 512 numbers. The same accounts for the car ratios, which describe the percentage of car images within each bin of the lookup table, therefore telling the probability of the car class in each bin.

6.3 The Modes of the Randomized Forest Detector

The MEX interface is set up in a way to be possible to use the detector in three different modes, depending on the parameters sent to the MEX function. It is convenient because the same MEX function can be called while training, testing or performing detections. The first two arguments of the function are common, the rest is usage-specific. A call to the MEX function from MATLAB looks followingly.

```
carDetectorLN(mode, filename, ...);
```

The argument *mode* takes one of the values "train", "test" or "detect", which sets the detector mode. The argument *filename* says where the trained detector will be saved or which text file to read the detector from.

6.3.1 Training

The call invoking the training mode of the MEX function is displayed below.

```
average_depth = carDetectorLN('train', filename, training_set,
   training_labels, angles, no_of_trees, min_leaf_perc, no_of_features);
```

First, the training samples need to be extracted. The process of extraction of positive data is thoroughly described in Section 3.3. It runs in MATLAB and its outcome is a RxCxM matrix of positive data, where R represents the number of rows of each image, C the number of columns and M the size of the training data set. Next, the negative training samples are extracted as described in Section 3.5, producing a matrix of the size RxCxN. The two matrices are concatenated to form the training set *training_set*. The corresponding labels are stored in *training_labels*.

The rotation angle of each positive training sample is either estimated (the JT and the Dejvicka datasets) or read from the database description files (the KITTI dataset) and stored in *angles*. The last three parameters serve for the training setup.

When the training mode of the C++ program is called, the training set is first translated into integral images and stored in the TrainingSet class. Before training each tree, the training set is randomly shuffled. Conveniently, what gets shuffled are the images' indices and not the whole array of integral images. The chosen indices are then being passed together with the pointer to the TrainingSet instance in order to avoid the training set copying.

When the training is finished, the detector is saved to the desired text file.

6.3.2 Testing

The name "testing mode" is a little misleading since, in reality, it is a classification mode. However, in this mode, the detector obtains a testing set (*testing_set* and *testing_labels*) of images, which is in the same form as previously described training set. They are classified by the forest and the statistics – the numbers of true positives, true negatives, false positives and false negatives are returned. The mode serves for a fast evaluation of a detector's classification performance.

```
stats = carDetectorLN('test', filename, testing_set, testing_labels,
   angles, confidence_threshold);
```

When the testing mode of the C++ program is called, the TrainingSet class is conveniently used for the representation of the testing set. The desired detector gets loaded and each image from the testing set is classified. The statistics are computed and returned to MATLAB as *stats*, where it is possible to do further processing. No statistics concerning rotation estimation are returned because it turned out not to work as well as thought it would be. Nevertheless, it can be included in the future.

6.3.3 Detection

The detection making mode is the main usage mode of the program. It is invoked by the command shown below, where *image* is a grayscale image to be searched for cars. It is possible to switch on or off non maxima suppression by setting *use_nms* to 1 or 0 respectively. The *confidence_threshold* argument sets the threshold for a window to be accepted as a positive detection.

```
[bb, stats] = carDetectorLN('detect', filename, image, use_nms,
   confidence_threshold);
```

The process of making detections in the C++ program is sketched in Figure 6.3. It demonstrates the use of the SlidableIntegralImage and the EdgeDetectingSlidableII class. Sliding a window is always performed within these classes by calling the method *slideWindow*. When the end of the image is reached, the image is resized and the window slides again from the start. As soon as the image is too small to be further resized, the detection loop is quit and the found positive windows are grouped using non maxima suppression (see Section 6.4.4 for details about NMS implementation).

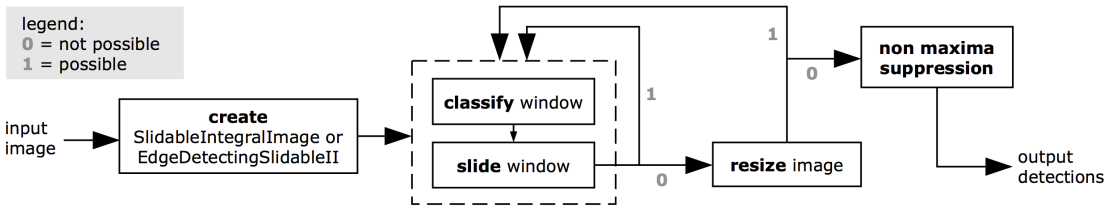


Figure 6.3. The detection process outline.

The output consists of two variables. The variable *bb* contains the found bounding boxes together with their confidences and angles. The variable *stats* contains the total number of windows scanned, the number of positive windows before NMS and after NMS.

The detection process is more sophisticated when the EdgeDetectingSlidableII class is used. When an instance of this class is created, the LSD algorithm is called on the image and all relevant horizontal lines are found (see Section 4.7 for details). Then, the line integral images (see Section 6.4.3) are computed. When the *slideWindow* method is called, it is always checked for the number of horizontal lines within the window. If the number is insufficient the window keeps sliding until a satisfactory window is found.

6.4 Implemented Algorithms

6.4.1 Integral Image

The concept of integral image was presented in Viola and Jones [19]. It allows for a fast evaluation of rectangular features because it precomputes the sums of pixel intensities beforehand. From those sums, it is possible to determine the sum in any rectangle in the image using 4 operations only.

As described in [19], the value of the pixel x, y in the integral image is the sum of intensities from the original image of all pixels above and to the left of x, y inclusive. Each pixel in integral image is computed as

$$II(x, y) = \sum_{x' \leq x, y' \leq y} image(x', y'),$$

where $II(x, y)$ is the sum on position x, y in the integral image and $image(x', y')$ is the pixel intensity in the original image.

To determine the sum of pixel intensities in a particular rectangle area D, displayed in Figure 6.4, we need to compute the following. Integral image stores the sums for areas A, $A + B$, $A + C$ and $A + B + C + D$. To get D do

$$(A + B + C + D) - (A + B) - (A + C) + (A) = D. \quad (1)$$

This process clearly shows how convenient it is to store the data in the integral image form.

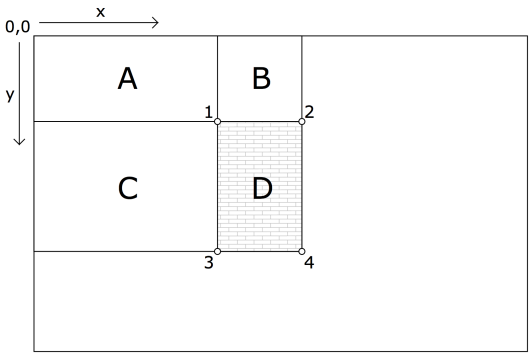


Figure 6.4. Integral image.

6.4.2 Bilinear Interpolation

Bilinear interpolation is a method for resampling images. I used bilinear interpolation because it gives better results than nearest neighbor interpolation and also it is easier to implement than bicubic or any other more sophisticated interpolation. The implemented version is described in Wikipedia[1]).

Bilinear interpolation computes the intensity value of the new pixel A as a weighted combination of the intensities of the four surrounding pixels P_1 to P_4 from the original image. The situation is illustrated in Figure 6.5.

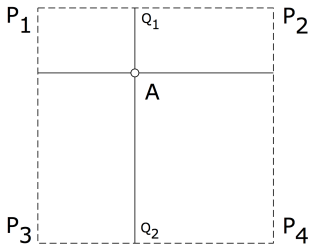


Figure 6.5. Bilinear interpolation.

[1]) http://en.wikipedia.org/wiki/Bilinear_interpolation

The intensity value $i(A)$ of the pixel A is determined by first interpolating in x direction, which gives the intensity values of Q_1 and Q_2 and then interpolating in y direction. The intensity values of Q_1 and Q_2 are

$$i(Q_1) = i(P_1)\frac{P_{2x} - A_x}{P_{2x} - P_{1x}} + i(P_2)\frac{A_x - P_{1x}}{P_{2x} - P_{1x}},$$
$$i(Q_2) = i(P_3)\frac{P_{4x} - A_x}{P_{4x} - P_{3x}} + i(P_4)\frac{A_x - P_{3x}}{P_{4x} - P_{3x}}.$$

Now, by interpolating in y direction, the intensity value of pixel A is obtained as

$$i(A) = i(Q_1)\frac{Q_{2y} - A_y}{Q_{2y} - Q_{1y}} + i(Q_2)\frac{A_y - Q_{1y}}{Q_{2y} - Q_{1y}}.$$

This process runs in a loop and is carried out for each pixel of the new (resampled) image.

6.4.3 Line Integral Image

Line integral image is one of the contributions of this thesis. It was designed to speed up the window pruning based on LSD.

As well as the previously described integral image (Section 6.4.1) its concept is based on summing from the top left corner of the image. However, instead of pixel intensities, the lines that have both their ends inside of the rectangular areas are summed. For example in Figure 6.6 the number of lines within the area A would be 5.

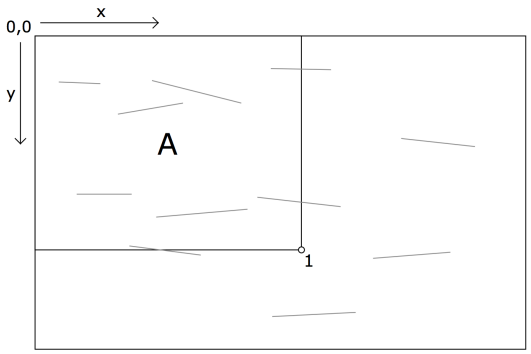


Figure 6.6. Line integral image.

The line integral image is created in two steps. First, an image (array) that marks the pixels where each line has its maximum x coordinate and maximum y coordinate is created. The summation is then carried out on this new image.

The number of lines within any area can be then computed similarly as in Equation (1). Yet, that would not be sufficient. If the computation would be made for the area D in Figure 6.4, most probably we would get a higher number of lines than there actually are. It is caused by the lines that cross the boundaries between the areas C and D and between B and D. These lines would not be included in B or C, but they would be included in A, therefore resulting in inconsistencies.

To solve that, I proposed one more type of integral image, which would compute the number of lines that cross given column or row. Again, this new type of integral image

is created in two steps. First, an image (array) is computed that marks where each line crosses rows (or columns) from the equation of line:

$$y = ax + b; \qquad x = \frac{y - b}{a}.$$

In the case of creating column-wise summations, the column crossings are marked and then summed column-wise. For row-wise summations, the row crossings are marked and then summed row-wise.

6.4.4 Non Maxima Suppression

There are many possible implementations of non maxima suppression. The basic non maxima suppression algorithm is described in Section 4.2.2, however, it is possible to enhance it in many ways.

I extended the basic version by combining the bounding boxes with a lower confidence with the bounding boxes with a higher confidence. The bounding boxes are ordered by confidence from the highest to the lowest.

The bounding box with the highest confidence is taken, let us call it the main bounding box. Then, starting from the highest confidence, one bounding box after another is taken and its percentage of intersection over union (IOU) with the main bounding box is computed. If the percentage of IOU is greater than 0.5 the bounding box is combined with the main bounding box in the following manner:

$$\begin{aligned}
M_L &= \frac{\mathrm{cnf}(M) \cdot M_L + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i) \cdot BB_{iL}}{\mathrm{cnf}(M) + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i)};\\
M_T &= \frac{\mathrm{cnf}(M) \cdot M_T + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i) \cdot BB_{iT}}{\mathrm{cnf}(M) + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i)};\\
M_R &= \frac{\mathrm{cnf}(M) \cdot M_R + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i) \cdot BB_{iR}}{\mathrm{cnf}(M) + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i)};\\
M_B &= \frac{\mathrm{cnf}(M) \cdot M_B + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i) \cdot BB_{iB}}{\mathrm{cnf}(M) + \mathrm{iou}(M, BB_i) \cdot \mathrm{cnf}(BB_i)},
\end{aligned}$$

where M_L, M_T, M_R and M_B are the left, top, right and bottom coordinates of the main bounding box. The same notation is used for the bounding box BB_i that is being combined. The confidence value of a bounding box is represented as $\mathrm{cnf}(\cdot)$ and $\mathrm{iou}(\cdot, \cdot)$ is the intersection over union value.

In other words, the bounding box's coordinates are computed as a weighted average of the main bounding box and the bounding box that is being combined. The weight of the combined bounding box is given by its confidence and its IOU with the main bounding box.

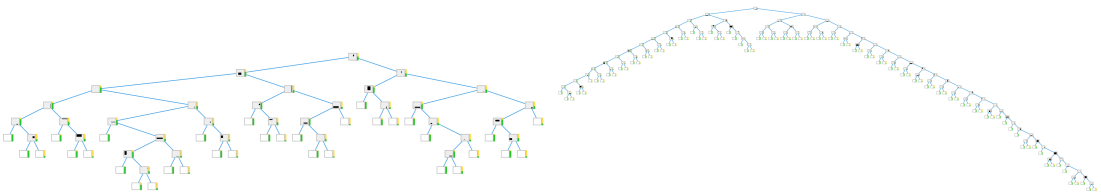
Each time the main bounding box is combined with some other one, the search is carried out again from the start because the coordinates of the main bounding box changed, making it possible for a bounding box with former insufficient IOU to now overlap enough to be combined.

Chapter 7
Discussion and Analysis

From the foregoing experiments, it is obvious that the randomized forest detector is not able to catch up with the WaldBoost detector. For now. Here, I will sum up the possible drawbacks of the RF detector and try suggesting solutions and future work.

The fact that extremely randomized forests yield better results (Section 5.3.1) is probably caused by the unbalance of the trees in the forests. When the required minimum split of training samples is not high, the extremely randomized trees do not grow very deep, however, they are very balanced (Figure 7.1a) because the feature selection does not take into account the measure of the split. Also, the trees in the forest grow very differently, which results in partitioning the sample space more than in the case of more precise feature selection.

When the feature selection process tries more features, it has a higher chance of better splitting the training set. This, however, causes the trees to get unbalanced (Figure 7.1b), which has a negative impact on the detector's speed and performance. The trees then grow deep in the very left and very right branch and end up being actually almost cascade classifiers. Also, the growth process of one tree assimilates the process of another tree because similar splits are achieved. Then, when those trees are put together, it results in having less space partitions than in the case of random split selection and the desired effect of combining the trees' output is not achieved.



a) Balanced tree. b) Unbalanced tree.

Figure 7.1. Tree samples from trained RFs.

The balance of the trees was improved by incorporating the possibility of making decisions in all nodes of a tree and not just in leaves. Nevertheless, it is not sufficient.

From the entropy gain criterion, one can derive that the split is balanced if the numbers of samples of all classes in the parent node are equal. Therefore, a possible improvement could be to weigh the training samples in a way to simulate equiprobable distribution of classes in each node.

Probably, a different feature selection would make an important difference. By studying the position and type of the features chosen during training (Figure 7.2 and 7.3), it can be noticed that the position below a car predominates as well as the 9-bit feature with

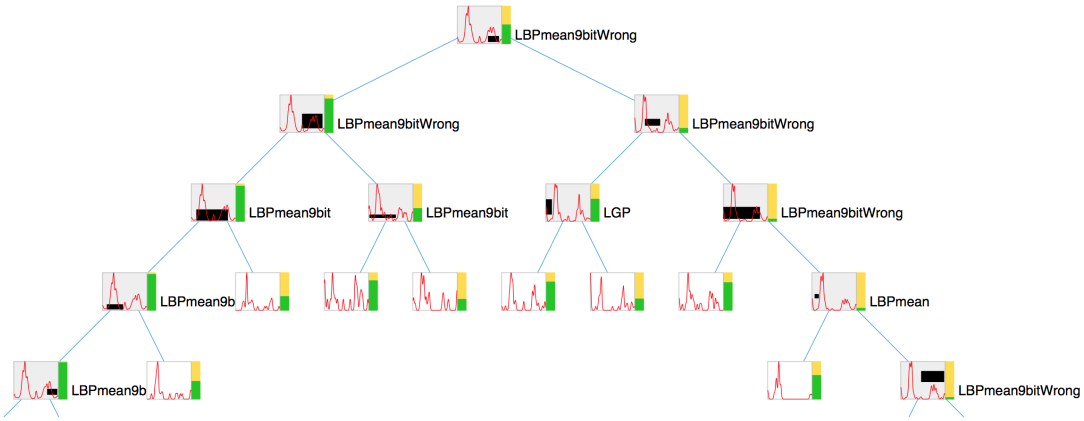
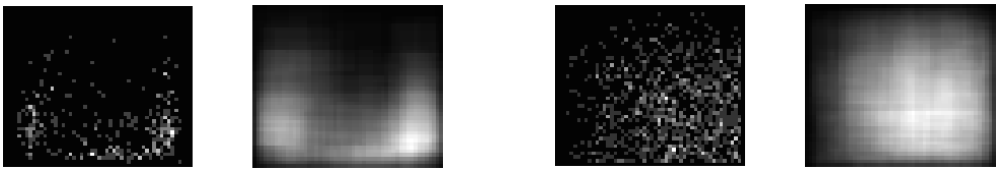


Figure 7.2. Feature selection in a tree. Feature position (black) within a detection window (grey) and distribution of rotation angles (red). The green/yellow bar represents the Bayesian probability of car/background images in a node.

mean. This fact forestalls the car rotation detection since the shadow below a car is similar for all car rotations.

It is interesting to look at the feature selection in detail. When we study Figures 7.3b and 7.3c, it is obvious that the feature selection algorithm that chooses the best feature out of many actually forces the features outside the position of the cars. Whereas in the case of random selection, the features cover the whole window better. This fact could also have an impact on the detector's performance. Figure 7.3a is included only to show the positions of the cars in the windows.

a) Positive data samples.



b) Choosing out of 1000 features. c) Random selection.

Figure 7.3. The distribution of chosen features in the 52x44 window. The center points are shown in the left images, whereas on the right, the whole areas are shown. The lighter the area, the more features.

There also arises the question, whether a different type of features would make a difference. The suitability of LBP features for the task of car detection is questionable since their nature says they are texture-based. Whereas, cars are defined much better by their geometric proportions.

In Figure 7.4 we see that even though in the nodes there are still many car samples, the training algorithm was not able to find a feature (out of 1000 random) that would

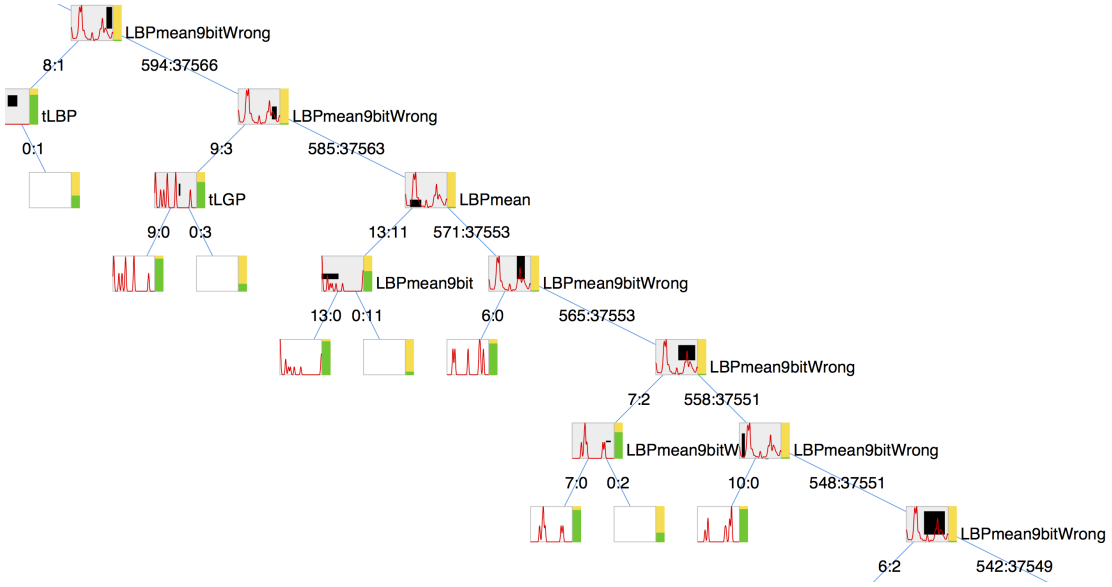


Figure 7.4. Feature selection when background samples dominate. The legend is the same as in Figure 7.2. The numbers show the number of car:background samples in a branch.

significantly split the training set. Although there are still about 500 car samples in the nodes, the best feature found was able to separate only about 10 of them from the background.

This problem was even more significant while I tried training the detectors on data set which had approximately 10%/90% car/background ratio. Soon, the features were not able to separate many cars from the huge amount of background samples and, in the last leaf of the branch, there were many cars left (for example 500 car images). When we think about that, we actually lost 500 cars from the training set, which, in the end, results in bad detector performance because the detector is never trained to recognize those cars. Using different type of features could probably partially resolve this problem.

The speed of the randomized forest detector (an the WaldBoost detector as well) showed that to achieve real-time performance, there is a need for a faster window evaluation then is possible with the detectors themselves. That is why we proposed the window pruning method based on LSD. The idea was to eliminate even more than one window by evaluating a single test. This was not achieved, however the proposed line integral image allowed for making one computation per detection window to decide whether the window can even contain a car.

The LSD-based window pruning does its job, nevertheless, it discards some positive detections. It can be partially caused by the computation of lines within a window because the proposed combination of line integral images is not flawless. The lines that cause problems in computation are displayed in red in Figure 7.5. Those lines get included in the column (row) integral images, because they intersect the row (column) boundaries. However, the whole line is not included in the line integral image, therefore, subtracting the numbers of intersections yields negative numbers.

To prove the method's functionality this was enough. However, in future use, a better concept would have to be proposed to avoid these inconsistencies.

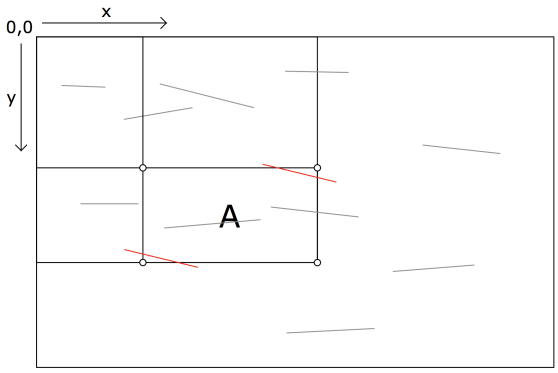


Figure 7.5. Error in line integral image computation (red lines cause problems).

A better performance of the detector could possibly be reached by improving even more the process of extracting background samples for training. As suggested before, the background samples could be tested for presence of horizontal lines as well and only samples with more than a certain number of horizontal lines would be used for training. It is obvious that during detection, the areas with an insufficient number of lines are skipped. However, they are still included in training, which, from this point of view, is actually useless.

Another obvious drawback of the randomized forest detector is the used non maxima suppression algorithm. In comparison with the algorithm in the WaldBoost detector, this one is really simple. This problem is actually related to the fact that the detector is producing a fair amount of FPPW. To achieve high recall, the detector has many FPPW. And, when they are combined, the correct detections very often disappear. It results in missing many cars, having the combined bounding boxes wrongly placed or not having them at all. Thus, apart from improving the detector's FPPW, a more sophisticated method for NMS should be used.

This observation makes me suggest that each detector should, in fact, have its own non maxima suppression algorithm. Each detector produces its own (and different) detection windows, which can be differently distributed around the desired object – a car. My idea is to create a NMS algorithm, which would then be trained to suit each detector explicitly.

When we think about it, also the detection of a car's rotation angle could help with combining detection windows. It would not make sense to combine two detection windows that have completely different angles assigned. Nevertheless, as mentioned in Section 4.5.5, this would make the training more difficult and, thus, probably yield deeper trees and prolonged detection time.

However, the rotation angle could have a positive impact on the detector's performance as well. It might not be obvious, but the WaldBoost detector actually has an advantage because it learns three different groups of car poses. Each of those groups is much more specific by itself then when used altogether as one group, which could cause different (and more discriminative) feature selection in the case of WaldBoost. Therefore, incorporating a criterion of how well a feature splits the rotation angles to the feature selection criterion could yield better results.

The last improvement I propose is to tune the image scaling function. Right now, bilinear interpolation in the basic form is used to resample images. It could be further improved by running Gauss filtering on the image every time it is scaled. It would

smoothen the image and probably also improve the line segment detection, which could lead all the way up to enhancing the detection window pruning.

Even though many improvements were made, the randomized forest detector still would not achieve similar performance to the WaldBoost detector. However, considering the time available for this task and my experience, I could hardly compete with the WaldBoost detector, which has been developed in Czech Technical University in Prague for several years.

Nevertheless, we still have ideas for improvements and we believe that randomized forest or a forest in itself could have advantages over WaldBoost. Especially when we look at the process of classification, WaldBoost uses the same features for all windows, however in the case of a forest, different features can be used based on the branch the sample falls into.

Conclusions

We investigated the problem of car detection on roundabouts, which proved to be a challenging task due to the high variability of traffic situations and cars poses present.

I collected and annotated a new data set, which is composed of images extracted from video sequences. The video sequences were shot from a car while driving on a roundabout, therefore representing exactly the situation examined in this thesis. The annotations were made by hand on approximately every tenth image and then interpolated through the others. The dataset will be made public.

The principle of the WaldBoost detector was studied and experiments were performed on the Dejvicka and the KITTI data sets. Even though the boosted cascade classifier showed good results, they were insufficient for a future use in real applications. Thus, we decided to use the detector as a reference and propose a new solution.

We proposed a new car detector based on randomized forest. We studied the algorithm and designed the structure of the program. The basic algorithm was implemented and several improvements were made to the algorithm to improve its performance and speed. The detector was then tested against the reference WaldBoost detector.

To do the above mentioned, I needed to study many technical publications as well as manuals for MEX functions, C++, MATLAB and big data handling in C++ and others. I wrote and ran many scripts for data extraction, translation, training, testing and detection.

The performed experiments showed that the proposed randomized forest detector does not have the qualities of the WaldBoost detector. It reached similar miss rate to the WaldBoost detector, while having FPPW higher by one order of magnitude. Given the fact that the WaldBoost detector has been developed for several years, it could have been expected that the new randomized forest detector, created in three months, is going to have an inferior performance.

Nevertheless, we pointed out several flaws and interesting actualities of the randomized forest detector that were found during a deeper analysis. We believe that resolving those problems could lead to a significant improvement of the performance.

Unfortunately, time did not allow me to implement some suggested enhancements and perform further testing on other data sets. I believe that I will be working on this problem in the future and that we will be able to get closer to the state-of-the-art performance.

References

[1] V. Belle, T. Deselaers, and S. Schiffer. Randomized trees for real-time one-step face detection and recognition. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, Dec 2008.

[2] Claudio Caraffi, Tomáš Vojíř, Jiří Trefný, Jan Šochman, and Jiří Matas. A system for real-time detection and tracking of vehicles from a single car-mounted camera. *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 975–982, 2012.

[3] Ondřej Chum and Andrew Zisserman. An exemplar model for learning object classes. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[4] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *In Computational Learning Theory: Eurocolt '95*, page 23–37, 1995.

[5] Yoav Freund, Robert Schapire, and Abe N. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence 14.*, pages 771–780, 1999.

[6] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In A. Criminisi and J. Shotton, editors, *Decision Forests for Computer Vision and Medical Image Analysis*, Advances in Computer Vision and Pattern Recognition, pages 143–157. Springer London, 2013.

[7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[9] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282 vol.1, Aug 1995.

[10] Cheng-Hao Kuo and R. Nevatia. Robust multi-view car detection using unsupervised sub-categorization. In *Applications of Computer Vision (WACV), 2009 Workshop on*, Dec 2009.

[11] Jiří Matas and Tomáš Vojíř. Robustifying the Flock of Trackers. *16th Computer Vision Winter Workshop. Mitterberg, Austria*, February 2-4, 2011.

[12] Frank Moosmann, Bill Triggs, and Frederic Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Advances in Neural Information Processing Systems 19*, pages 985–992. MIT Press, 2006.

[13] Constantine Papageorgiou and Tomaso Poggio. A trainable object detection system: Car detection in static images, 1999.

[14] Bojan Pepik, Michael Stark, Peter Gehler, and Bernt Schiele. Occlusion patterns for object class detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[15] Robert Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Machine Learning*, pages 80–91, 1999.

[16] J. Shotton, A. Blake, and R. Cipolla. Contour-based learning for object detection. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 503–510 Vol. 1, Oct 2005.

[17] Jiří Trefný and Jiří Matas. Extended set of local binary patterns for rapid object detection. *Proceedings of the Computer Vision Winter Workshop*, 2010.

[18] Paul Viola and Michael J. Jones. Robust real-time face detection. *International journal of computer vision 57.2*, pages 137–154, 2004.

[19] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001.

[20] R.G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. LSD: A fast line segment detector with a false detection control. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(4):722–732, April 2010.

[21] Bo Wu and R. Nevatia. Cluster boosted tree classifier for multi-view, multi-pose object detection. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, Oct 2007.

[22] Wei Zheng and Luhong Liang. Fast car detection using image strip features. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2703–2710, June 2009.

[23] Zhenfeng Zhu, Yao Zhao, and Hanqing Lu. Sequential architecture for efficient car detection. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[24] Jan Šochman and Jiří Matas. Waldboost-learning for time constrained sequential detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 2. IEEE*, 2005.

Appendix A
Contents of the Attached CD

Path	Description
/thesis_LiborNovak.pdf	The text of this thesis.
/RF_detector/	The randomized forest detector Xcode project.
/RF_detector/carDetectorLN/	The randomized forest detector source codes.
/RF_detector/carDetectorLN.mexmaci64	A runnable MEX file for 64-bit OS X.
/Dejvicka_dataset/	The Dejvicka dataset.