Robust Real-Time Number Sign Detection on a Mobile Outdoor Robot

Sebastian A. Scherer* Daniel Dube* Philippe Komma* Andreas Masselli* Andreas Zell*
*Department of Computer Science, University of Tuebingen, Tuebingen, Germany

***Abstract*— We present the evolution of a highly-efficient system for visually detecting number signs in real-time on a mobile robot with limited computational power. The system was designed for use on a robot participating in the 2010 "SICK robot day" robotics challenge, in which the robot needed to autonomously find 10 number signs and drive to all of them in a given order as fast as possible. Our most efficient method has proven to be robust and successful, since our robot won the competition using the final vision systems described in this paper.**

***Index Terms*— number detection, number classification, real-time, robotic vision,**

I. Introduction

Automatic sign detection is an essential task for robots working in environments built for humans: Signs can serve as landmarks and convey messages for both humans and robots. In this work we try to find the best system for detecting a set of number signs used for marking subgoals in the 2010 "SICK robot day" robotics challenge. The goal of this challenge was to build a robot that autonomously finds and drives to 10 number signs in a given order within at most 10 minutes. Hence the system had to run on an on-board processor of a mobile robot and had to provide real-time performance.

Even though the system presented here solves a very specific task, there are several domains of research which are related to the problem at hand: recognition of text on certain objects such as license plates [5, 11, 9] and containers [12], video OCR [16], and text detection in natural images [15, 23, 20]. A few examples for the latter field of research are Liang et al. [14], who give an overview of document analysis from camera-captured images. Yamaguchi et al. in [22] address the problem of recognizing phone numbers in natural images. Both take advantage of the fact that there is more than a single character and that characters usually arranged in lines. In contrast, de Campo et al. [7] focus on recognizing single characters in natural images, which is more similar to our problem at hand.

Another related area of research is the detection of signs, e.g. traffic signs. In comparison with video OCR, sign detection is a more challenging problem due to the requirement of coping with highly dynamic settings in real-time. Note, however, that the task of sign detection is facilitated by the fact that almost all traffic signs are either colored in or at least framed in an easy-to-detect signal color to make them easier to spot. This also implies that there is usually a distinct edge around their boundary, which renders their detection less demanding and makes color and shape information popular features for detecting traffic signs [8].

Fig. 1. Our robot driving towards sign number 5 during its winning run at the "SICK robot day" robotics challenge 2010.

For the "SICK robot day" competition, it was not totally clear whether there would be an always visible boundary that could be used for sign detection. Hence, we could not apply car number extraction techniques based on color segmentation [13] or rectangular area detection [11].

The contribution of this work is to provide a framework which addresses these problems in real-time. As shown in Sect. V the proposed method allows for robust number detection and classification in a highly dynamic environment.

The remainder of this paper is organized as follows: In Sect. II we describe the 2010 "SICK robot day" robotics challenge. Sect. III provides details of our experimental setup. In Sect. IV we describe four different number detection and classification approaches we implemented. The results are presented and discussed in Sect. V. Finally, Sect. VI gives conclusions.

II. The 2010 "SICK robot day" Challenge

The 2010 "SICK robot day" challenge was an international robotic race for universities, colleges and other research groups. Participating teams were required to build and program an autonomous mobile robot that is able to explore and navigate an unknown environment, find and recognize landmarks given by number signs, and reach all of them in a given order as fast as possible. The organizers originally announced that the challenge could take place either indoors or outdoors in

course 30×40 meters in size with obstacles that can easily be detected by laser range finders. As the challenge had to take place indoors in the end, however, the actual course was circular-shaped with a diameter of only approx. 15 meters. In addition to static obstacles that were moved by the organizers between consecutive runs, there was always a second robot competing on the course at the same time. Collisions with static objects would lead to a penalty of 5% of the total time and collisions with the other robot to disqualification. We also knew in advance that the number signs to be found consisted of black numbers from zero to nine printed in size 37×58 cm on white rectangular boards. The robots had to find and reach these number signs in a given order. The quantity of number signs reached in the correct order and the time required for this determined the ranking of all participating robots. Note that it would be fatal to miss a number sign early in the given order: Signs correctly reached after one was missed would not count towards a robot's score.

III. Experimental Setup

A. The Mobile Robot

The mobile robots used for these experiment are custom-built based on a remote-controlled monster truck "E-MAXX" by Traxxas. Among several other sensors they contain three cameras: The main camera is a forward-looking AVT Marlin mounted on a pan unit so it can keep a number sign focused while driving around corners. It captures grayscale images sized 780×580 at a rate of 30 Hz. We additionally employed two Firefly MV cameras by Point Grey Research looking left and right, hoping to find more signs due to the larger total area of view. These two cameras grab grayscale images sized 640×480 at a rate of 7.5 Hz. All processing is done on a Mini-ITX computer featuring a 2.26 GHz Core 2 Duo Mobile CPU mounted on the robot. The number sign recognition module is only one of many computationally demanding modules running on the computer at the same time.

B. Data

We have a total of 29 logfiles available of our robot driving in different environments with number signs recorded on 7 different days in different locations, indoors and outdoors. We split these logfiles into a training set (21 logfiles), used to train the classifiers of sections IV-D and IV-A, and a validation set (8 logfiles). From each logfile we consider only one camera image per second to avoid processing large numbers of similar images. This still leaves us with 6179 and 2292 images, respectively, in the training and validation set. We semi-automatically labeled all images in both sets as we needed correctly labeled training data for learning our classifiers and validation data for the performance analysis in Sect. V.

IV. Number Sign Detection Algorithms

In this section we present four iterations of our sign recognition software evolved during the preparation of the "SICK robot day" challenge of 2009 and 2010. The problem to be solved is detecting black number signs printed on an otherwise white board in images captured by a camera mounted on a moving robot.

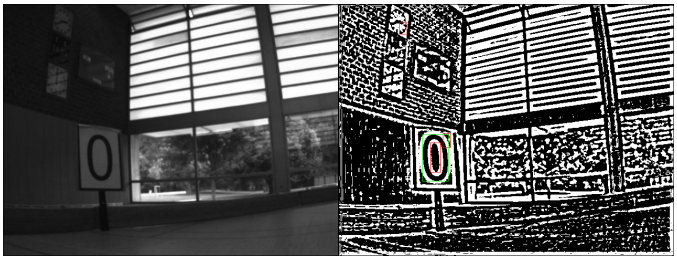

Fig. 2. Image as seen by the robot (left) and binarized image (right) with blobs classified as numbers (marked in green) and as non-numbers (marked in red).

We first describe the application of the object detection framework by Viola et al. [19] to our problem in Sect. IV-A: A general-purpose approach, which is well-known for its good performance detecting mainly faces, but can also be trained to detect arbitrary objects like soccer balls [18]. For the remaining three approaches presented in this work, we decompose the task at hand into three subproblems: Thresholding or binarization, connected component labeling, and classification of the connected components. We deem this decomposition reasonable as we try to find black numbers printed on white background: Proper binarization retains this property in the resulting binary image and the numbers printed onto white signs become black blobs which can be easily extracted using connected component labeling. Thresholding is the task of classifying image pixels as either dark (represented by the value 0) or light (represented by 1). The obtained binary image provides the basis of the following connected component labeling process. Connected component labeling identifies connected regions and assigns each connected region a unique label. In the context of number detection, region detection is important since each number is represented by connected components in the image. After the labeling process we obtain a set of connected regions (or image patches) containing both numbers and non-numbers. For identifying the respective number and filtering out non-numbers, a classifier has to be established.

The approach described in Sect. IV-B relies on many readily-available methods, e.g. by OpenCV [2] and pattern matching for classification. The one described in Sect. IV-C speeds up the binarization step by using an integral image, connected component labeling using flood fill, and classification using 1-nearest neighbor. In our final approach described in Sect. IV-D, we again try to speed up connected labeling using a run-based approach and improve classification by training an artificial neural network.

A. The Boosted Cascade Classifier (Viola&Jones) Approach

This approach is based on the boosted cascade detector by Viola et al. [19] used for real-time face detection. They use the AdaBoost algorithm to train a classifier which then decides in a the recall phase whether an image patch shows a face or not. An image is scanned for faces by applying the classifier on a window that is moved across the image

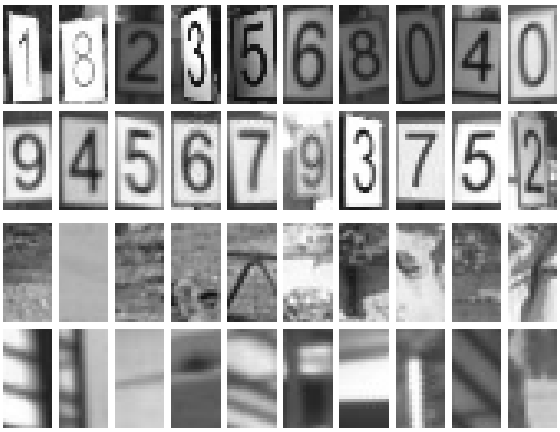

Fig. 3. Patches used for training the classifier.

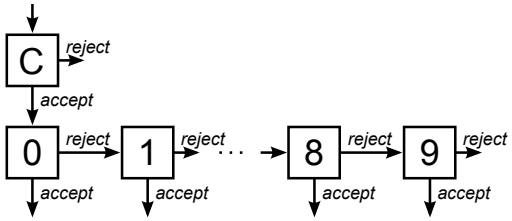

Fig. 4. Structure of the classifier composed from multiple cascade detectors. All image patches must pass a common stage (C) to get examined by the classifiers specific for each single number, going from 0 to 9. Patches that are rejected by either the C stage or the 9 classifier will be considered as non-numbers.

in different scales. Window positions at which the classifier predicts a face are memorized to produce the final detection result for the image. One crucial idea in [19] is the introduction of a so called *attentional cascade*, which rapidly improves the speed of the overall detector. A classifier consists of a cascade of simpler classifiers, or layers, which are applied successively. It starts with the simplest layer, which is designed to be quickly evaluated and highly sensitive, allowing a high false positive rate. This layer can already reject many non-faces, so the remaining layers focus on the image patches accepted by the first layer. The whole classifier only accepts an image patch as a face if it has been predicted positive by all layers.

Originally published as a fast and robust face detector, the system can be trained with arbitrary objects. We created a number detector by training the system with patches of number signs that were cropped out of our log files (see also section III-B). Examples of the image patches are shown in figure 3.

Since a single detector can only distinguish between two classes, usually objects and non-objects, we arrange multiple detectors to create a classifier which is capable of telling not only if a number is found, but also which number is predicted (see Figure 4). The first stage has been designed to quickly select patches which are candidates for a number and reject the majority of image patches, which do not have to be processed further. Only patches that pass the first stage are examined more sophistically by the classifiers sensitive for a certain number. By using a first stage common for all numbers we extended the idea of the attentional cascade described in [19], overcoming the burden of applying 10 different classifiers, one for each number, to all window positions and scales when scanning an image. As shown in 4, the structure of the additional processing is linear and relies on a robust response of the detectors that are specific for each number, e.g. if the "one"-classifier erroneously predicts a one instead of a nine, the "nine"-classifier never gets a chance to correct this. However, due to processing speed and ease of implementation this structure has been chosen to be used as opposed to a more sophisticated approach.

1) Training: The stage common for all number candidates consists of a 10 layer cascade trained with an initial set of 1472 images of all 10 numbers and 12252 images of non-numbers, i.e. the images were used for training the first layer of the cascade. For the last layers the bootstrapping method described in [19] is used, taking only those images for training that have been predicted positive by the first layers, and refilling the non-number set by gathering false positives from large images that do not contain numbers. Gathering is done by scanning these images with the classifier created so far, consisting of the former layers. The detection rate is set to 0.97 per cascade layer, the false positive rate is set to 0.4 per layer, which means that the common stage already rejects about $1-0.4^{10} = 99.990\%$ of all non-numbers. The classifiers for a certain number are trained with 112 to 193 positive examples and initially 12252 negative examples (non-numbers), also using a detection rate and false positive rate of 0.97 and 0.4 per layer, respectively. Bootstrapping is applied for images that do not contain the certain number, i.e. all other numbers are present. First training results show that this is necessary to ensure a high specificity of the classifier. Training is done until the non-number training set cannot be refilled with false positives, which results in classifiers consisting of 13 to 22 cascade layers.

B. The OpenCV Approach

For our second approach, we focused on achieving real-time number detection by using basic and fast image processing methods. We therefore used standard algorithms, e.g. adaptive thresholding and pattern matching, as implemented in the OpenCV library.

1) Thresholding: Depending on the lighting conditions, the setup of the camera and the captured scene, the detected gray value of the white board background or the black number is highly variable. Methods separating pixels into fore- or background by globally adjusted thresholds are therefore not applicable. In an adaptive approach, this threshold is determined dynamically taking spatial variations in illumination into account. For example, Wellner [21] employs a moving average technique in which the mean $\overline{m}$ of the last s visited pixels is considered. If the value of the current pixel is t percent less than the average $\overline{m}$ then this pixel is assigned the value of 0 (dark) and 1 (light) otherwise. Since this approach depends on the scanning order of the pixels and the neighborhood samples are not evenly distributed during the average calculation, Bradley et al. [1] adopt an alternative

approach for determining the threshold. They compute the average of a $s \times s$ window of pixels centered around the pixel under consideration and define a certain percentage of this average as the threshold value.

We used the implementation of the described adaptive thresholding approach from the open source image library OpenCV with a dynamic window size of 15×15 pixels to assign each pixel to either light background or dark foreground. Although this setup works fast and stable for most conditions, it has major difficulties detecting very close numbers. There, the dynamic window is smaller than the line width of the number, hence the inner number area is randomly chosen depending on the image noise. The computation time of the adaptive threshold, however, increases with the window size, thus a larger window is no option to deal with this problem.

2) Component Labeling: We applied a contour-based approach [4] [1] to obtain the connected components of the binary image. The algorithm identifies the components by tracing a contour chain for each connected region. Since we were only interested in blobs which are likely to be numbers, we applied several filters with experimentally determined thresholds to discard some of the extracted components, which saves computation time in the classification step. In our case, numbers are higher than wide, so we removed all components with a aspect ratio greater than 1. Since numbers have usually straight contours, we only kept blobs with a ratio of perimeter to diagonal between 2 and 10. Furthermore, the relative line width of the numbers is equivalent to a certain value, so we discarded blobs with a ratio between blob area and bounding box area lower than 0.1 and greater than 0.5.

3) Classification: We based the classification task on a k-nearest-neighbor classifier (KNN) with $k = 1$. Since the image patches are two dimensional and vary in size whereas the KNN classifier requires one dimensional inputs of constant length, the connected regions had to be preprocessed. For this we scaled every contour chain to fit in a 16×16 pixels bounding box. Now the OpenCV method *cvWarpImage* was used to crop this bounding box from the image. Then we set all not blob related areas of this patch to the background color and scaled the pixel values of the patch linearly to fill the interval between 0 to 255. Finally, we determined the nearest neighbor of the extracted patch by calculating a normalized distance between the patch and every patch of a labeled reference set. In our case, the reference set consists of images of the numbers from zero to nine rendered with rotation angles between $-15°$ and $15°$ in steps of $5°$. Given two image patches p_1 and p_2, the normalized distance is calculated by $d = \frac{\|p_1 - p_2\|_{L2}}{255 \cdot \sqrt{A}}$, where $\|.\|_{L2}$ is the L2 norm, calculated by the OpenCV method *norm*, and A is the area of the patch. If the minimal distance between the extracted patch and a training patch is lower than a threshold of 0.33, the label of the training patch is assigned to the patch. In the other case, the patch is classified as a non-number. The threshold was experimentally chosen to yield a good ratio between correct detections and false positives.

[1] We used the open source library cvBlobsLib which implements [4] and is based on OpenCV: `http://opencv.willowgarage.com/wiki/cvBlobsLib`.

C. The Accelerated k-Nearest Neighbor (kNN) Approach

The third approach aimed at reducing the run-time complexity of each individual step by means of several modifications: an integral image-based adaptive thresholding approach, connected-component labeling using flood fill, and accelerated k-nearest neighbor classification.

1) Thresholding: Using integral images [6, 19], the average calculation for adaptive thresholding can be accomplished in constant time, rendering the approach of [1] practical for real-time applications. Given an image I with pixel intensities $I(x, y)$ at pixel locations (x, y), an entry within the integral image $\mathcal{I}(x, y)$ stores the sum of all intensities which are contained in the rectangle defined by the upper left corner (0,0) and the lower right corner (x,y). Then, the average intensity $\bar{i}(x, y)$ of an arbitrary rectangular area of I given by (x_{ur}, y_{ur}) and (x_{ll}, y_{ll}) can be calculated as:

$$\bar{i}(x, y) = \frac{\mathcal{I}(x_{ur}, y_{ur}) - \mathcal{I}(x_{ur}, y_{ll}) - \mathcal{I}(x_{ll}, y_{ur}) + \mathcal{I}(x_{ll}, y_{ll})}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}, \tag{1}$$

where x_{ur}, y_{ur} and x_{ll}, y_{ll} denote the upper-right and lower-left x- and y-coordinates of the given rectangle, respectively. Equation (1) shows how the average can be computed efficiently. Further acceleration was achieved by choosing a power of two as the window size, which enables the use of a right-shift instead of a division operation.

Adaptive thresholding using an integral image is a two-stage process. At first, the integral image has to be calculated, which can be accomplished in linear time. In the second stage, the original image is binarized by comparing each pixel intensity of I with the average intensity of the rectangular area which surrounds the pixel under consideration.

2) Component Labeling: For the labeling task, we employed a simple seed fill algorithm: Given a certain location in the binary image (x, y), its neighbors are recursively tested whether they are assigned the same binary value as the pixel under consideration. If this is the case the respective neighbor (x^*, y^*) is assigned the same label as the pixel (x, y). Further, this process is recursively repeated to all non-visited neighbors of pixel (x^*, y^*). Recursion stops if the binary value of the neighboring pixel (x^*, y^*) differs from the one of (x, y).

All pixels with the same label assigned define a set of connected components. We further represented the area which was covered by the minimum bounding box surrounding a single connected component set by a binary matrix. Here, a matrix element was assigned the value of 1 if it represented a connected component and 0 otherwise. Finally, the set of all binary matrices established the basis of the succeeding classification step.

3) Classification: Following the classification scheme of the first approach, we also employed k-nearest neighbor classification with $k = 1$ given rescaled image regions as inputs. In contrast to the previous approach, however, these image regions did not originate from the acquired grayscale image but from their respective connected components extracted in the previous step. Further, resampling was achieved by nearest neighbor interpolation yielding rescaled connected components of size 32×32 pixels. In the following step, the

32×32 matrix of binary values was transformed into a 1024 bit sized one dimensional vector by concatenating each individual row of the matrix. Here, the one dimensional vector was represented by a byte array of length 128, $F_n = \{f_{n,1}, \ldots, f_{n,128}\}$, each byte containing 8 pixels of the binary vector. Classification was performed by calculating the Hamming distance between a candidate input vector and each entry of the number training set. Similar to the first approach the training set consisted of unmodified and transformed numbers rotated by an angle between -15° and $+15^\circ$ with step size 5°, yielding 70 training patterns in total. The image in the training set with the minimum distance to the candidate vector determined the classification result. If the Hamming distance exceeded a certain threshold, the considered image patch was defined to be a non-number. We determined this threshold experimentally by choosing a threshold value which resulted in the smallest classification error. Adopting our representation of connected areas, the Hamming distance between two input vectors F_1 and F_2, $H(F_1, F_2)$, can be efficiently computed using addition and XOR operations only: $H(F_1, F_2) = \sum_{i=1}^{128} \mathit{bitcount}(f_{1,i} \otimes f_{2,i})$, where *bitcount* determines the number of 1-bits in the resulting bitstring after the XOR operation. In our implementation, the function *bitcount* was realized using a lookup table covering all 2^8 possible configurations of an input vector entry $f_{n,i}$. We disregarded further acceleration measures such as using 16 bit up to 64 bit integers instead of 8 bit integers as the basis data type for the input vector representation (cf. [17]) or employing specialized instructions of the SSE4.2 instruction set as proposed in [3]. This is because the classification step proved to be fast enough and hence had not to be accelerated.

D. The Artificial Neural Network (ANN) Approach

In the fourth approach we reused the algorithm to compute an adaptive thresholding using the integral image. We notably changed, however, how connected components are extracted and classified.

1) Extracting Suitable Connected Components: We based our connected components labeling algorithm on the run-based two-scan labeling algorithm by He et. al. [10]. This algorithm regards runs, i.e. continuous horizontal line segments with the same pixel value, instead of pixels as the smallest elements of connected components, which promises a significant speedup for binary images with large areas of constant value. It requires extra space for a label image, which assigns the same unique label to all pixels that belong to the same connected component upon completion of the algorithm, and for three arrays of limited size to store temporary information about so-called provisional label sets.

The original algorithm by He traverses the image twice: In the first scan, the algorithm traverses the image pixel-wise to extract runs. If a run is not connected to any previous run, a new label is assigned to all of its pixels. Otherwise it inherits the label of the connected run. In case there are two or more runs with different labels both connected to each other, they are merged to form a provisional label set. This merging operation can be implemented efficiently since all information about provisional label sets is in compact data structures.

In the second scan, the original algorithm traverses the whole image again to compute the final label image, replacing all provisional labels with their final ones.

In our case, however, we do not need the complete final label image. As we usually only need to extract a small number of connected components, we modified He's algorithm in the following way: In addition to the data structures proposed by He, we also store the bounding box of each provisional label set. Thus, we can disregard connected components depending on the dimension of their bounding box and extract only connected components which are likely to be numbers. Once we have selected a smaller number of connected components as candidates, we extract these efficiently, based on their bounding box instead of traversing the whole label image for a second time.

2) Classification of Patches: To further improve classification performance over the previous kNN classifier without requiring much more computation time, we employ an artificial neural network (ANN) to classify image patches.

The neural network has to decide to which of the 11 classes a patch belongs: It is either one of the 10 numbers or "not a number". We implemented this using the 1-of-11 target coding scheme, which requires one output neuron per class: For each input, we expect the output neuron corresponding to its class to return 1 and all others to return 0.

The structure of our neural network is a multilayer perceptron with 160 input units (one per pixel of patches resized to 10×16) and only one hidden layer consisting of 20 hidden units. All units of our neural network use the logistic activation function, except the output layer which uses the softmax activation function, so we can interpret the output of each output unit as the posterior probability of the input patch belonging to its corresponding class.

We used the Stuttgart Neural Network Simulator (SNNS) [24] to train and evaluate the neural network because of its ability to automatically generate efficient C code evaluating a previously trained network using its snns2c module.

We trained the network using standard backpropagation on a large dataset containing real-world patches extracted from logfiles in which a robot moved around environments that contained all 10 signs. The classifier of the previous approach supplied us with a preliminary labeling of this dataset and we only needed to manually verify or correct this labeling afterwards. Since these logfiles contain a relatively small number of patches that correspond to actual numbers compared to a large number of non-number patches, we synthetically generated more training patches by warping images of numbers using different perspective transformations. We relied on early stopping to prevent overfitting to the training data and employed a 10-fold cross validation on the training data to find that the average test error was minimal after 53 epochs of training.

By adding a large number of synthetic training patches, we significantly changed the ratio between number and non-number patches and thus the prior class probabilities assumed by the network. We need to correct this by scaling the posteriors returned by the network with the ratio of relative frequencies of each class before and after adding synthetic

TABLE I

DETECTION PERFORMANCE OF THE OPENCV, KNN, ANN AND VIOLA/JONES APPROACHES. THE FIRST TWO VALUES ARE RELATIVE TO THE NUMBER OF TRUE NUMBER SIGNS IN THE VALIDATION SET.

	OpenCV	kNN	ANN	Viola/Jones
true positives in %	35.0	34.6	49.7	**64.0**
false negatives in %	65.0	65.4	50.3	**36.0**
false positives per frame	0.012	0.011	0.012	**0.007**

TABLE II

CLASSIFICATION PERFORMANCE OF THE OPENCV, KNN, ANN AND VIOLA/JONES APPROACHES.

	OpenCV	kNN	ANN	Viola/Jones
correct detections in %	**100.0**	99.8	99.6	97.6
wrong detection in %	**0.0**	0.2	0.4	2.4

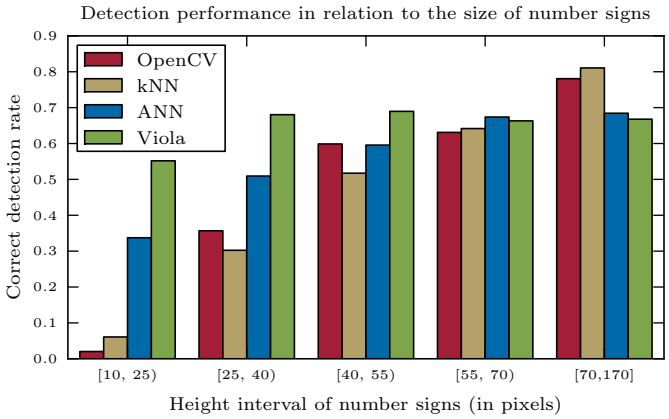

Fig. 5. Detection performance depending on the visible size of numbers. The correct detection rate in this plot is equivalent to the true positives rate from table I times the correct detection rate from table II.

TABLE III

PROCESSING TIME PER FRAME IN MILLISECONDS.

	OpenCV	kNN	ANN	Viola/Jones
thresholding	3.5 ± 0.4	2.5 ± 0.4	**2.4 ± 0.3**	-
segmentation	9.3 ± 5.3	9.3 ± 1.3	**2.3 ± 0.7**	-
classification	0.4 ± 0.3	0.2 ± 0.4	**0.1 ± 0.2**	-
overall	13.2 ± 5.5	12.0 ± 1.6	**4.8 ± 1.0**	193.5 ± 73.3

data.

Since we were more concerned about false positives and wrong detections than about false negatives, we applied a loss function that assigns a loss of 1 to false negatives and a loss of 10 to false positives and wrong detections. Classes are then chosen to minimize the expected loss.

V. EXPERIMENTAL RESULTS

We evaluate all four of the approaches described in chapter IV using the validation dataset described in III-B with regard to both classification performance and computation time.

A. Detection Performance

In order to compare the detection performance, we decouple the two problems of detecting a general number sign and the classification of detected signs. Each true number sign is either correctly detected (true positive) or missed (false negative). Detections that do not correspond to a true number sign are counted as false positives. Since numbers as found by the described detectors might slightly differ in position and size from the ones labeled by humans, we consider two labels to coincide if the intersection the area of both covers at least one-fourth of each label's individual area. The resulting detection rates can be seen in table I. Rates of the OpenCV and kNN approach are almost identical because they mainly differ in details of their implementation. The ANN approach achieves many more true positives while keeping the number of errors at a similar rate due to the better classification performance of the artificial neural network. The Viola/Jones-based detector performs significantly better than all other methods.

The classification performance, i.e. the percentage of correctly detected signs that were also assigned the correct label, is shown in Table II. This is consistently high with the exception of the Viola/Jones approach which tends to confuse number signs more often than the others.

The overall ratios of correctly recognized numbers as reported in Table I might seem low to the reader. This is because many of the number signs visible and tagged in our log files suffer from motion blur or are far away from the robots. In combination with the artifacts introduced by the discretization in the thresholding step of the first three approaches, this makes classifying very small numbers correctly a hard problem. Another important issue to keep in mind is, that for our application, low ratios of false positives or incorrectly classified numbers are much more important than higher recognition rates: As long as our robot has not seen the next number it needs to reach, it can simply keep exploring until it will find it eventually. If, on the other hand, it drives towards an incorrectly recognized (false positive) sign and decides it has reached it, there is no way to recover from this bad decision and the competition would be lost.

Fig. 5 shows how detection performance of different approaches depends on the size in pixels of the number sign as seen in the image. While the performance is comparable for large numbers, the most significant differences can be observed for small numbers. Good performance for small numbers, however, is essential, as we want to find number signs early on when the robot is still far away. Once the robot is close to a sign, we hope to have an accurate estimate of its position and can afford to not detect it from time to time.

B. Computation Time

Table III lists the computation time required by the described methods. We measured these times by running the number sign recognition systems on our robot in batch mode, i.e. with no other jobs running at the same time and utilizing only one of the two cores.

It is worth noting that our own implementation of adaptive thresholding using the integral image is significantly faster

than the implementation provided by OpenCV on a window size of 15×15 pixels. The most important improvement with respect to computation time, however, comes from using a run-based algorithm for segmentation as described in the ANN approach. While using runs might not help that much when processing images with many very small connected components, e.g. documents with very small letters, binarized images of natural scenes in our experience always contain at least some uniform regions, in which using runs improves performance considerably.

VI. Conclusions

We have implemented and compared the performance of four methods for detecting number signs on a mobile robot and managed to create a robust system that requires less than 5 ms per image. Using this system, our robot is able to detect number signs in real-time even when processing images from three cameras at the same time, while still leaving enough computational resources for other modules running on the robot. Specialized algorithms using efficient simplifications, e.g. binarization, are key to image processing on mobile robots in high-speed environments like robot races. Unfortunately, they usually require easily exploitable properties. For the task described here, we were able to exploit the fact that there is a white region around the black numbers to be detected. This allowed us to implement a robust number sign detection algorithm that is several orders of magnitude faster than general-purpose object detection algorithms.

Our robot described in III-A won the 1^{st} place in the "SICK robot day" robotics challenge 2010 using the artificial neural network-based number sign detection system. A video of our first run at the SICK robot day challenge 2010 can be found at `http://www.youtube.com/watch?v=tpILYAUGxfU`.

Acknowledgment

The authors would like to thank all other colleagues of our team that participated in the "SICK robot day" robotics challenge: Karsten Bohlmann, Yasir Niaz Khan, Stefan Laible and Henrik Marks

References

[1] D. Bradley and G. Roth. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, 12(2):13–21, 2007.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision - ECCV 2010*, volume 6314 of *Lecture Notes in Computer Science*, chapter 56, pages 778–792. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[4] F. Chang. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220, February 2004.

[5] X. Chen, J. Yang, J. Zhang, and A. Waibel. Automatic detection and recognition of signs from natural scenes. *IEEE Transactions on Image Processing*, 13(1):87–99, 2004.

[6] F. C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 84)*, pages 207–212, New York, NY, USA, 1984. ACM.

[7] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[8] A. de la Escalera, J. M Armingol, and M. Mata. Traffic sign recognition and analysis for intelligent vehicles. *Image and Vision Computing*, 21(3):247 – 258, 2003.

[9] K. Deb, H.-U. Chae, and K.-H. Jo. Vehicle license plate detection method based on sliding concentric windows and histogram. *Journal of Computers*, 4(8):771–777, 2009.

[10] Lifeng He, Yuyan Chao, and K. Suzuki. A run-based two-scan labeling algorithm. *Image Processing, IEEE Transactions on*, 17(5):749–756, May 2008.

[11] C.G. Keller, C. Sprunk, C. Bahlmann, J. Giebel, and G. Baratoff. Real-time recognition of u.s. speed signs. In *IEEE Intelligent Vehicles Symposium*, pages 518–523, Eindhoven, June 2008.

[12] S. Kumano, K. Miyamoto, M. Tamagawa, H. Ikeda, and K. Kan. Development of a container identification mark recognition system. *Electronics and Communications in Japan, Part 2*, 87(12):38–50, 2004.

[13] M. Lalonde and Y. Li. Detection of road signs using color indexing. Technical Report CRIM-IT-95, Centre de Recherche Informatique de Montreal, 1995.

[14] Jian Liang, David Doermann, and Huiping Li. Camera-based analysis of text and documents: a survey. *International Journal on Document Analysis and Recognition*, 7:84–104, 2005. 10.1007/s10032-004-0138-z.

[15] J. Ohya, A. Shio, and S. Akamatsu. Recognizing characters in scene images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16:214–220, February 1994.

[16] T. Sato, T. Kanade, E. K. Hughes, and M. A. Smith. Video OCR for digital news archive. In *Proceedings of the 1998 International Workshop on Content-Based Access of Image and Video Databases (CAIVD '98)*, CAIVD '98, pages 52–60, Washington, DC, USA, 1998. IEEE Computer Society.

[17] S. Taylor, E. Rosten, and T. Drummond. Robust feature matching in 2.3μs. In *IEEE CVPR Workshop on Feature Detectors and Descriptors: The State Of The Art and Beyond*, June 2009.

[18] André Treptow, Andreas Masselli, and Andreas Zell. Real-time object tracking for soccer-robots without color information. In *European Conference on Mobile Robotics (ECMR 2003)*, pages 33–38, Radziejowice, Poland, 2003.

[19] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, May 2004.

[20] K. Wang and J. A. Kangas. Character location in scene images from digital camera. *Pattern Recognition*, 36(10):2287–2299, October 2003.

[21] P. D. Wellner. Adaptive thresholding for the digitaldesk. Technical Report EPC-1993-110, 1993.

[22] T. Yamaguchi, Y. Nakano, M. Maruyama, H. Miyao, and T. Hananoi. Digit classification on signboards for telephone number recognition. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 359–363, 2003.

[23] J. Yang, X. Chen, J. Zhang, Y. Zhang, and A. Waibel. Automatic detection and translation of text from natural scenes. In *Proceedings of International Conference on Acoustics and Signal Processing 2002(ICASSP '02)*, Orlando, FL, USA, May 2002.

[24] Andreas Zell, Niels Mache, Ralf Hübner, Günter Mamier, Michael Vogt, Michael Schmalzl, and Kai-Uwe Herrmann. SNNS (stuttgart neural network simulator). In Josef Skrzypek, editor, *Neural Network Simulation Environments*, volume 254 of *The Springer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Norwell, MA, USA, February 1994. Chapter 9.