



CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY

BACHELOR THESIS

Car Detection on a Mobile Robot by Fusing Visual and 3D Lidar Data

Erik Derner

{derneeri,svoboda}@cmp.felk.cvut.cz

CTU-CMP-2012-10

May 25, 2012

Available at

<ftp://cmp.felk.cvut.cz/pub/cmp/articles/svoboda/Derner-TR-2012-10.pdf>

Thesis Advisor: Tomáš Svoboda

The work was supported by EC project FP7-ICT-247870 NIFTi
and by the Czech Science Foundation under Project P103/10/1585.

Research Reports of CMP, Czech Technical University in Prague, No. 10, 2012

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: <http://cmp.felk.cvut.cz>

Car Detection on a Mobile Robot by Fusing Visual and 3D Lidar Data

Erik Derner

May 25, 2012

BACHELOR PROJECT ASSIGNMENT

Student: Erik D e r n e r

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Car Detection on a Mobile Robot by Fusing Visual and 3D Lidar Data

Guidelines:

A mobile robot that is being developed for urban search and rescue (USAR) is equipped with various sensors including an omnidirectional camera and rotating lidar sensor. Consider the lidar sensor calibrated with respect to the camera.

1. Improve the existing sliding window visual detector and experimentally verify its performance in harsh visual condition in the area of foreseen robot applications. Suggest a possible robustification of the detector by using data from other robot sensors.
2. Perform experiments on the data from the mobile robot.

Bibliography/Sources: Will be provided by the supervisor.

Bachelor Project Supervisor: Ing. Tomáš Svoboda, Ph.D.

Valid until: the end of the winter semester of academic year 2012/2013


prof. Ing. Vladimír Mařík, DrSc.
Head of Department




prof. Ing. Pavel Ripka, CSc.
Dean

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Erik D e r n e r

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Detekce automobilů na mobilním robotu s využitím kamery a laserového dálkoměru

Pokyny pro vypracování:

Mobilní robot vyvíjený pro úlohu záchranných prací v zastavěných oblastech je vybaven rozličnými senzory pro snímání okolí. Mimo jiné všesměrovou kamerou a otáčivým laserovým dálkoměrem.

Uvažujte laserový dálkoměr zkalibrovaný s kamerou.


1. Navrhněte vylepšení existujícího vizuálního detektoru a experimentálně ověřte jeho omezení pro detekci automobilů za velmi obtížných podmínek panujících v zamýšlené oblasti nasazení robotu a navrhněte vhodné vylepšení s přihlédnutím k možnému využití dat i z jiných senzorů než je kamera.
2. Výsledek experimentálně ověřte na reálných datech z mobilního robotu.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Tomáš Svoboda, Ph.D.

Platnost zadání: do konce zimního semestru 2012/2013




prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry


prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 1. 2012

Prohlášení autora práce

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 25.5.2012

Derner

.....
Podpis autora práce

Acknowledgment

I would like to thank my advisor Tomáš Svoboda for his great guidance and help throughout this project. I would also like to thank Karel Zimmermann for the valuable advice and comments. I am grateful to the members of the CMP NIFTi team for their assistance with the robot experiments. Finally, I would like to thank my parents for their support during my studies.

Abstract

In this work, we focus on the combination of visual and depth data in object detection. We propose an algorithm for object detection using a sliding window technique and a Bayesian-like classifier based on random ferns with Haar features. We test the algorithm on the car detection problem using a mobile robot for urban search and rescue equipped with a camera and a 3D lidar. The detector finds cars in a scene and estimates their approximate orientation. We show that the fusion of the visual images and the 3D data significantly improves the detection performance in comparison with utilizing only the visual data. The detector is able to overcome poor data quality from one of the sensors in harsh conditions.

Abstrakt

V této práci se zabýváme detekcí objektů ve vizuálních a hloubkových datech. Navrhujeme algoritmus pro detekci objektů pomocí techniky posuvného okna a klasifikátoru založeného na Haarových příznamech a vycházejícího z bayesovské klasifikace. Algoritmus testujeme na úloze detekce automobilů pomocí mobilního robotu vyvíjeného pro záchranné práce v zastavěných oblastech a vybaveného kamerou a laserovým dálkoměrem. Detektor vyhledává automobily a určuje jejich přibližnou orientaci. Experimentálně jsme ověřili, že kombinací obrazových a hloubkových dat se značně zvýší úspěšnost detektoru oproti používání výhradně obrazových dat. Algoritmus je schopen detekce i v náročných podmínkách, kdy je kvalita dat z jednoho ze senzorů snížena.

Contents

1	Introduction	4
1.1	Motivation	4
1.2	Previous work	4
1.3	Our approach	6
2	Algorithm	9
2.1	Random ferns with Haar features	9
2.1.1	Learning	14
2.1.2	Classification	17
2.1.3	Space and time complexity	18
2.2	Combination of visual and 3D data	20
2.2.1	Learning process of the car detector	20
2.2.2	Car detection using a sliding window	22
3	Experiments	25
3.1	Detection performance – sliding window experiments	27
3.1.1	Probability threshold	30
3.1.2	NMS cover ratio limit	30
3.1.3	Sliding window step	30
3.2	Classification performance – correct class estimation	35
3.3	Benefits of the visual and 3D data fusion	35
4	Conclusion	40
A	Data acquisition	44
A.1	Extraction of images and point clouds from robot data files	44
A.2	Projection of the 3D point cloud to the visual image	45
A.3	Projection output	47
B	Implementation	49
C	Enclosed CD	51

Chapter 1

Introduction

Object detection is one of the most important tasks in the field of computer vision. We concentrate on the car detection problem using a mobile robot for urban search and rescue, shown in Figure 1.1.

1.1 Motivation

The mobile robot is a part of the European Union integrated project NIFTi – Natural human-robot cooperation in dynamic environments [3]. The robot is designed to be used in critical situations such as car accidents in tunnels. We focus on the car detection problem, which should help the robot operator to quickly find cars in a scene. We propose an algorithm for car detection in a scene using data from the robot’s camera and 3D lidar. The detector also estimates an approximate orientation of the car. This is useful to determine the areas to access the car or view into the car, see Figure 1.2.

1.2 Previous work

There are many approaches to object detection. In this section, we present an overview of related work. Contrary to the majority of the papers in this field, we have to deal with rather sparse point clouds. Additionally, we have no generalized 3D model of the target object (car).

One of the efficient methods to find free-form 3D objects in point clouds is proposed in [7]. A global model description based on oriented point pair features is built in the learning stage and the model is matched locally using a fast voting scheme in the classification stage. The point pair features describe the relative position and orientation of two oriented points. This method is

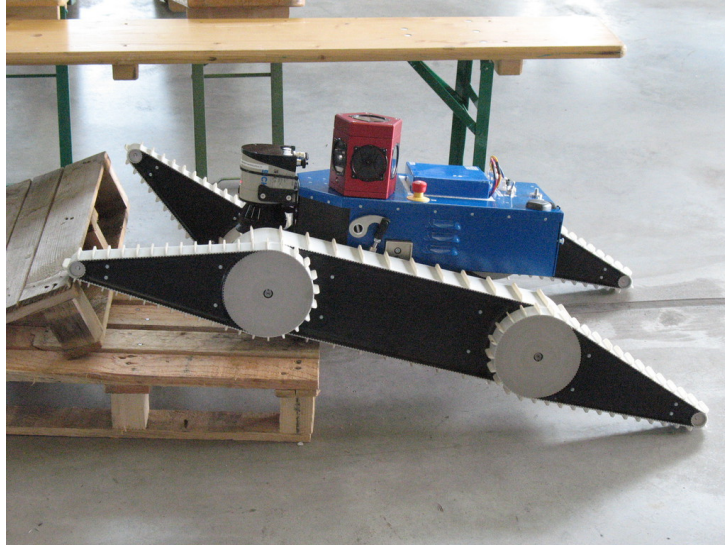


Figure 1.1: NIFTi BlueBotics mobile robot equipped with an omnidirectional camera and a 3D lidar [3].

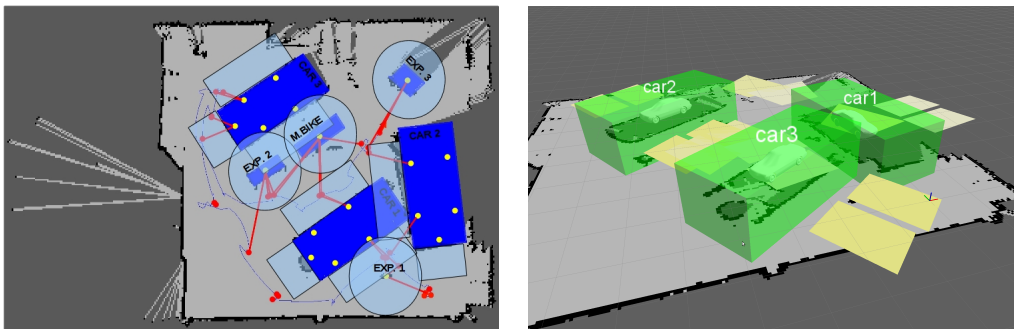


Figure 1.2: A motivation for the estimation of the approximate car orientation. The yellow points and rectangles denote the areas of access or view into the car [3].

fast and able to deal with sparse point clouds, but it requires an accurate 3D model of the target object, which is unsuitable for our task.

The works involving the combination of the visual and 3D data have recently gained popularity since the Kinect device appeared on the market. A recent work [9] concentrates on labelling objects in 3D scenes reconstructed from RGB-D (color+depth) image sequences. The sliding window detectors trained on various object views are utilized to assign class probabilities to pixels in every RGB-D frame. These probabilities are projected into the reconstructed 3D scene and integrated using a voxel representation. An efficient inference on a Markov random field over the voxels is performed, combining cues from view-based detection and 3D shape, to label the scene.

Another work concerning RGB-D data [4] on people tracking by a mobile robot is based on the clustering of the scene by using 3D information in conjunction with a reliable classifier based on histograms of oriented gradients, which is a method based on support vector machines, to identify people among these clusters. Nevertheless, both methods [9] and [4] take profit from tracking, which is not applicable in our problem because of the low frame rate of the 3D lidar.

The article [12] focused on detection and tracking in urban environments is the closest to our task. It deals with the fusion of the visual and 3D data and aims at people and car detection. The 3D points are classified by the boosted conditional random fields. The visual detector utilizes an extension of the implicit shape model, which learns a codebook of local descriptors from a set of labeled images and uses them to vote for centers of the detected objects. The objects are tracked using a Kalman filter with multiple motion models. The fusion of the data from the camera and the 3D lidar is however performed by the tracking module, which is unapplicable in our problem.

Although the paper [10] deals only with 3D data acquired by an autonomous mobile robot, it is very close to our work in other aspects. The detector described in this work is based on Haar features and utilizes 3D point clouds projected to an image plane. Objects are learned using classification and regression trees and the Gentle AdaBoost algorithm.

1.3 Our approach

We have not found a method that would be directly suitable for our task. Consequently, we propose a novel method based on our previous experience.

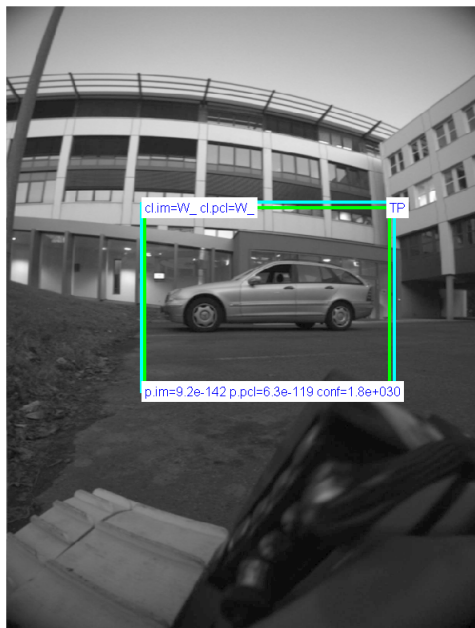
In [6], we found out that classification using *random ferns* is fast and efficient and we wanted to try it also on the car detection problem. Let us briefly explain the concept. Assume we have a set of randomly generated

binary features, i.e., features with two possible values – 0 and 1. These features are evaluated in each window. The pairs of image pixel intensities are an example of binary features; in our work, we use Haar feature pairs. We want to estimate a probability model from labeled training samples – simply said, which evaluation of the binary features is ‘typical’ for each class. As will be explained later, neither calculating the joint probability for all binary features, nor assuming complete independence of all binary features is appropriate. A random ferns-based classifier estimates the probability model by making a compromise between these two ‘extremes’ – by modelling only some of the dependences between the binary features. We use a modified version of the algorithm originally described in [11] for the classification of each window.

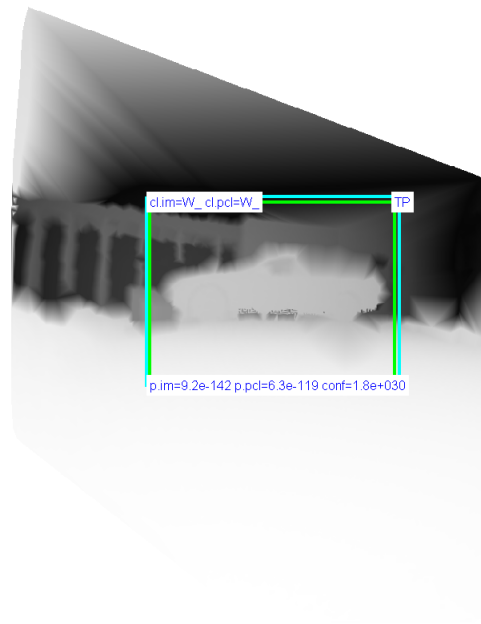
The proposed method is also based on our previous work on the sliding window alignment in car detection [5], which was intended to extend the work [15]. We improve the random ferns-based classifier that is used only for the visual data in [5] by extension for the 3D data and enhancing its performance by means described in the following text.

In this work, we use both visual images from a camera and 3D points from a lidar to make the detector more robust in the context of the car detection problem in harsh visual conditions. The visual and 3D representations of a scene are processed separately, so that the detector is resistant to failure of one of the sensors. The detection process in the visual and 3D data is in principle the same, because the 3D points are projected to the image plane and an interpolated depth map is constructed. Besides the detection, we also estimate an approximate orientation of a detected car, see Figure 2.1.

The detection is based on a sliding window algorithm. The concept is simple: a sliding window is moving over the query image. An occurrence of a car in a certain orientation is tested in each window using a random ferns-based classifier with Haar features. A sample result of the car detection is in Figure 1.3.



a



b

Figure 1.3: Example of the car detector output projected on a visual image (a) and on an interpolated depth image (b). The light blue rectangle represents the ground truth, the green rectangle indicates the position of the car estimated by the detector. This window was correctly classified as a car in the West orientation by both the visual and the 3D classifier.

Chapter 2

Algorithm

In this chapter, we describe the algorithm for car detection in visual images and 3D data using the *sliding window* method.

The main concept is that a rectangular sliding window is moving over a query image and in each position, the most likely class, i.e., one of the eight approximate car orientations, see Figure 2.1, is estimated. In case that the detector is unsure about the decision, it states that there is no car in the current window.

We take advantage of 3D point clouds to extend the visual information from a camera. The point clouds are mapped to 2D grayscale depth images with an identical point of view as the camera. These images are handled the same way as the visual images, i.e., using the same algorithm. See Section A.2 for more details.

In this work, the term *detector* is used for the ‘high level’ algorithm, which serves to find cars and determine their orientation in a query image using the sliding window technique on both the visual and 3D data. The detector is described in Section 2.2. On the other hand, the term *classifier* refers to the ‘low level’ algorithm that assigns each window to one of the eight classes. It is based on *random ferns* [11] with *Haar features* and described in detail in the following section.

2.1 Random ferns with Haar features

Haar features

We have chosen Haar features as the lowest-level descriptors of a window. We use five different types of Haar features, see Figure 2.2. These features of various sizes are ‘placed’ into the window on different positions. The grayscale image intensities are summed in the black and white areas of a

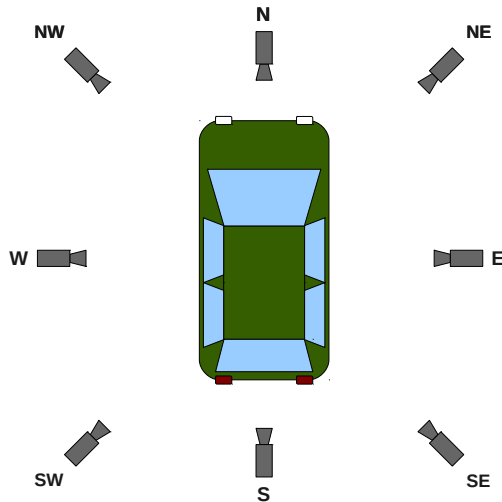


Figure 2.1: Orientations of a car that are distinguished by the classifier – eight approximate positions of a camera.

single feature. We take the absolute value of the difference of the black-area-sum and the white-area-sum as a Haar feature *response*.

We use Haar features instead of pixel intensities used in the original work [11]. The experiments we performed in [5] showed that Haar features outperform pixel intensities in the car detection problem.

The Haar features are randomly grouped into pairs. That allows binary evaluation of each pair, i.e., which response of the two Haar features is higher. *Evaluation* of a set of Haar features on a certain image window is a result of the comparison of the responses in all feature pairs. Hence, for a set of N feature pairs, there are 2^N possible evaluations.

Probability model estimation

The classifier attempts to classify each window into one of the eight *classes*, i.e., car orientations, as shown in Figure 2.1. The classification is based on a probability model, which assigns each possible evaluation of a window the probability of the presence of a car in a certain orientation. More accurately, it is the conditional probability of a class given the evaluation of a particular window. The window is assigned to the class with the highest probability. Please note that the window is assigned to one of the classes even though there is no target object at all. Such a classification should be discarded by the detector if the probability of the most likely class is lower than a given

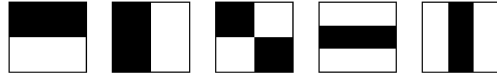


Figure 2.2: Types of Haar features used by the classifier.

threshold. The selection of the particular threshold value will be discussed later.

To enable this classification, we need to estimate the probability model. Calculating the joint probability of all the N feature pairs would require extreme amount of memory, as 2^N values – probabilities of each of the possible evaluations of the set of all feature pairs – would have to be stored for each class. Assuming independence of all binary features would rapidly reduce the necessary amount of memory, but it completely ignores the correlation between features [6].

Random ferns

The main idea of the random ferns-based classifier consists in grouping the feature pairs to arbitrary sets of equal size, called *ferns*, and calculating a probability model from evaluations of these sets, assuming independence of the sets. The grouping is again based on random selection. Schematic visualisation of ferns is shown in Figure 2.3.

The sets are in fact not strictly independent, but this modification preserves performance while using reduced amount of memory and reasonable computational time.

In the learning stage, the probability model is calculated for each of the ferns instead of the whole set of all feature pairs. That means, for each fern, the conditional probability of each class given each evaluation is estimated from the training data set.

Assuming the independence of the ferns, the conditional probability of a class given the evaluation of a particular window is simply calculated as a product of the conditional probabilities of a class given the evaluation of each fern. For a given evaluation of a window, these conditional probabilities are calculated for all classes and the window is assigned to the class with the highest product. Illustration of the classification using ferns is in Figure 2.4.

In the following sections, we will describe the learning and classification process in more detail.

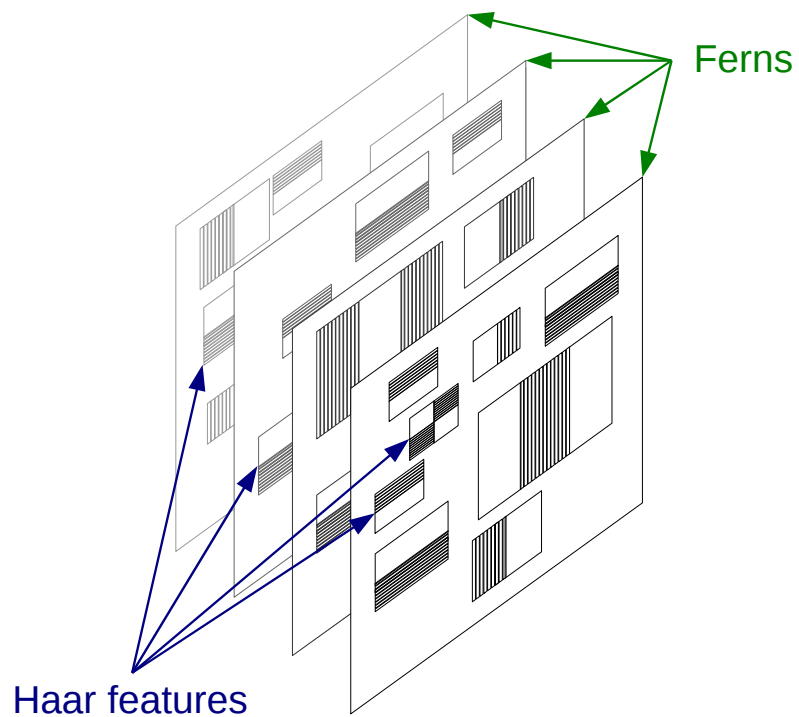


Figure 2.3: Schematic visualisation of a ferns-based classifier. Ferns can be described as frames with the same size as the window. All ferns have the same number of Haar features, but the Haar features are different in each fern – various positions, sizes and types.

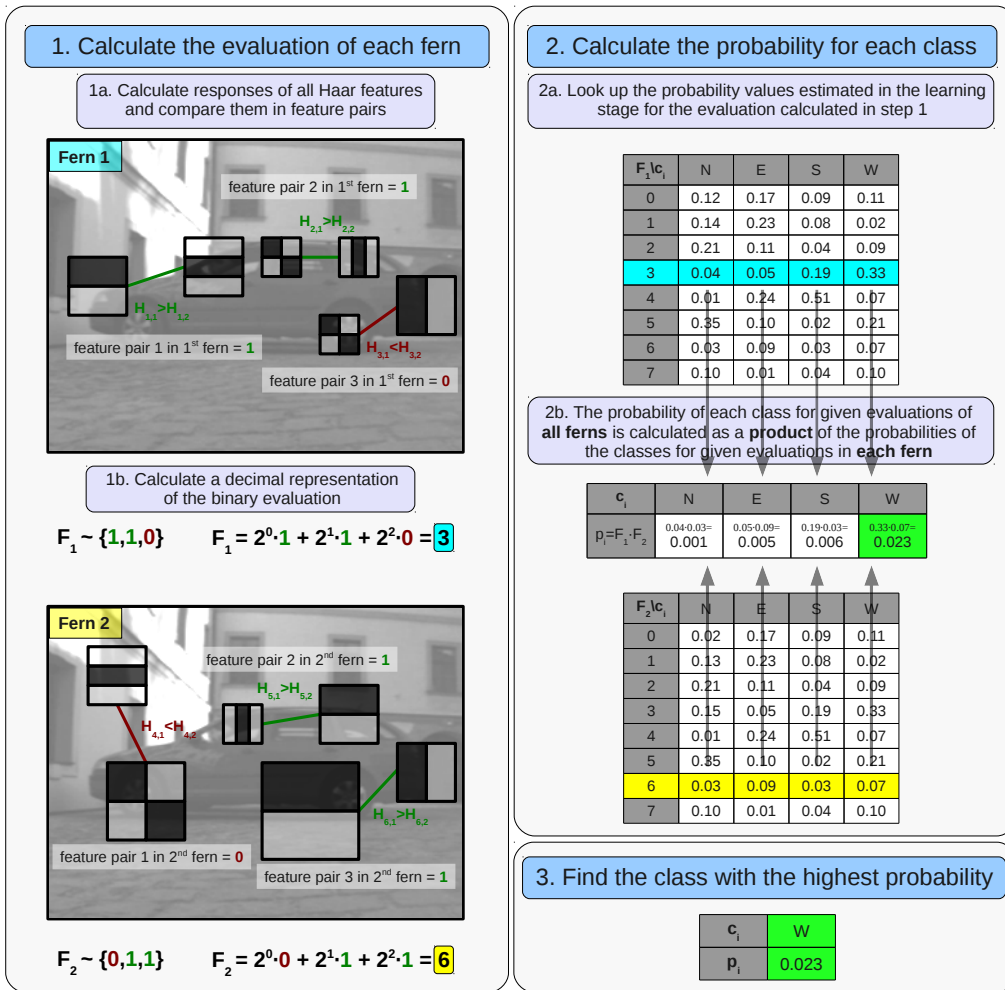


Figure 2.4: Classification of a window with a car in the West (W) orientation. Visualisation of the classification process for four classes (N, E, S, W), two ferns and three feature pairs in each fern. The values are only illustrative.

2.1.1 Learning

At the beginning of the learning stage, M ferns are generated, each containing S pairs of Haar features, i.e., total $M \cdot S = N$ feature pairs per window. The type, location and size of the features is picked completely at random. These ferns remain the same throughout the learning and classification process.

After that, the classifier is learned on labeled training samples. The Haar features are evaluated on each sample and the number of occurrences of each particular evaluation in each fern is computed. At the end, the occurrences are converted to probabilities.

The evaluation of the Haar features is computed from an *integral image*. This allows very effective calculation of the responses, as explained in the following text.

Integral image

An integral image, also known as a summed area table, is an algorithm for quick and efficient calculation of the sum of values in a rectangular subset of a grid.

The value at any point (x, y) in the integral image is the sum of intensities of all the pixels above and to the left of (x, y) , inclusive:

$$I(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y'), \quad (2.1)$$

where $i(x', y')$ is the pixel intensity at (x', y') in the original image.

The integral image is computed in a single pass over the input image using the following formula:

$$I(x, y) = i(x, y) + I(x - 1, y) + I(x, y - 1) - I(x - 1, y - 1), \quad (2.2)$$

where $I(-1, y) = I(x, -1) = I(-1, -1) = 0$.

Once the integral image is computed, the sum of a rectangular area can be calculated from the following formula, using the notation in Figure 2.5:

$$\sum_{\substack{A(x) < x' \leq C(x) \\ A(y) < y' \leq C(y)}} i(x', y') = I(C) + I(A) - I(B) - I(D). \quad (2.3)$$

The calculation is performed in a constant time, with only four array references and three basic arithmetic operations [14].

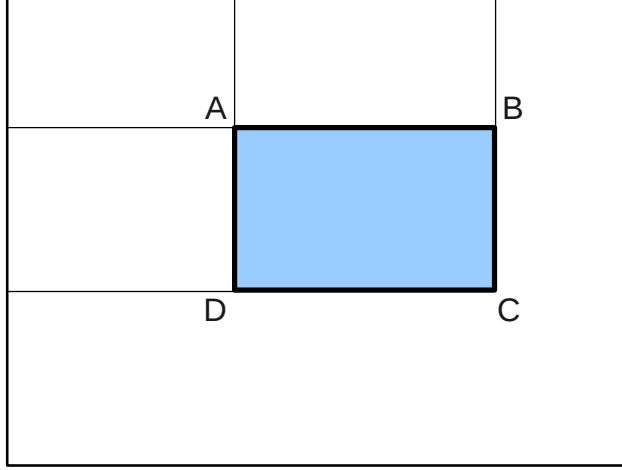


Figure 2.5: Integral image – fast calculation of the sum of the values in a rectangular area using the Equation 2.3.

Probability model

The goal of the classifier is to assign a given window to the most likely class. This can be formulated as finding

$$\hat{c}_i = \arg \max_{c_i} P(C = c_i | f_1, f_2, \dots, f_N) , \quad (2.4)$$

where C is a random variable representing the class and $f_j, j = 1, \dots, N$, is the binary evaluation of the j -th feature pair.

We calculate the conditional probability of a class given the evaluation using the Bayes' formula:

$$P(C = c_i | f_1, f_2, \dots, f_N) = \frac{P(f_1, f_2, \dots, f_N | C = c_i) P(C = c_i)}{P(f_1, f_2, \dots, f_N)} . \quad (2.5)$$

Since the denominator is only a scaling factor, independent from the class, and since we assume uniform prior probability distribution, the problem reduces to finding

$$\hat{c}_i = \arg \max_{c_i} P(f_1, f_2, \dots, f_N | C = c_i) . \quad (2.6)$$

The algorithm follows a Semi-Naive Bayesian approach, since it models only some of the dependencies between the feature pairs. The conditional

probability of a particular evaluation for a particular class is calculated using the formula

$$P(f_1, f_2, \dots, f_N | C = c_i) = \operatorname{argmax}_{c_i} \prod_{k=1}^M P(F_k | C = c_i), \quad (2.7)$$

where f_j is the evaluation of the j -th feature pair, c_i is the class that the current evaluation is assigned to and F_k is the evaluation of the k -th fern. As we use labeled samples for learning, the class c_i is known in the learning stage.

During the learning stage, we determine the values necessary for evaluation of the right side of Equation 2.7. The evaluation of a feature pair f_j , consisting of Haar features $H_{j,1}$ and $H_{j,2}$, is determined as follows:

$$f_j = \begin{cases} 1, & r(H_{j,1}) > r(H_{j,2}) \\ 0, & \text{else} \end{cases} \quad (2.8)$$

The function $r(H_{j,q})$ returns the response of a Haar feature. We calculate the sum S_b of the image pixel intensities that correspond to the black areas of the Haar feature and the sum S_w of the intensities that correspond to the white areas (Figure 2.2). The response of a Haar feature is the absolute value of the difference of S_b and S_w .

Evaluation of a fern F_k is a set of evaluations of the Haar features that form the k -th fern,

$$F_k \approx \{f_{\sigma(k,1)}, f_{\sigma(k,2)}, \dots, f_{\sigma(k,S)}\}, \quad k = 1, \dots, M, \quad (2.9)$$

where $\sigma(k, j)$ is a random permutation function that returns integers in range from 1 to N . The feature pairs are divided into ferns this way.

We store the evaluation as a decimal number using the formula

$$F_k = \sum_{j=1}^S (2^{(j-1)} | f_{\sigma(k,j)} = 1) = \sum_{j=1}^S 2^{(j-1)} f_{\sigma(k,j)}, \quad (2.10)$$

so that it can be easily used for indexing [11].

Model estimation from training samples

For each training sample from the class c_i , we calculate the evaluation of all ferns $F_k, k = 1, \dots, S$. For each evaluation F_k that is calculated on the current sample in the k -th fern, the value of N_{F_k, k, c_i} is increased by 1. (All of

these values are initially set to 0.) This way, we get the number of occurrences of each possible evaluation for each fern.

Finally, we calculate the probabilities from the number of occurrences using the formula

$$p_{F_k, k, c_i} = P(F_k | C = c_i) = \frac{N_{F_k, k, c_i} + 1}{\sum_{F_k=0}^{2^S-1} N_{F_k, k, c_i} + 2^S}. \quad (2.11)$$

These are the only probability values necessary for the classification. We add 1 to the numerator and 2^S to the denominator to overcome the issue that some of the ferns could return zero probability of a particular evaluation, which would result in total zero product despite possibly high probabilities in other ferns [11].

2.1.2 Classification

The classification of a window is based on the probability model estimated in the learning stage. Let \hat{p}_i be the probability of the most likely class \hat{c}_i for a particular window. For each class, the product of the conditional probabilities $P(F_k | C = c_i)$ is calculated. These values represent the probability of a particular evaluation of a particular fern given the class. The classifier selects the class with the highest product.

$$\hat{p}_i = \max_{c_i} \prod_{k=1}^M P(F_k | C = c_i) \quad (2.12)$$

$$\hat{c}_i = \operatorname{argmax}_{c_i} \prod_{k=1}^M P(F_k | C = c_i) \quad (2.13)$$

The classifier assigns the most likely class to each window, even though there is no target object at all. The detector, which receives the values \hat{c}_i and \hat{p}_i from the classifier, decides, whether the classification will be accepted or not. This is based on a probability threshold: If the probability \hat{p}_i of the most likely class is higher than a given threshold, the most likely class \hat{c}_i is assigned to the window. If the probability \hat{p}_i is lower than that threshold, the detector rejects the classification and states that there is no target object in the current window [11].

The selection of the threshold is ambiguous, there is in general no optimal value. Selection of a reasonable threshold depends on the data the detector is used on and on the desired TP/FP ratio. We discuss the selection of the threshold in Section 2.2.2.

2.1.3 Space and time complexity

Choice of the parameters

Now we discuss, which values are the optimal for the classifier parameters M and S , i.e., the number of ferns and the number of feature pairs per fern, respectively.

In the experiments, we use 50 ferns of size 11, i.e., 22 Haar features in each fern. That sums up to a total of 1100 features per window. The experiments showed that such number and size of ferns is quite optimal for our application.

A lower number of ferns yields worse results, while a higher number of ferns would require modifications of the probability model estimation process, as the probability values underflow the *double* data type precision.

Changing the fern size did not result in any significant improvement in the recognition performance either. The choice of the parameters was discussed also in [11] and the authors came to the conclusion that using 30–50 ferns with 11 binary features is optimal.

Space complexity

The major part of the memory is used for storing the probabilities p_{F_k, k, c_i} (Equation 2.11) necessary for the classification. Let T be the number of classes. M is the number of ferns and S is the number of feature pairs in each fern. The probabilities are stored in an array of $2^S \cdot M \cdot T$ elements.

Let's take the values used in our experiments as an example – $M = 50$ ferns, $S = 11$ feature pairs per fern, $T = 8$ classes. Assuming the double precision (8 bytes), we need $8 \cdot 2^{11} \cdot 50 \cdot 8 = 6\,553\,600$ bytes of memory to store the probability model.

Time complexity

Let us remind that M is the number of ferns, S is the number of feature pairs per fern, $N = M \cdot S$ is the total number of feature pairs and T is the number of classes. Let U be the number of training samples for each class, assuming it is the same for all classes for the simplicity. A window has a size of $w \times h$ pixels.

Learning The learning stage consists of the following steps:

1. **Calculate an integral image.** The integral image is calculated in a linear time with respect to the image size – $O(w \cdot h)$.

2. **Evaluate feature pairs.** The response of each Haar feature is computed in a constant time, as explained in Section 2.1.1. Evaluation of one feature pair is also performed in a constant time, since it is only a simple comparison of two values – responses. Therefore, all Haar features are evaluated in $O(N)$.
3. **Store the evaluation.** For each fern, the evaluation calculated in the previous step is converted to a binary number. The number of occurrences of this evaluation is increased for each particular fern.

This process is performed for each of the training samples from all classes, i.e., $T \cdot U$ times.

Also some other computations, such as the random generating of the Haar features and partitioning them into ferns at the beginning or the recalculation of the number of occurrences to probabilities at the end, are performed in the learning stage. However, these tasks are of a negligible significance in comparison to the process that is done with each sample.

Classification One of the most significant advantages of a ferns-based classifier is the fast classification. The classification of a window is performed in these steps:

1. **Calculate an integral image.** The integral image is calculated in a linear time with respect to the image size: $O(w \cdot h)$. This step is even not necessary in our implementation, because the detector calculates the integral image from the whole query image and passes the respective part, according to the sliding window position, directly to the classifier.
2. **Evaluate feature pairs.** As explained above, the evaluation is computed in a linear time with respect to the number of feature pairs – $O(N)$.
3. **Calculate probabilities for all classes.** Based on the evaluation, the product of M values is calculated for each of the T classes.
4. **Find the class with the highest probability.** Finding a maximum among the T classes is naturally performed in $O(T)$.

The calculation of the Haar feature responses is very fast and the other tasks require a rather small amount of operations, so that the whole classification process is done in a split second.

2.2 Combination of visual and 3D data

In this section, we describe the learning and classification process of the detector, which uses the output of the classifier described in the previous section at its core.

We make the detector more powerful by using not only the visual data from a camera, but also 3D point clouds from a lidar. The 3D point clouds are transformed to the viewpoint of the camera and interpolated. This results in a 2D grayscale depth image with the same resolution as the visual image. Thus, the same algorithm may be used both for visual and 3D data. For a detailed description of the transformation procedure, see Section A.2.

In Section 2.1.1, we described the learning procedure of a classifier. The complete learning process of the detector is a bit more complex task, hence we will describe the complete learning procedure of the detector on a higher level now.

In the following Section 2.2.2, we will describe the detection process using a sliding window.

2.2.1 Learning process of the car detector

The learning process consists of two stages: *training stage*, where many randomly generated classifiers are learned on a training data set, and *testing stage*, where the best classifier is selected. This helps select the configuration of the randomly generated ferns, i.e., the types, sizes and positions of the Haar features, which suits the best the desired application.

The detector is learned on both the visual and 3D data. There are two types of the detector:

- **Detector with separate classifiers.** The image classifier and the 3D classifier are learned fully separately. That means that the procedure described in the following text is run twice, once for the image classifier and once for the 3D classifier. The *image classifier* is learned on visual images from a camera and the *3D classifier* is learned on interpolated grayscale depth images generated from 3D lidar point clouds. The image classifier is then used to classify only visual images and the 3D classifier evaluates depth images.
- **Detector with a combined classifier.** Only one classifier is used for classification of both the visual and 3D data. The combined classifier is in principle the same as the separate classifiers, the only difference is that it is learned both on the visual and 3D samples.

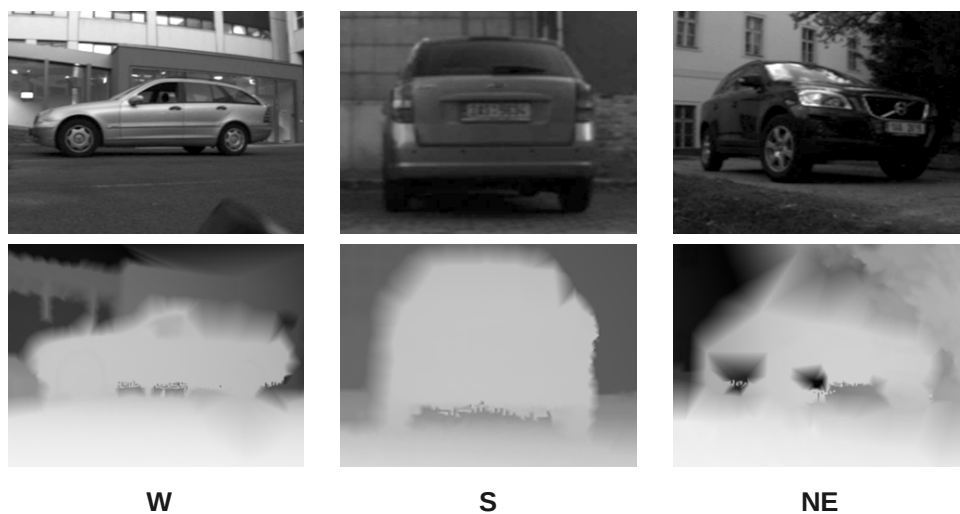


Figure 2.6: Training samples with cars in different orientations: W – West, S – South, NE – North East. Visual images are in the first row, interpolated depth images are in the second row.

Training stage

As the Haar features are generated completely at random, more classifiers are trained and the classifier that suits the best the car detection application is selected in the testing stage.

Each classifier is learned on labeled training samples, i.e., cars with known orientation. Examples of training samples are in Figure 2.6. The classifier learning process is described in detail in Section 2.1.1.

Testing stage

The selection of the best classifier is based on the performance of a classifier on a testing set. The performance is measured by the recognition rate on labeled samples, i.e., the ratio of the samples assigned to the correct class. To prevent an overfitting of a classifier, only the samples that were not used in the previous stage of the learning process should be used for testing the classifier performance. Hence we suggest splitting the set designated for learning into two parts.

There is one more task that is performed during the testing stage – storing the data for a future probability threshold calculation. The probability

threshold is used to distinguish windows with a car from windows without a car in the classification stage. As the samples are labeled, we are able to determine for each sample whether it was assigned to the correct class or not. For each sample, we store the probability \hat{p}_i of the most likely class, as described in Section 2.1.2. A set of the probability values of the correctly classified samples and a set of the probabilities of the incorrect ones is stored, which helps us to select the optimal probability threshold in the classification stage.

2.2.2 Car detection using a sliding window

In the first stage of the classification process, the probability model estimated in the learning stage is used to assign each window to the most probable class. The windows, which probability is higher than a given threshold, are passed to the next stage, where a confidence value is calculated for each window. In the last stage, multiple responses in similar locations are removed using non-maxima-suppression.

Sliding window

A sliding window with a fixed aspect ratio moves over a query image by a given step in horizontal and vertical direction from the top left corner to the bottom right corner, so that all positions in the ‘grid’ given by the step are covered, see Figure 2.7. This process is repeated for different sliding window sizes. That enables the detector to find cars of various dimensions.

For each window, the most likely class \hat{c}_i is determined together with the probability \hat{p}_i of the most likely class by the algorithm described in Section 2.1.2. If the probability is lower than a given threshold, it is assumed that there is no car in the current window. This is performed for both visual and 3D data, each using ‘their’ classifier in the case of separate classifiers, or using the same classifier in the case of the combined classifier.

The selection of the threshold value is based on the means of the probabilities of the most likely classes in correct and incorrect samples determined in the learning stage (more details can be found in the end of Section 2.2.1). The default threshold value is calculated as a mean of these two means and may be manually adjusted for better performance. The threshold value is generally different for the visual and 3D data.

The windows with the probability over the threshold for the visual data, 3D data or both are passed to the following stage.

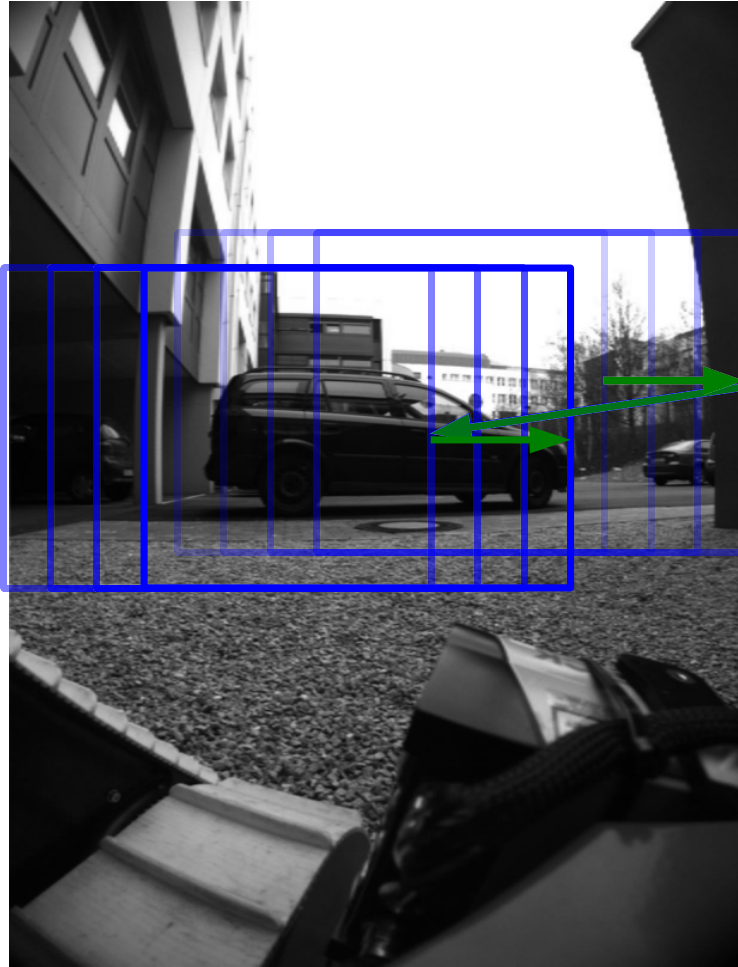


Figure 2.7: Visualisation of the progress of a sliding window on a query image.

Calculating confidence

In this stage, a *confidence* is calculated for each window. The confidence value is based on how much is the probability \hat{p}_i for the current window higher than the threshold value. If the threshold is exceeded for both the visual and 3D data, the two resulting confidence values are multiplied, so that the windows with a car detected both in the visual and 3D data are the strongest.

Selection of the strongest responses

As there may be, and usually are, multiple windows in similar positions, we use the non-maxima-suppression algorithm to eliminate these multiple responses.

The algorithm is quite simple. The window with the highest confidence is found and all windows that are in a similar position are removed. Near windows are determined by a *cover ratio*, which is calculated by division of the intersection area and the union area of the ‘strongest’ window and all other windows. The windows with the cover ratio higher than a given threshold are removed. This process is repeated until no more windows remain. The process has to be repeated, because there might be more than one car in the query image, and we would like to find all of them.

Chapter 3

Experiments

We performed various experiments to evaluate the car detector on the data from a mobile robot. We concentrated on the detection performance, i.e., how many cars are found in query images using the sliding window method, and on the classification performance – how many windows with cars are classified to the correct class (orientation). We also compared the separate classifiers with the combined classifier. The effect of the change of various parameters, such as number of ferns, sliding window step and non-maxima-suppression cover ratio limit, was also explored.

Generating samples

We were contending with the lack of data throughout all the experiments. The data acquisition process was rather complicated, because, as far as we know, data sets of both visual and depth data with cars are not commonly available. We had to acquire the data ourselves, using a mobile robot – Figure A.1. Because of the robot malfunction, we were not able to acquire as much data as would be appropriate. We also encountered difficulties with the 3D lidar. The results are thus affected by the sparsity of the training data, but the detector could be meaningfully evaluated on such a data set anyway. We suppose that a larger training data set would help achieve significantly better results.

To alleviate the lack of the training data at least a little, we extend the data set by generating samples with some ‘noise’ in the ground truth rectangle size and position, which also makes the detector more robust. The change of the size and position of the ground truth rectangle is about 5 % with respect to the ground truth rectangle dimensions.

The cars in the training samples are surrounded by a bounding box, which includes at least 10 % of the background on each side, while retaining the



Figure 3.1: Bounding boxes of the samples. If the car is ‘wide’, margins are added to the tight car bounding box to form 10 % of the resulting sample image width on each side. If the car is ‘high’, the margins are added in the vertical direction, again approximately 10 % of the sample image height, both on the top and on the bottom. The margins from the tight car bounding box in the other direction occupy more than 10 % on each side to retain the fixed aspect ratio.

fixed aspect ratio – see Figure 3.1. Another examples of the training samples are in Figure 2.6.

Data sets

In the experiments, we used three data sets: training set, testing set and evaluation set. As explained in Section 2.2.1, the training set is used to learn many different randomly generated classifiers, the testing set serves to select the best one among these and the classifier is finally evaluated on the evaluation set. The number of images in each class is in Table 3.1.

To ensure a fair evaluation, the training, testing and evaluation sets are disjunctive in all experiments. Neither two same images, nor two same cars in an identical position on different images, appear in any two of the sets simultaneously. On the other hand, due to the lack of data, the same cars from slightly different views may be present in more than one set, but we tried to prevent this as much as possible.

Table 3.1: The number of cars in specific orientations in the data sets used in the experiments. The evaluation set in this composition is used only in the class estimation experiment described in Section 3.2.

Orientation	Training set	Testing set	Evaluation set
N	9	4	4
NE	4	2	2
E	4	4	3
SE	11	3	3
S	16	8	10
SW	11	3	5
W	6	3	4
NW	7	4	5

Parameters

If not stated otherwise, we used the classifier parameters that are summarized in Table 3.2.

Table 3.2: Parameters of the classifiers used in the experiments.

Number of ferns	50
Number of feature pairs per fern	11
Classes	{N, NE, E, SE, S, SW, W, NW}
Number of samples per class	100

The minimal cover ratio of two rectangles in the non-maxima-suppression was generally set to 1 %. The minimal cover ratio of a GT rectangle and a classified rectangle to be accepted as a true positive was 50 %. The step coefficient of the sliding window was set to 15.

3.1 Detection performance – sliding window experiments

The most important task of the car detector is to localize cars in a query image, so we concentrated on this in the first set of experiments. We performed experiments to test the effect of the probability threshold value, minimal

cover ratio for the non-maxima-suppression and sliding window step on the detection performance.

The evaluation set consists of 10 query images with 11 cars and 10 query images without cars. The image resolution is 616×808 pixels. We tested both the combination of the separate classifiers and the single combined classifier.

The positions and sizes of the sliding window are determined by the following parameters:

- **Step coefficient.** The coefficient gives the number of ‘uniformly distributed’ window positions in the horizontal direction. The actual step in pixels is calculated to meet the requirement of the number of positions of the sliding window in the horizontal direction. The step in pixels is the same in the horizontal and vertical direction.
- **Sizes.** The sliding window also varies in its size to be able to detect cars of various dimensions. In all the experiments, the sliding window starts at its ‘natural’ size, which is 320×240 pixels – this is the size of the training samples – and then is resized by 5 %: $8\times$ reduced, $16\times$ enlarged from the default 100 % by 5 % in each step. This range covers all cars in the data set from the robot used in our experiments and may be naturally adjusted if necessary.

The total number of windows in all positions and scales per image is approximately 3500.

Ground truth is known for the query images. Each window, for which the detector states that there is a car, is evaluated as correct – true positive (TP) – if the cover ratio, i.e., the quotient of the intersection and the union of the ground truth rectangle area and the window rectangle area, is at least 50 %. Otherwise it is incorrect – false positive (FP). The correctness of the class (orientation) is not tested in this experiment, we concentrate on this in the experiment described in Section 3.2.

An example of the output of the detector using the combination of the visual and 3D classifier is shown in Figure 3.2.

Note Let’s assume that the query image is divided into four equal quarters by three horizontal lines. We attempt to detect the cars only in the two quarters in the middle, i.e., in the second and third part. It is given by the robot view that the cars never appear in the top or bottom quarter of the image, which is evident for instance from Figure 3.2. By this modification, we save a half of the computation time.

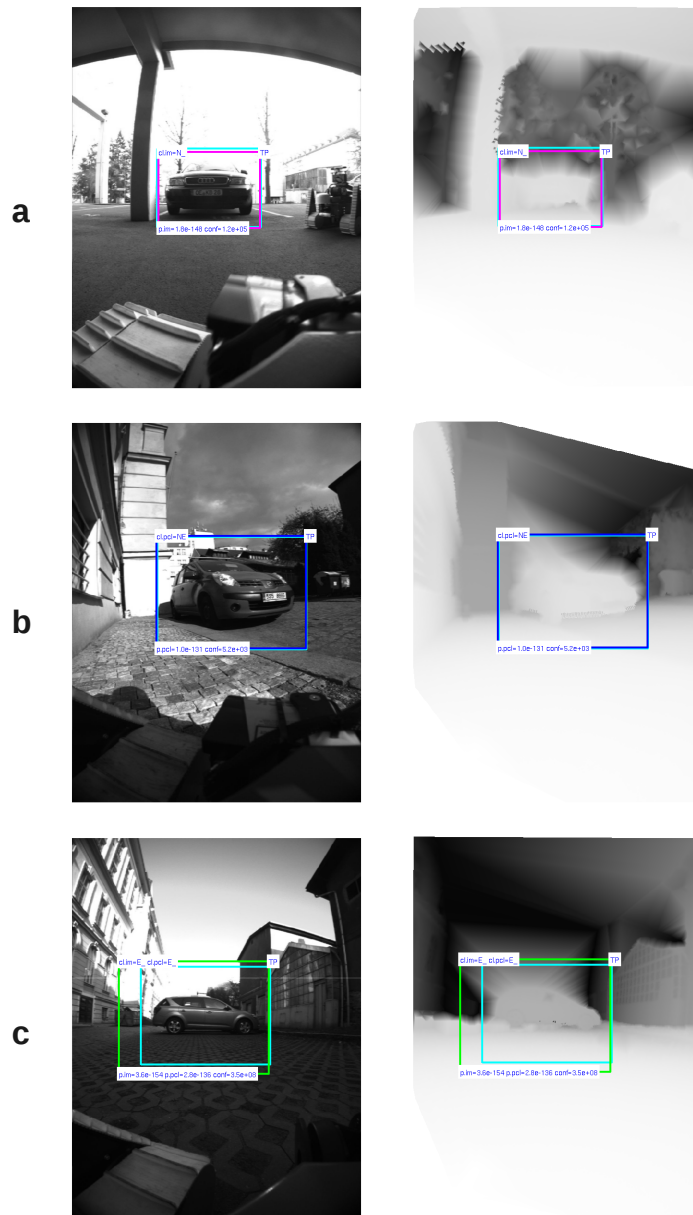


Figure 3.2: Example of the results of the detector on the data from the mobile robot. Visual images are in the left column, interpolated depth images are in the right column. The light blue rectangle represents the ground truth. Legend: a – car found only by the visual classifier (magenta rectangle), b – car found only by the 3D classifier (blue rectangle), c – car found by both the visual and 3D classifier (green rectangle). The classifiers were also correct in the estimation of the orientation in all cases.

3.1.1 Probability threshold

At first, we calculate the ROC curves to have a general overview of the detector performance. We tested the performance for the combination of the separate classifiers and for the single combined classifier for different probability thresholds. As can be seen in Figure 3.3, a combination of two separate classifiers – a visual classifier learned on visual data and a 3D classifier learned on 3D data – is more efficient than a combined classifier that is learned on both the visual and 3D data.

In this experiment, the TPR value never reaches 1, even if the probability threshold is set to 0. Some of the cars are never found, because the non-maxima-suppression removes the true positive windows that have a lower confidence than a false positive window. For an example, see Figure 3.4.

The decline in the performance of the separate classifiers at approximately $\text{FPR} = 0.6$ (Figure 3.3) is caused by the following: One TP window is overridden by a FP window, which was detected only by the visual classifier so far, but once the probability threshold is exceeded even for the 3D classifier. As the confidences from the both classifiers are multiplied, the non-maxima-suppression selects the FP window with a higher confidence. The TP window has a lower confidence, because it is still detected only by the visual classifier. The relevant images are shown in Figure 3.5

3.1.2 NMS cover ratio limit

Now we test the effect of the minimal cover ratio for the non-maxima-suppression algorithm. We set a fixed probability threshold, based on the results of the previous experiment, and evaluate the detector performance for various minimal cover ratio values. It is obvious from Figure 3.6 that the detection performance is the best for the lowest values of the minimal cover ratio.

3.1.3 Sliding window step

We also evaluate the effect of the sliding window step on the detection performance. Again, we set a fixed probability threshold, based on the first experiment. Figure 3.7 shows that the classifier achieves the best performance with 15 window positions in the horizontal direction when using the separate classifiers and with 30 window positions for the combined classifier.

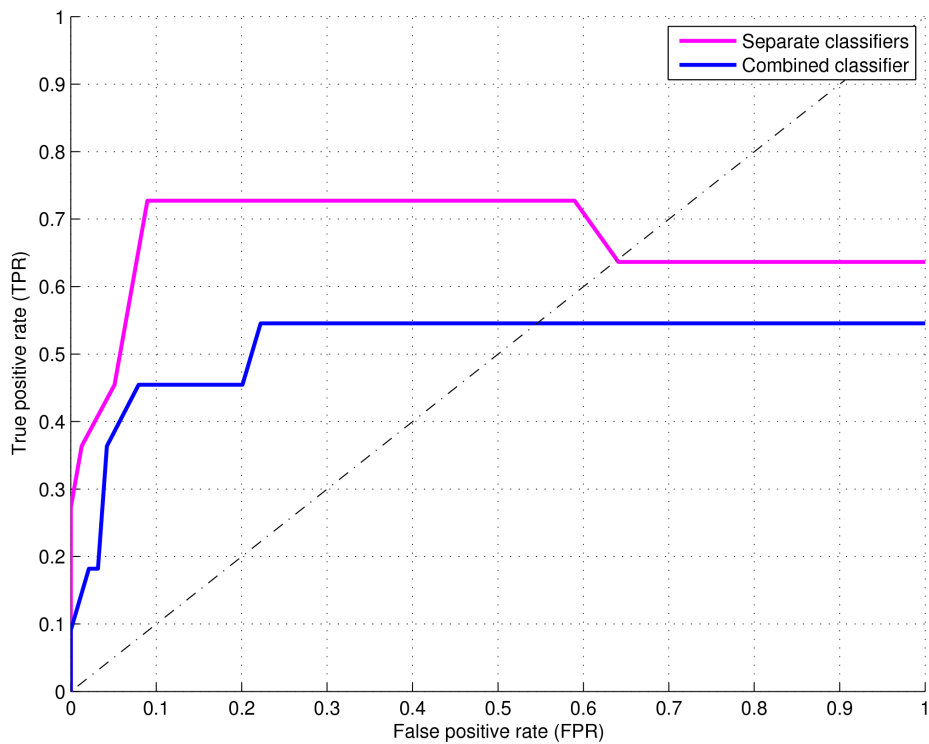


Figure 3.3: Comparison of the performance of the separate classifiers and the combined classifier in the sliding window experiment. The total number of positive windows is the number of cars in the data set, the total number of negative windows is the number of all detected FP windows after non-maxima-suppression in the data set for the probability threshold set to 0.

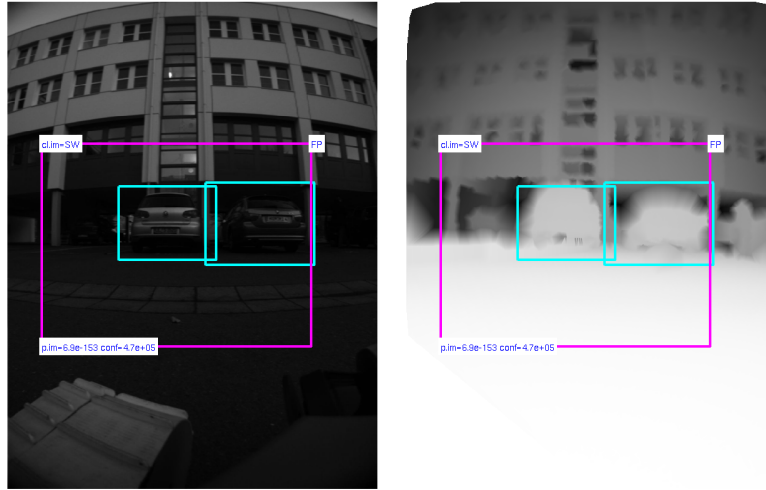


Figure 3.4: An example of detection failure. The true positive windows are overridden by a false positive window with a higher confidence.

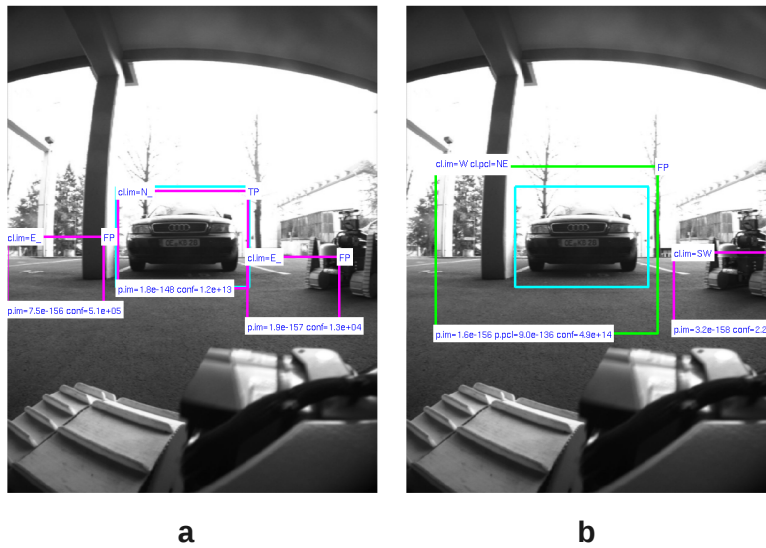


Figure 3.5: Detection failure after decreasing the probability threshold. Legend: a – the true positive window (along with two false positive windows) is detected by the visual classifier, b – the true positive window detected by the visual classifier is overridden by a false positive window detected by both classifiers, which multiplies their confidences.

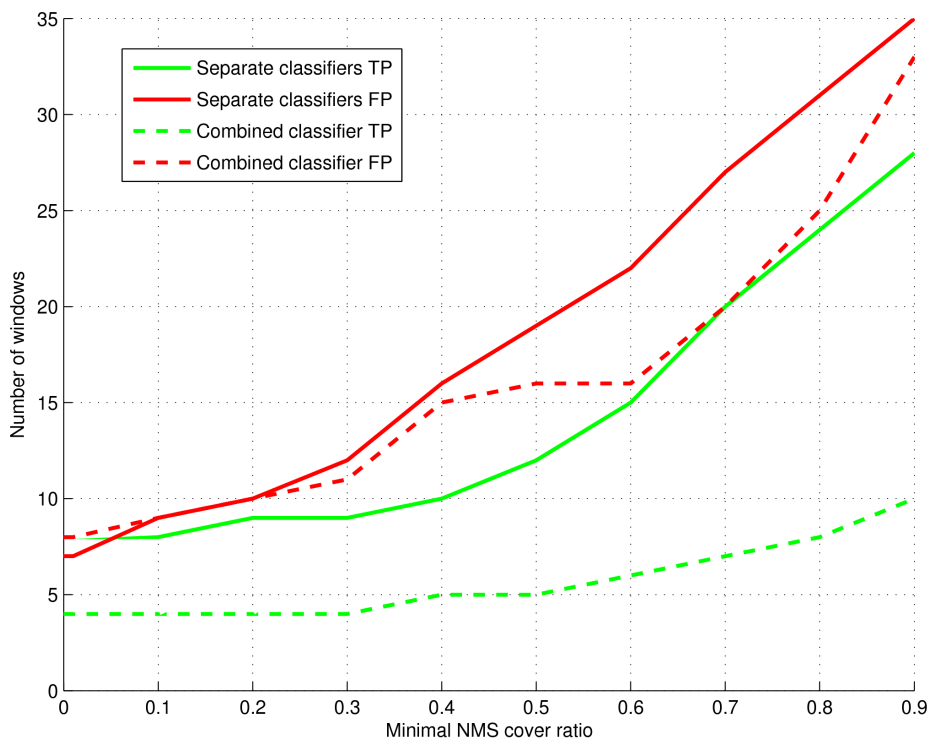


Figure 3.6: Effect of the minimal cover ratio value for the non-maxima-suppression on the performance of the separate classifiers and the combined classifier in the sliding window experiment. The y -axis shows the absolute count of TP and FP windows in the data set. The TP value may be higher than the number of cars in the data set – one car may get multiple TP ‘hits’ for higher minimal cover ratio values.

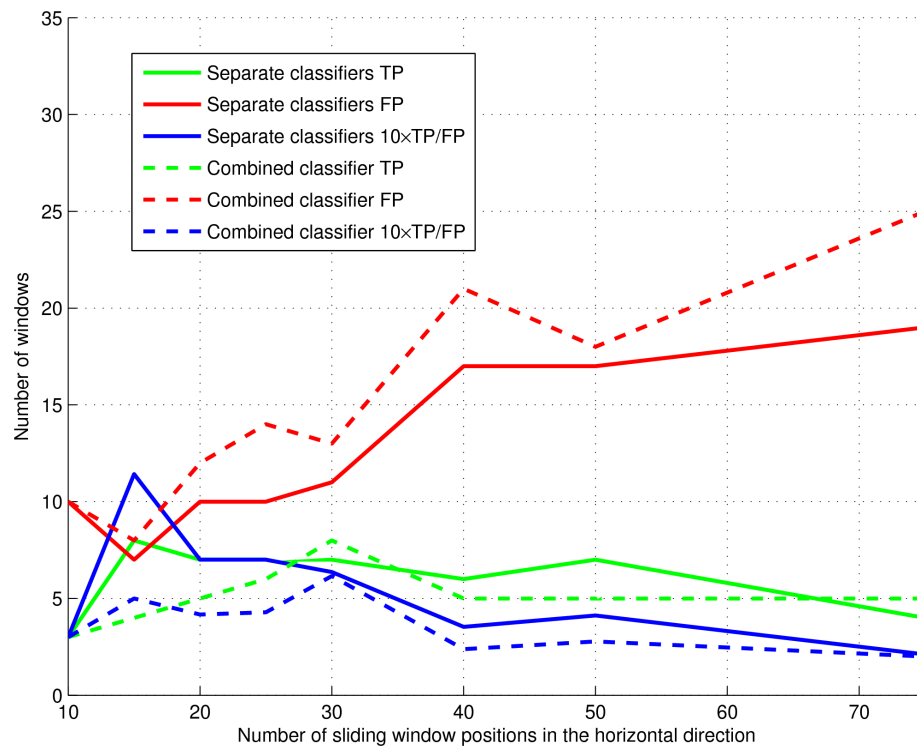


Figure 3.7: Effect of the sliding window step on the performance of the separate classifiers and the combined classifier in the sliding window experiment. The y -axis shows the absolute count of TP and FP windows in the data set.

Table 3.3: Summary of the overall performance of the classifiers in the class estimation experiment. The recognition rate indicates how many samples were assigned to the correct class.

Classifier	Recognition rate (%)
Separate visual classifier	50.5
Separate 3D classifier	82.5
Combined classifier	59.6

3.2 Classification performance – correct class estimation

The experiments described in this section concern the estimation of the approximate orientation and the comparison of the separate 3D and visual classifiers with the single combined classifier.

The evaluation was performed on a data set of 100 samples for each class in the evaluation set. The samples were generated by the same process as the training and testing samples, as described at the beginning of this chapter.

The classifiers were evaluated on the same type of data that was used for learning: the separate visual classifier was evaluated on visual images, the separate 3D classifier was evaluated on depth images and the combined classifier was evaluated on both visual and depth images.

The summary of the overall performance of the classifiers is in Table 3.3. Figure 3.8 shows the classification results of the visual classifier, Figure 3.9 shows the results of the 3D classifier. It is obvious from the table and from the graphs that the 3D classifier significantly outperforms the visual classifier.

The detailed results for the combined classifier are in Figure 3.10.

3.3 Benefits of the visual and 3D data fusion

One of the main points of this work is the fusion of the visual and 3D data. In this experiment, we compare the performance of the combination of separate classifiers and the single combined classifier with the performance of only the visual classifier on visual data and only the 3D classifier on 3D data. Figure 3.11 shows that utilizing the combination of separate classifiers for evaluation of both visual and 3D data significantly increases the detection performance.

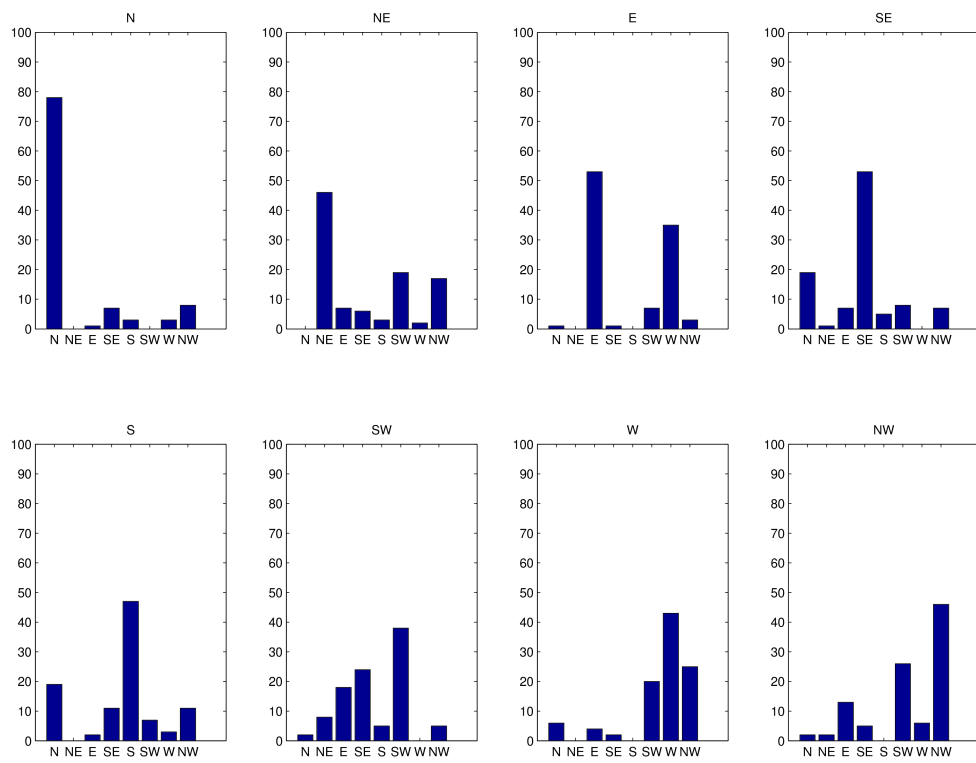


Figure 3.8: Detailed results of the visual classifier in the class estimation experiment. Each graph represents a (ground truth) class. The bars show, how many samples were assigned to which class (in %).

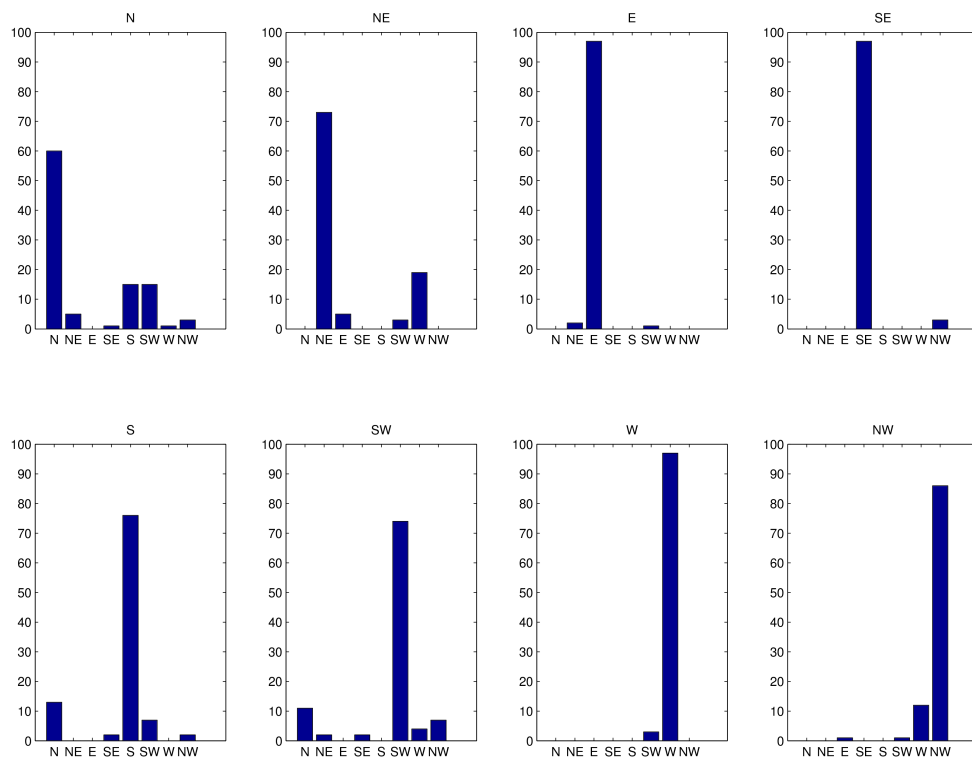


Figure 3.9: Detailed results of the 3D classifier in the class estimation experiment. Each graph represents a (ground truth) class. The bars show, how many samples were assigned to which class (in %).

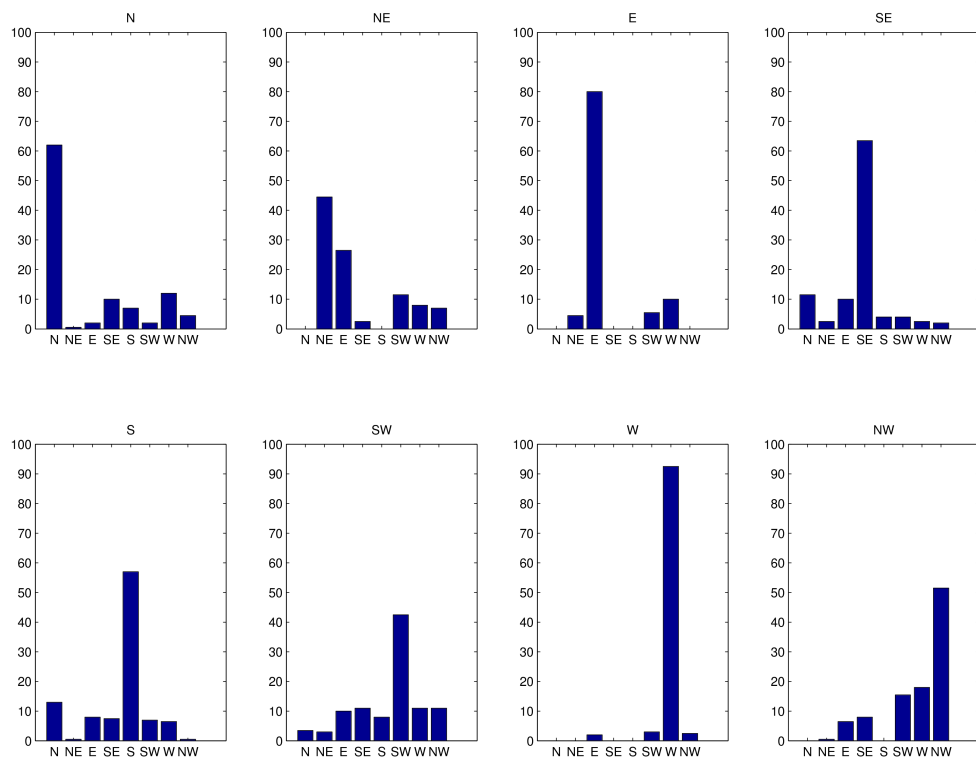


Figure 3.10: Detailed results of the combined classifier in the class estimation experiment. Each graph represents a (ground truth) class. The bars show, how many samples were assigned to which class (in %).

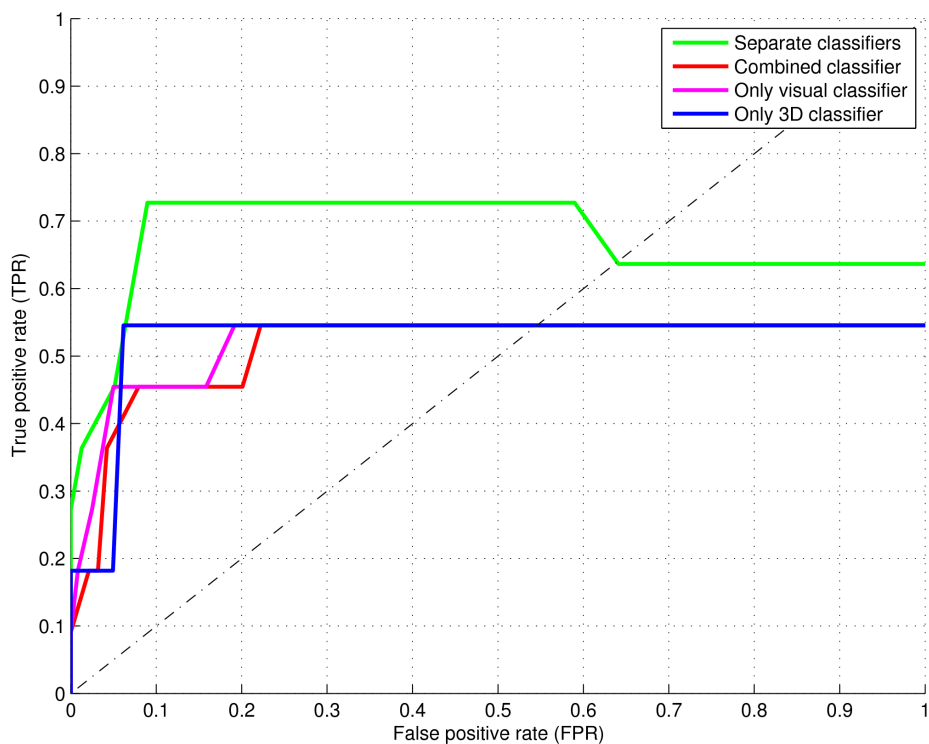


Figure 3.11: Comparison of the classifiers using combination of both visual and 3D data with utilizing only visual or only 3D data. The total number of positive windows is the number of cars in the data set, the total number of negative windows is the number of all detected FP windows after non-maxima-suppression in the data set for the probability threshold set to 0.

Chapter 4

Conclusion

We have proposed, implemented and tested an object detector utilizing both visual data from a camera and 3D data from a lidar. One of the intended applications, on which was the detector also tested, is the car detection problem.

Based on the experience we gained in our previous work [5] and [6], we have implemented a detector using the sliding window technique and classifiers based on random ferns with Haar features. We have designed two types of detectors utilizing both the visual and 3D data. The first one uses a combination of two separate classifiers – a visual classifier for the visual data and a 3D classifier for the depth data. The second one uses a single combined classifier for both visual and 3D data. The experiments showed that the former one has a better performance, but the latter one may be more efficient in some specific situations as well.

The detector finds positions of cars in a scene and estimates approximate orientations of the cars. The experiments confirm the expectation that the performance of a ferns-based classifier on the depth images is significantly higher than on the visual images. In our experimental data set, 82.5 % of the depth image samples were assigned to the correct class by the 3D classifier, while only 50.5 % of the visual image samples were classified correctly by the visual classifier.

We also tested the dependence of the recognition performance on the values of various detector parameters, such as sliding window step and minimal cover ratio for the non-maxima-suppression. It turned out that a smaller sliding window step, i.e., a more dense grid of windows, does not guarantee better recognition performance. A higher cover ratio limit increases the number of true positives, but also significantly increases the number of false positives. The detector was able to find 3 cars out of 11 with no false positives, 4 cars and one false positive or 8 cars and 7 false positives in our

evaluation data set consisting of 10 scenes with cars and 10 scenes without cars.

We have also created a tool for data acquisition from a mobile robot equipped with a camera and a 3D lidar. We had to create a data set for the evaluation of the detector ourselves, because we have not found any suitable data sets with both the visual and 3D representation of cars. We have scanned and processed scenes with cars in the outdoor experiments with the robot. Regrettably, we encountered various difficulties regarding the outdoor experiments, which disallowed us to gain the required amount of data. Besides that, a considerable part of the 3D data from the lidar was of a very poor quality. The lack of data had an adverse impact on the detector performance in the experiments.

Further work should involve particularly experiments on a larger and better quality data set to confirm the theory that the results are not very good because of the lack of data. An algorithm to reduce the number of false positives, in other words, a better way to determine the car locations in a scene than the probability threshold, should increase the performance of the detector. Using richer 3D descriptors like surface normals and curvatures may improve the performance as well.

Bibliography

- [1] Documentation – ros wiki, May 2012. <http://www.ros.org/>.
- [2] The new nifti robot platform – nifti extranet, May 2012. <http://www.nifti.eu/news/the-new-nifti-robot-platform/>.
- [3] Nifti extranet, May 2012. <http://www.nifti.eu/>.
- [4] Filippo Basso. Rgb-d people tracking by detection for a mobile robot. Master thesis, Universita degli Studi di Padova, Padova, Italy, 2011. <http://tesi.cab.unipd.it/35111/1/tesi.pdf>.
- [5] Erik Derner and Tomáš Svoboda. Random ferns for sliding window alignment in car detection. Research report, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, February 2012.
- [6] Erik Derner, Tomáš Svoboda, and Karel Zimmermann. Random ferns for keypoint recognition, image matching and tracking – implementation and experiments. Research Report CTU–CMP–2010–16, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, October 2010. <ftp://cmp.felk.cvut.cz/pub/cmp/articles/svoboda/Derner-TR-2010-16.pdf>.
- [7] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 0:998–1005, 2010. <http://campar.cs.tum.edu/pub/drost2010CVPR/drost2010CVPR.pdf>.
- [8] Vladimír Kubelka and Tomáš Svoboda. Nifti lidar-camera calibration. Research Report CTU–CMP–2011–15, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, December 2011. <ftp://cmp.felk.cvut.cz/pub/cmp/articles/svoboda/Kubelka-TR-2011-15.pdf>.

- [9] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Detection-based object labeling in 3d scenes. In *IEEE International Conference on on Robotics and Automation*, 2012. http://www.cs.washington.edu/homes/kevinlai/publications/lai_icra12.pdf.
- [10] Andreas Nuchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. Accurate object localization in 3d laser range scans. *ICAR 05 Proceedings 12th International Conference on Advanced Robotics 2005*, pages 665–672, 2005. http://kos.informatik.uni-osnabrueck.de/download/icar2005_1.pdf.
- [11] Mustafa Özuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(3):448–461, 2010. <http://cvlab.epfl.ch/publications/publications/2010/OzuysalCLF10.pdf>.
- [12] Luciano Spinello, Rudolph Triebel, and Roland Siegwart. Multiclass multimodal detection and tracking in urban environments. *I. J. Robotic Res.*, 29(12):1498–1515, 2010. http://www.rec.ri.cmu.edu/fsr09/papers/FSR2009_0035_3cce6b436f1c0c55f716131b3e2cb631.pdf.
- [13] Ranjith Unnikrishnan and Martial Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. Technical report, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 2006. http://www.cs.cmu.edu/~ranjith/lcct_files/lcct_doc_1.1.zip.
- [14] Wikipedia. Summed area table — wikipedia, the free encyclopedia, May 2012. http://en.wikipedia.org/w/index.php?title=Summed_area_table&oldid=489324679.
- [15] Karel Zimmermann, David Hurych, and Tomáš Svoboda. Improving cascade of classifiers by sliding window alignment in between. In G. Sen Gupta, Donald Bailey, Serge Demidenko, and Dale Carnegie, editors, *Proceedings of the Fifth International Conference on Automation, Robotics and Applications*, pages 196–201, Private Bag 11 222, Palmerston North, New Zealand, December 2011. School of Engineering and Advanced Technology, Massey University, Massey University. <ftp://cmp.felk.cvut.cz/pub/cmp/articles/hurycd1/icara2011.pdf>.

Appendix A

Data acquisition

We have created a tool to easily acquire data for the detector from a mobile robot. In this chapter, we will describe the process of data acquisition using this tool.

A robot equipped with an omnicaamera and a 3D lidar, see Figure A.1, was used to obtain the visual and 3D data. The robot runs the Robot Operating System (ROS) [1]. Messages in the ROS system, including images from the omnicaamera and 3D point clouds from the lidar, are published on certain topics. To receive these messages, subscribers are registered to listen on the topics.

We use the ROS function `rosvbag` to record all messages published on certain topics into a single *bag file* during the outdoor data acquisition process. A recorded bag file may be later played, so that all the messages are published exactly the same way as when running on the robot in real time. We have created a tool to extract the visual and 3D data from a bag file as separate images and text files. Since the omnicaamera and the 3D lidar have different origin, the 3D data have to be transformed to the viewpoint of the single cameras of the omnicaamera. After that, various visualisations of a projection of point clouds to visual images are created, including interpolated 2D depth images that are used for learning.

A.1 Extraction of images and point clouds from robot data files

The visual and 3D data extraction from a bag file is performed by the Bash script `bag2imgpcl.sh`. The script takes a bag file pathname as an input argument and produces directories with images from the cameras 0–4 of the omnicaamera and point cloud files recorded in the bag file.

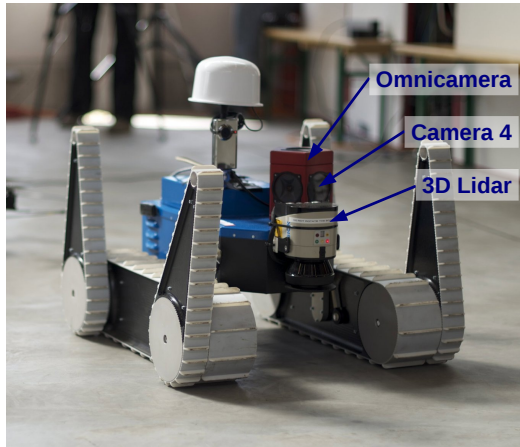


Figure A.1: Sensors of the NIFTi robot used for the car detection [2].

The cameras 0–4 are the cameras ‘around’ the omnicaamera, the camera on the top is the camera 5. We use almost without exception the images from the camera 4 as the data for our detector. The other cameras have the disadvantage that the view is occluded by robot parts, such as the flippers, or the image area is not fully covered by the 3D points because of the limited lidar range.

A.2 Projection of the 3D point cloud to the visual image

The 3D point cloud is projected to each of the cameras 0–4 of the omnicaamera. Transformation parameters are available [8] for the transformation from the lidar origin to the origin of the camera 4 and for the transformation from the omnicaamera origin to the origin of all cameras. The intrinsic parameters for the cameras are known and are the same for all cameras.

For the description of the projection algorithm, we use the following notation, based on [13]:

- Rotation matrices are denoted as $R_{\langle \text{from} \rangle - \langle \text{to} \rangle}$ and are of size 3×3 .
- Translation vectors are denoted as $\mathbf{t}_{\langle \text{from} \rangle - \langle \text{to} \rangle}$ and are of size 3×1 .
- The focal length in pixels is in the 2×1 vector \mathbf{f} .
- The principal point coordinates are in the 2×1 vector \mathbf{c} .

- The scalar α is the skew coefficient. It defines the angle between the x and y pixel axes.
- The vector \mathbf{k} of size 5×1 contains image distortion coefficients (radial and tangential distortions).

Using these parameters, we are able to project the point cloud to the viewpoint of all cameras in the following steps:

1. Transform the original 3D points \mathbf{X}_L in the point cloud from the lidar origin to the camera 4 origin.

$$\mathbf{X}_{C_4} = \mathbf{R}_{L-C_4} \mathbf{X}_L + \mathbf{t}_{L-C_4}$$

2. Transform points \mathbf{X}_{C_4} from the camera 4 origin to the omnicaamera origin.

$$\mathbf{X}_O = (\mathbf{R}_{O-C_4})^T (\mathbf{X}_{C_4} - \mathbf{t}_{O-C_4})$$

3. Transform points \mathbf{X}_O from the omnicaamera origin to the origin of individual cameras.

$$\mathbf{X}_{C_i} = \mathbf{R}_{O-C_i} \mathbf{X}_O + \mathbf{t}_{O-C_i}, \quad i = 0, \dots, 4$$

4. Let $\mathbf{X}_{C_i} = [X, Y, Z]^T$. Calculate the normalized pinhole projection by division of X - and Y -coordinates by the Z -coordinate.

$$\mathbf{x}_n = \begin{bmatrix} X/Z \\ Y/Z \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

5. Let $r^2 = x_n^2 + y_n^2$. Calculate the tangential distortion vector \mathbf{d} .

$$\mathbf{d} = \begin{bmatrix} 2k_3x_ny_n + k_4(r^2 + 2x_n^2) \\ k_3(r^2 + 2x_n^2)2k_4x_ny_n \end{bmatrix}$$

6. Using the tangential distortion vector \mathbf{d} , calculate the normalized point coordinates including the lens distortion.

$$\mathbf{x}_d = (1 + k_1r^2 + k_2r^4 + k_5r^6) \mathbf{x}_n + \mathbf{d} = \begin{bmatrix} x_d \\ y_d \end{bmatrix}$$

7. Calculate the final pixel coordinates $\mathbf{x}_p = [x_p, y_p]^T$ of the 3D points in the viewpoint of the camera i .

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \alpha f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}$$

A.3 Projection output

The script produces various images with projection of the 3D points. Before the projection process can begin, two additional steps have to be performed.

- **Distance calculation.** For each point \mathbf{X}_L in the point cloud, we calculate its Euclidean distance from the lidar origin. This is necessary for the construction of a depth map.
- **Matching of images and point clouds.** On the robot, the rate of saving images is much higher than the rate of saving point clouds. That means that the corresponding images and point clouds have to be matched. Each image and point cloud has a time stamp, so the time-nearest image is found for each point cloud.

For each image from a camera matched with a point cloud, the script creates four images, see Figure A.2. The visual image (c) and the corresponding interpolated grayscale depth image (b) are used as input data for the car detector.



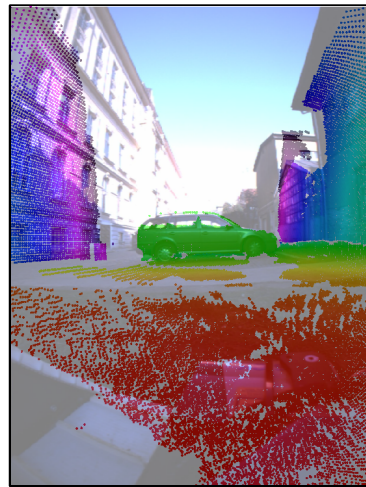
a



b



c



d

Figure A.2: Images produced by the projection tool: a – 3D points on a white background, visualized in a grayscale depth image, b – interpolated grayscale depth image, c – visual image that was time-matched with the point cloud, d – combined color depth image – both visual data and 3D points displayed in one image.

Appendix B

Implementation

We have implemented the random ferns-based detector in MATLAB version 7.8.0. Source codes are available on the enclosed CD in the `detector` directory in the following subdirectories:

- `ferns_detector` – core functions – learning and classification, sliding window algorithm, non-maxima-suppression etc.
- `exp_demo` – demonstration scripts – learning of the car detector on a sample data set, car detection in a query window
- `bag2imgpcl` – tool for the extraction of data from a robot and for the projection of 3D points to visual images – see Chapter A

Detector

In this section, we briefly describe the most important source codes in the `detector/ferns_detector` directory.

As explained in Chapter 2, the car detection consists of two main stages: learning and classification. At first, the classifier(s) of a single window have to be learned. The function `learn_ferns()` is used for this purpose. The classification of a window is performed by `classify_ferns()`.

The car detection process is controlled by the function `find_cars()`. At first, all windows are classified by the `eval_windows()` function, which runs the sliding window. In the next stage, the windows with the probability under the given threshold are discarded and the confidence is calculated for the rest using the `classify_windows()` function. The last stage is non-maxima-suppression performed by `nms()`, which selects the windows with the highest confidence and removes all windows that intersect with the ‘strongest’ windows by more than a given cover ratio limit. Detailed information can be found in the source codes.

Demonstration scripts

We have created demonstration scripts to show the detector functionality. There are four scripts in the `detector/demo` directory, which can be directly run ‘as they are’:

- `go_detect.m` – Detect cars in a given image using learned classifiers. The detected windows are shown in two figures, with the visual and the depth image as a ‘background’. This version utilizes the combination of separate classifiers. The filename of the query image can be specified in the `im_name` variable. By default, the learned classifiers used for the experiments described in Chapter 3 are loaded. It is also possible to learn the classifiers again, using the `go_learn.m` script. To use alternative classifiers, change the argument of the `load` function in the `Parameters` section. The probability threshold coefficients in `prob_threshold_im` and `prob_threshold_pcl` may be adjusted to achieve better results.
- `go_detect_combi.m` – The same as `go_detect.m`, but using a combined classifier.
- `go_learn.m` – Learning of separate classifiers. The learned classifiers are saved in a file named `model_<timestamp>.mat`.
- `go_learn_combi.m` – Learning of a combined classifier. The learned classifier is saved in a file named `model_combi_<timestamp>.mat`.

The scripts utilize the MEX function in the `detector/ferns_detector` directory. We provide the compiled binary for 32-bit Microsoft Windows platforms and 64-bit Unix platforms. If you are running another operating system, please make the binary yourself by executing the following command in MATLAB in the `detector/ferns_detector` directory:

```
mex computeHaarInt_mex.cpp
```

Note To avoid unnecessary reimplementations, some auxiliary codes from other authors from the CMP were used. The author is noted in all source codes.

Appendix C

Enclosed CD

The enclosed CD contains two directories: `detector` and `thesis`. The source codes, data and results are in the `detector` directory, the thesis PDF and TeX sources are in the `thesis` directory.

There are five subdirectories in the `detector` directory. The subdirectories `ferns_detector`, `exp_demo` and `bag2imgpcl` were described in Chapter B. The remaining two subdirectories are described in the following text.

data – The directories `cars_qvga_train` and `cars_qvga_test` contain visual and depth image samples with cars for the training and testing stage of the detector learning process, as described in Section 2.2.1. The directories `sliding_eval` and `sliding_eval_combi` contain visual and depth images with ground truth bounding boxes of cars used to evaluate the detector. The files in these directories were used in the sliding window experiments described in Section 3.1.

results – The results of the experiments described in Section 3.1. The directories named `*prob` contain the results of the experiments testing the effect of the probability threshold on the detection performance. The `*cover` directories contain the results of the experiments with various non-maxima-suppression minimal cover ratio values. The effect of the sliding window step coefficient on the performance is captured in the `*step` directories. The subdirectories of these directories are named by the particular parameter values. In each directory, there are `*.mat` files with the same names as the directories. These files contain a ‘summary’ of the detection results on the whole data set in two variables: `nTP` – number of true positive windows – and `nFP` – number of false positive windows.